2018 IEEE 11th International Conference on Cloud Computing

Real Time Metering of Cloud Resource Reading Accurate Data Source Using Optimal Message Serialization and Format

Tariq Daradkeh*, Anjali Agarwal*, Nishith Goel† and Marzia Zaman†
*Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada
Email: {t_darad, aagarwal}@encs.concorda.ca
†Cistech Limited, Ottawa, Canada
Email: nishith@cistech.ca, marzia@cistel.com

***Abstract*—In this paper the technology of collecting the logs and the logs message format is considered in addition to investigating multiple log data sources as an input to the cloud management system. A comparison between message exchange technologies (JSON, XML) is evaluated with the latest message format technology (Google Protocol Buffer) when used in combination with the message transmission protocols (XML-RPC, REST, Network Socket). In addition, the sampling rate that gives the accurate reading of resource usage is investigated, which is used to select among different log data sources to achieve the accurate log update time. Logs sampling rate of 1.0 second is found to be the best with "xentop"data source. The result of the experiment shows using Protocol Buffer with Socket protocol gives the best results in reducing message size. Network socket with JSON gives the best processing delay and traveling time.**

***Index Terms*—Protocol Buffer, XML, JSON, REST, XML-RPC, Data Source, Log Sampling Frequency, Log Accuracy.**

I. Introduction

Cloud monitoring and logging play an important role in cloud management system. Logs are generated from multiple data sources in cloud environment such as Physical Machines (PMs), Virtual Machines (VMs), network devices and the cloud running applications services. Each single cloud component has numerous logs and events data source generators, which could be generated from low level frameworks such as Hypervisor (the virtualization software) up to the running applications on top of the VMs. Logs must be captured and filtered or aggregated based on the need of the cloud management system, which can be used to discover cloud status and future actions for cloud management system. Cloud management models can manage the resources dynamically via reconfiguring cloud resources using numerous approaches such as VM migration, storage migration, network reconfiguration, load balancing, affinity relation, hot spot avoidance, server consolidation, holistic approach and fault tolerance [1]. These resource reconfiguration methods can be generalized in a closed loop control model proposed as following [2]: Cloud resource monitoring, logs analyzing, resource planning and finally executing the planned decisions. Figure 1 depicts the closed loop cloud management system.

Each cloud management layer focusses on a certain type of logging and cloud resource metering that makes cloud monitoring features jointly to monitor specific resources and diverge on the other. For example, in [3] the authors suggested a cloud ecosystem that classifies cloud management into layers to build a private cloud. The classification is based on cloud operation and usage started from top to lower layer: a) Cloud consumers that consist of end users, other cloud providers and service providers, platform and software as a service; b) Cloud management that concerns about monitoring and controlling virtualization in cloud infrastructure as a service; c) Virtual infrastructure management that supports primitives to manage VMs across multiple physical machines; d) VM manager that considers the virtualization software (hypervisor or virtual machine manager or monitor). In all cloud management layers, the monitoring tools are a standalone component. For instance, in cloud infrastructure management such as OpenStack [4] there are five monitoring tools used for different purposes where CEILOMETER is used for data collection service in OpenStack telemetry, CLOUDKITTY is used in billing by converting metrics into price, MONASCA is an open source monitoring as a service, AODH is an alarm service, and finally PANKO is a metadata and event collector.

OpenNebula has its own built-in monitoring and logging tools that work to collect events and pull and push, where in OpenNebula 4.2 and later version uses the udppush as a default mode in monitoring resources [5]. Eucalyptus use cloud watch as monitoring tool [6]. Many of open source projects that are developed to monitor cloud services focus on numerous parts of cloud environment. Some of these tools are sophisticated to monitor specific services such as billing, elasticity and power consumption. Cloud monitoring tools [7] vary in implementation and the ways to get the cloud system logs. Most of the cloud monitoring system developers focus on the monitoring system design and its components and how they communicate. However, the technology of collecting the logs and the message logs message format is not considered in implementation, in addition to using a common log source as an input to the cloud management system. The contribution of this paper is to investigate the technologies of message com-

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00067

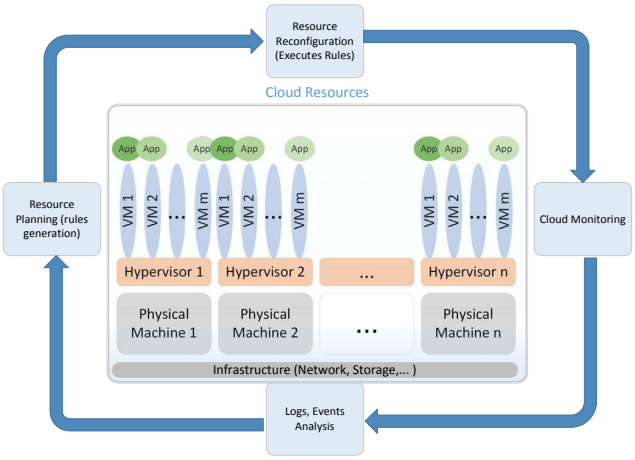

Fig. 1: Closed Loop System for Cloud Management

munication and format in cloud monitoring and management system. The best logs message format and encoding must be used to reduce the message size and delay overhead. Selecting logs data source to collect performance metrics in a good sampling time is very important to get accurate logs values. This paper is organized as following: Section II discusses the background of cloud management, logging streaming and log message format. In Section III message data model, message format and communication protocols are discussed. Section IV explains the proposed monitoring module. Section V shows the cloud setup implementation. Finally, conclusion and future work is presented in Section VI.

II. Related Work

We classified the related work into three categories about cloud monitoring and logs collection covering the monitoring framework, log requests communication and log message format with transmission protocols.

Category one discusses the monitoring framework design and components. There are a variety of number of cloud management software and tools, which rely on resource monitoring to trigger the right action. OpenStack, CloudStack and Eucalyptus are examples of cloud management system for infrastructure (IaaS) that belong to open source software group where the components of each system are developed independently. In OpenStack cloud computing [8] Nova is the part that is responsible for provisioning and managing VMs. Nova needs information about physical and logical resources to gauge the resource utilization and performance. The critical issue is to choose the attributes of the cloud infrastructure and processes that must be monitored.

The solution for this is to propose tools and parameters that can monitor the cloud infrastructure that will be used by Nova cloud OpenStack management system. These attributes are applied on three various kinds of monitoring software for different layers of cloud service model. A monitoring list for cloud infrastructure, which identifies the cloud resources is considered by the framework in [8]. They focus on four types of monitoring as following: 1) Compute infrastructure, which maps the physical infrastructure resources to the virtualized resources while considering the elastic feature of resource provisioning and achieving service isolation with fault tolerance and high availability, 2) Infrastructure usage that measures usage of physical resources such as CPU and RAM, 3) Background processes that run in background, and have numerous hidden services such as cloud orchestration, live migration, and policy enforcement, and 4) OS process utilization, which is related to measuring the cloud processes usage of the VM resources such as CPU, memory and network. Nova engine continues to track the functionality and attributes of cloud system to get the system status, based on which the CPU resources are provisioned by Nova for VMs (compute as a service). Nova has six components: Nova-api, queue, Nova-db, Nova-conductor, Nova-scheduler and Nova-compute. These independent components cooperate with each other to manage the VM and compute resources. A prototype monitoring engine has been developed for Nova considering the monitoring lists of cloud service model layers and is applied on third party monitoring tools for Nova, which are Zenoss, Zenpack, CollectD and Zabbix. These tools are used to monitor the four types of logs mentioned above. Some of these tools can read logs from specific data source, whereas all of these read logs from standard data source.

The authors of [8] also focus on Nova cloud system orchestration based on the aforementioned monitoring framework that classifies the cloud log data sources into four types. These types of data source explain the resource that must be selected, the background services that must be monitored, and resource usage that must be read. Each class of log reading is used to feed the Nova engine to orchestrate cloud system, and to maintain policy enforcement and cloud management actions. Cloud orchestration and provisioning is done by relating cloud status with resource availabilities such as hosts, hypervisors, virtual machine and the tenant usage. The Nova is using standard API such as Libvirt and XAPI to communicate with the hypervisor such as KVM and XenServer to apply resource reconfiguration. These tools use XML-RPC as transport protocol with XML message format. Another work focusses on private cloud monitoring management architecture [9] that covers IaaS cloud service model layer. Cloud management use the logs by decoding the logs to trigger the actions, then store the logs and report them to the end user or to the cloud administrator. The authors in [8] and [9] do not consider the transportation technology and cloud message format.

Category two explains logs initiation transaction in cloud environment from producer and consumer perspective. In [10] authors focus on the mechanism of reading cloud monitoring logs. The logs are read either by the cloud management system requesting the logs data source through pull action or the data source just sends the log states to the cloud manager by push action. A hybrid model combines both push and pull to retrieve cloud system logs as a user oriented resources monitoring tool. The cloud logs can be initiated by cloud manager as

a pull for data as needed, or it can be continuously send from cloud infrastructure components as a push for the logs. Both the methods have pros and cons. The push will update the monitoring engine with system status changes which will give a consistent view about cloud system but with higher overhead in the network and logs processing. On the other hand, the pull will request the logs only once it is needed which will reduce the transaction but with less accurate actions. The combined approach named Push&Pull (P&P) is shown to be the best to work in a monitoring tool [10]. The logging type can determine the best mechanism to request the logs, for example notification alert system chooses push method rather than pull whereas critical applications such as database use pull method. However, it is best to choose a combined model for some of the applications like scientific application. The idea of P&P comes from Grid Monitoring Architecture (GMA) that considers a producer and consumer relation for the logs where producer sends the logs (unsolicited mode) or a consumer can request the logs (solicited mode). The logging producer must utilize log change to reduce sending the repeated logs based on a certain threshold. There is no need to send the logs if the change is smaller than a specified threshold. Other approaches for producer to decide when to send the logs (push method) proposed by [11] are Offset-Sensitive Mechanism (OSM), Time-Sensitive Mechanism (TSM), and a Hybrid Mechanism combining both OSM and TSM to eliminate sending unimportant logs. Consumer side mechanisms (pull method) are proposed in [12], [13] to read the logs based on a system change expectation with time to get refreshed data logs. In [14] an adaptive model is used to combine Push and Pull (PaP) and Push or Pull (PoP) in a web structure. A hybrid push protocol for resource monitoring in cloud computing platforms [15] enhances logs push method by combining periodic push with event driven push. It uses exponential weighted moving average to change the logs update trigger to tune logs frequency. Cloud computing has a complex architecture because of its heterogeneity in its components. A Runtime Model for Cloud Monitoring (RMCM) [16] focusses on monitoring platform design for cloud computing layers and functionalities for heterogeneous infrastructure. In all aforementioned works the emphasis is on communication initiator, where consumer must ask for the logs and producer must send the logs, but again the transportation protocol and message format were not considered.

Category three discusses the transportation protocols and message format. The related group of work concentrates on reading raw data from cloud resources using multiple monitoring techniques. The cloud components and its running services are considered as entities that are monitored based on different user's perspective based on the actor's roles and needs. The authors in [17] introduced a new message transport protocol for application communication named Selector Based Transport Protocol (SBTP) for JTangMQ, a message format exchange framework using Java Messaging Service (JVM). This protocol is based on queue message publishing and subscription. Ideally the message exchanges between different local processes use two methods: Message Passing Interface (MPI), and Shared Address Space. Inter-process communication can be done remotely mainly using synchronous message-based connection oriented methods such as Remote Procedures Call (RPC), or using asynchronous connectionless communication such as Message Oriented Middleware (MOM), and JVM. Message Transport Service (MTS) needs Message Transport Protocol (MTP) to handle the message delivery and error check such as HTTP and TFTP. This paper proposed an asynchronous message transport protocol based on UNIX *select()* function where multiple connections can be created mapped to multiple threads. Simple Object Access protocol (SOAP) has a good architecture that combines HTTP and XML to format message exchange. REST was introduced later as a simple model for RPC call using HTTP functions and it uses general message format as XML or JSON. A new asynchronous transport protocol, based on interacting platform JMS integrated with a JTangMQ platform was also proposed in [17]. We note to the best of our knowledge no work has used binary message encoding (protocol buffer) in cloud monitoring considering different transport protocols and log message format and size.

III. Data Model, Message Formats and Communication Protocols

In any communication protocol message design has an important role in exchanging the data between communication parties. Today and with HTTPv2 technology that makes autonomous communication between systems independent from user interaction requires message attribute to be generic that can adapt to any change in the system. Many message formats were developed starting from plain text messages with Tags ID such as XML, then JSON and later binary encoding messages like thrift and protocol buffer. Following sections explain these three types of message format:

A. Data Model and Message Format

1) XML Message Format: XML Message Format: Each field in XML message has two tags to describe. It is easy to construct and has a dynamic nature that can adapt to any change in the cloud environment. Each newly created VMs or added hardware can be appended under the VMs list or Hosts list. The message is sent as a string and needs to be parsed to extract the fields' records.

2) JSON Message Format: JSON message reduces number of tags that are needed to describe the field. This reduces the size of the message. Like XML message, it is easy to construct because it is a text based construction. It has a list or array description for multiple VMs or hosts. The JSON message contains arrays of hosts and VMs fields which can handle the change in cloud setup by dynamically adding the new VM attributes as text.

3) Protocol Buffer Message Format: Google Protocol Buffer known as (protobuf) is a serialization method of structured data message by using binary formatting [18]. Message structure format is described by descriptors based

on Interface Descriptor or Definition Language (IDL), which is used to describe the communication interface between client and server by defining objects as abstract descriptions in terms of their external interfaces.

Google Protocol Buffer has unlimited number of fields types and three types of fields rules as following: required, optional and repeated, which must be specified in the message definition until version 3 release where the compiler did not need to distinguish between the optional and required fields rules. The proto3 release is more generic and effective in message creation. In this work, the message can be either a log or an event from cloud system. The format of the message must be generic and adaptive to message changes based on the system status, for example number of VMs in cloud elastic model can be changed continuously which means a continuous change in message fields and values.

The log message must cover all the needed information about the system status and can be adapted to cloud reconfiguration changes, such as adding 'repeated' rule for each new Host and VM. We design a new *CloudLogsMessage* with two main types of messages: the *HostsLogs* and *VMsLogs*. Each one of them has a structure of array of logs to get the full status about the cloud system including CPUs and vCPUs, Memories, Network cards (physical interfaces) and Virtual Networks (virtual interfaces), Disks and VMs status. These arrays are implemented by 'repeated' rule where each new VM is added to the VMs list. Similarly, the upgraded or increased number of physical host can be handled by just adding Host to the *HostsLogs* list. Therefore, any extra logs that cloud needs can be read by two arrays in hosts and VMs messages using repeated *counter* and repeated *Emesgs* fields.

B. Data Flow Model and Communication Protocol

In this work, the data flow model is implemented based on the three initiation methods of logs collecting mechanisms to transfer logs from data source (logs producer) to cloud management system (logs consumer). These methods as mentioned before are push, pull and hybrid combining both approaches. Three types of network communication protocols (Network Socket, XML-RPC, REST) are used to deliver the message between cloud running components and management system. Each protocol supports the three message formats (XML, JSON and protobuf). Our implementation is based on XenServer7.2 hypervisor [19], which has a local API interface called XAPI to control it [20]. This toolstack manages a cluster of virtualization nodes by providing standard API interface for cloud management systems allowing it to monitor and control the virtualization infrastructure service. Figure 2 explains the full details of XAPI components and cloud management systems that it can support. The listening port numbers for communication follow the standard using HTTP and HTTPS (port 80 and 443), which makes XAPI simple to use and integrate with many cloud management systems. XAPI components include common services and functionality for XenServer cluster management such as network configuration, login authentication, filesystem, storage and high availability.

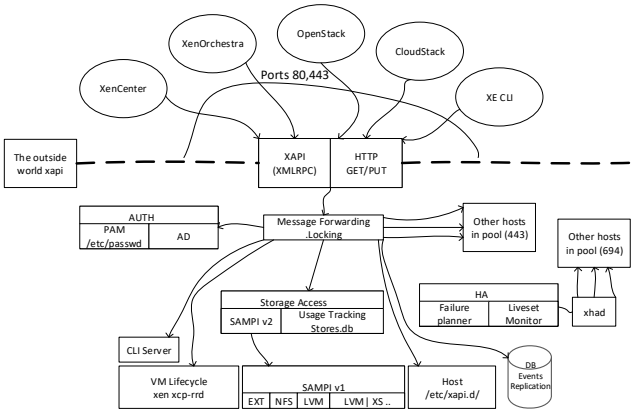

Fig. 2: XAPI Architecture [20]

XAPI uses XML-RPC protocol and XML message format in communication between cloud master nodes and operation node, and web service (REST API) for outside management such as cloud OpenStack framework.

Logs collection starts by gathering the resources usage or events from system data source generator of resource usage statistics. XenServer (special version of Xen hypervisor) is a bare-metal hypervisor which runs directly on top of the hardware. It is like an operating system such as Linux or windows and is considered as a real-time system that is affected by logs generation time managed by generating them in periodic rounds using sophisticated threads. For the XenServer hypervisor XAPI [21], toolstack manages the Xen Cloud Platform (XCP) cluster. The architecture of the system is based on internal communication between cloud cluster nodes components that work locally using 'xl' toolstack or globally using "xe"toolstack. The interaction between XCP nodes is through SSH protocol to exchange control messages. For logs transfer port 80 is used as listener in XML-RPC server communication interface. All communication between cloud services are also over HTTP and XML-RPC in XAPI of XenServer cloud system. The logged report is sent to the master computing node which works as cloud coordinator and management node. There is a direct communication between non-master nodes to allow VM migration or storage migration to transfer the VM memory image or the VM disk mirror.

In XML and JSON the message is a text based that make it flexible and easy to build and change the message data model. However, the message data model design is critical in protocol buffer because any change for the message design needs a recompilation for message language binding class.

IV. Proposed Monitoring Module

For an accurate cloud system monitoring tool, four factors must be considered:

- Logs data source: the resource usage statistics can be generated and captured by different tools, for example CPU usage can be measured in Linux OS by many commands and tools such as (top, ps, mpstat, sar, iostat). But which of these data source is the best to read that

gives an accurate statistic about the CPU usage change. In this work, general cloud system logs are collected with a major focus on CPU usage because it is a good indication of cloud system status. The location where the CPU utilization is read is important for both physical host and VM.

- Sampling rate: the frequency of measuring the resource usage statistics is important to get an accurate indication of the system status. Sampling frequency must consider the resource usage fluctuation in such a way it increases if a high change in frequency happens and decreases if change in frequency is lower.
- Logs change tracking: it is not a good idea to send invaluable information about cloud system to cloud manager. Logs that have same values or have negligible change should be skipped to reduce log processing and network overhead.
- Communication protocols and the message formats: message transport protocol and message format can reduce the delay time and message size in cloud management, which helps cloud manager to make best action for cloud system resources.

In our proposed work, a prototype for cloud monitoring module has been developed and integrated with XenServer hypervisor virtualization framework to read its logs. The proposed module supports the three communication protocols and three message formats as Figure 3 depicts. Our proposed monitoring module satisfies the four design factors, where each part is discussed in detail in the following sections:

A. Cloud Monitoring Message and Protocol

Table I shows a comparison between the three protocols running the three types of message formats, where three message sizes (S-Small, M-Medium and L-Large) are collected to test the effect of message size in message transmission protocol. The table depicts that protocol buffer has better reduction than other message formats, especially for larger message size. Network socket communication has the least transmission delay among other transmission protocols, which is the time to transfer the message from source (producer) to destination (consumer). Processing delay is the time to construct the message with the proper format (XML, JSON and protobuf), then serialize it through the communication channel. It is clear from Table I that JSON has the least processing delay. If response time is important for the cloud provisioning system, using socket with JSON is the preferred communication data model. But if message size is the important factor then protocol buffer with socket communication protocol is the best among the mentioned message formats.

Building the message is implemented by Java class mapper, which transfers the system class entities attributes into message. For XML and JSON we used "com.fasterxml.jackson.dataformat.xml"and "com.fasterxml.jackson.databind"packages where there are objects for XML and JSON class mappers that construct XML and JSON messages. Protocol buffer has a built in

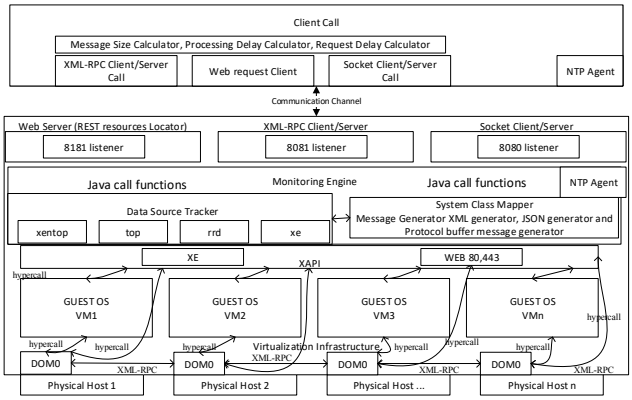

Fig. 3: Proposed Monitoring Module

direct mapper that maps a log class entity into a message. In the receiver side, we again use Java inverse mapper from a message into class entities values. The mapper receives text based XML and JSON messages which rely on DOM Parser for the message, which costs more processing time and code complexity. But the protobuf mapper is less complicated by using class get function, which takes less processing time getting log values from the message. Most of the processing delay in protocol buffer is caused by serialization (binary encoding) whereas XML and JOSN have a very small serialization delay. In XML-RPC and REST protocols message must be transferred into text, because the nature of the protocols is to send text only messages that causes an increase in the message size.

For message initiation, REST protocol can be the best fit to support pull mechanism in logs transmission request, however the other two protocols are easy to deploy for both pull and push methods. In REST a full webserver must be configured and Unified Resource Locator (URL) must be designed, but for network socket and XML-RPC protocols "server and client"are easy in implementation at both communication parties side.

B. Logs Tracking Engine

Cloud system resources generate usage statistics (logs) in a timely manner, based on the activity (workload) run in the system. Log value changes are important to indicate the system workload changes. Sampling reading logs is capturing a snapshot from system resource usage. A good monitoring tool always try to give an accurate indication about the system with less overhead in sending and processing data. This means not to send logs in a continuous way but only send the informative logs that show the system changes. A logs change tracking system must figure out when logs changes have important information to send or just skip without sending. This engine, must also emphasize the sampling rate in such a way that high frequency log statistics must increase the sampling rate to capture the log signals based on Shannon theory (Sampling Nyquist Rate "sampling rate must be at least two time of frequency change"). A challenging problem

TABLE I: Messages and Protocols Exchange Characteristics

	Protocol Type	Size (Bytes)			Transmission Delay (ms)			Processing Delay (ms)		
Message Format	**Protocol**	S	M	L	S	M	L	S	M	L
XML Message	XML-RPC	1078	1971	5783	122	122	125	53	56	57
	REST	1071	1964	5777	85	85	86			
	socket	1078	1971	5783	2	4	6			
JSON Message	XML-RPC	706	1268	3735	120	125	125	4	6	7
	REST	699	1261	3728	85	85	86			
	socket	706	1268	3735	2	4	4			
protobuf Message	XML-RPC	400	942	1944	123	123	125	12	17	17
	REST	393	934	1930	83	83	86			
	socket	165	313	632	43	47	48			

is to decide the sampling frequency and logs reporting to achieve the best indication about system in a good time that allows cloud system to act at the right time. Based on the aforementioned ideas the following issues must be considered: 1) Logs maintainer engine, which works locally in each cloud infrastructure node. This engine must be smart enough to do some processing on fresh collected logs, then tune the sampling rate and the transmission time. 2) Logs window size, which is a history record of recent log values. This array must be maintained always to compare between generated logs. Log transmission trigger is based on a predefined threshold value that can send informative logs. This method makes the update important and urgent for cloud system and elasticity scaling. For change tracking, we used a simple method that has a small log window size (two values), which can skip contiguous same value logs. We will consider log changes tracking in our future work.

C. Sampling frequency

Sampling rate must follow track changes engine analysis to achieve a dynamic sampler. But for the current system, we investigate the best sampling rate that must be considered by comparing between different sampling frequencies for a CPU usage in a VM. The natural thought is higher sample rate will give an accurate measurement for the system. But as Figure 4 depicts this is not true, because the higher sampling rate (0.1s) will start to show Guest VM OS CPU scheduler context switching time. This gives an unstable log report where CPU usage fluctuates between 0% usage and 100% usage for 0.1s sampling time. This will increase logs transmission, which causes instability of provision resources and extra data processing and transmission overhead. On other hand, lower sampling time (10s) will make the sampled report insensitive to system changes and will give inaccurate indication about system status. This delayed report has a crucial effect on cloud management system decisions. Cloud management system needs to measure the resources accurately, but with eliminating extra logs reading and report transmission overhead, keeping track of the critical changes (high change in resource usage and reporting this incident immediately) conducted by dynamic sampling rate monitoring system. From Figure 4 the best sampling rate is between 0.5s for high fluctuate workload to 3s for stable workload. This rate can meter cloud system resource usage accurately without losing any informative logs. Moreover, cloud management system can get system state on

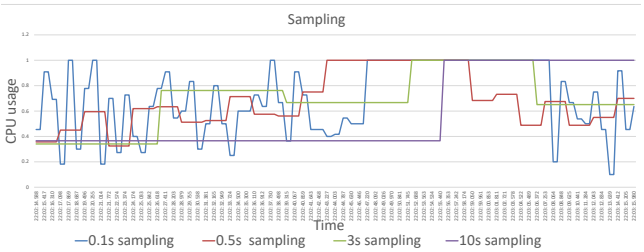

Fig. 4: Sampling Time Variation

right time to scale resources appropriately proportional to the demand in elastic cloud environment.

D. Logging Data Source

Xen implementation [22] is similar to Unix design for resource management and message intercommunication. The building blocks for system components interfaces in Xen match the Unix OS interfaces for asynchronous and synchronous message exchange between system components, as shown in Table II. In Xen, events do the same task as signals in Unix, where signals carry an asynchronous message coming from outside the current running programs, indicating an event E has occurred. This is done in Xen by registering the guest OS kernel to the hypervisor callback event to send and receive messages, which is managed by domain0 that specifies multiple channels to deliver multiple events.

System call is used in Unix to do any operation in system (privilege) mode, which causes changes in the context to ask the kernel to do the task on behalf of the user application. Xen has same methods called hypercall where the guest domain asks the privilege domain to handle the critical tasks. In general, we have two types of Xen API, the first one (the hypercall) that is used by guest domain and domain0 to communicate, and the second one (Xen Management API) that is used to manage the Xen hypervisor to allow certain tools to communicate with the hypervisor in order to configure and manage it. Both Xen API types are developed using XML-RPC protocol to exchange the messages. All data messages are formatted and encoded in a XML message datatype.

System resource status usage is collected based on the message trigger that the Xen interface API such as XAPI can call a system hypercall to read the CPU usage for a specific Guest Domain through a communication channel. The values read from the data source by using system call via Linux kernel

TABLE II: Xen Component Compared to Unix Components

Unix	Xen
System Call	Hypercall
Signals	Events
FileSystem	XenStore
POSIX Shared Memory	Grant Table

or hypervisor system call to read the CPU usage is based on scheduler state. Sources to read VMs CPUs state in a cloud system vary based on the type of the log data source for the VM. It can be from the hypervisor local call or by reading the VM operating system CPU utilization.

In XenServer, we can consider the XAPI as the main source to read the VM CPU utilization. In this case there are three data sources: 1) the 'xe' hypercall; 2) Round Robin Database that is updated every few seconds considering a consolidation of high frequencies by averaging the values. "'rrd"' is updated by a standalone tool that collects all guest domains and domain0 resource utilization based on hypervisor CPU scheduler for local host [23]. 3) "xentop"command, which is like the Linux "top"command. The "xentop"command collects resource statistics of the host server domains including domain0 [24]. The other way to read the VM CPU utilization is a direct read for Linux CPU scheduling by using "top"command based on Linux "ps"command. It tracks system, user, waiting, hypervisor and idle processes states.

In general, CPU scheduler is the program execution master that decides which task has the right to access the CPU resources and when to execute to maximize the CPU utilization [25]. CPU usage is measured by calculating how long the CPU is not in the idle state. Linux kernel 2.6 and later versions have various types of scheduling mechanisms [24] summarized as, Sched_FCFO (First Come First Serve), Sched_RR (Round Robin), Sched_Normal which has a fixed time slot for each task, and Sched Batch for CPU intensive tasks. Xen hypervisor uses two types of scheduler, Simple Earliest Deadline First (SEDF) which is based on priority queue in analyzing requests from domains to assign a weight for CPU sharing, and Credit scheduling that can handle domains (VMs) workload changes dynamicity by adjusting CPU and domain priority to get a fair share scheduling.

Figures 5, 6, 7, 8 and 9 show the difference between data source CPU usage statistics for VM under four different workloads (VM idle, VM running normal workload, VM running a very intensive CPU workload "infinite while loop"and CPU running a regular application workload Linux "yum command"). Note for both idle and normal workloads all the four data sources (xe, rrd, top, xentop) have a minor time delay between them indicating the CPU resource usage change. But in intensive workload we can see about seven seconds delay showing the system status changes between "xe/rrd") and "top/xentop"readings. This happens because log collection is periodic in "xe"and "rrd"data sources. For Figures 8 and 9 the CPU usage shows a weird behavior where in

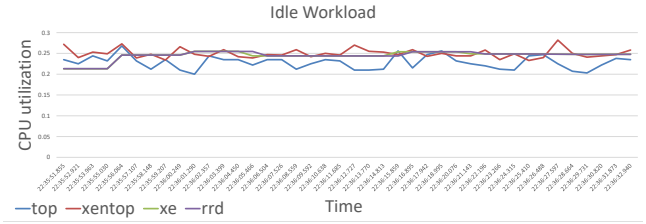

Fig. 5: Idle Workload

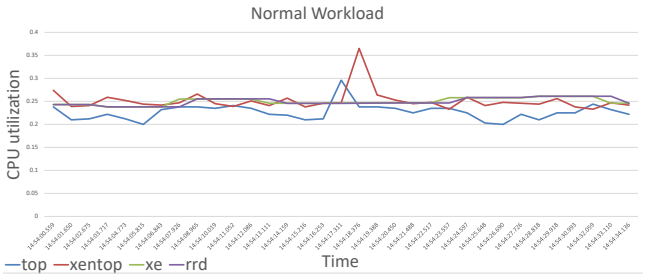

Fig. 6: Normal Workload

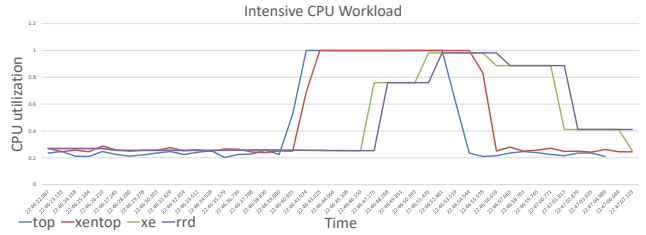

Fig. 7: CPU Intensive Workload

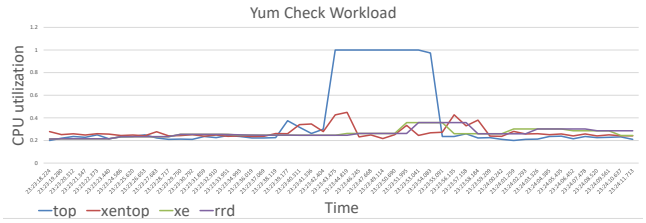

Fig. 8: Yum Command Workload

"top"the CPU usage indicates 100% but in other data sources it does not exceed 45%. This is because of the Linux and Xen scheduling techniques as explained before. Where in regular "YUM workload"there is a context switching between VM processes that shows in Xen CPU scheduler idle state, but in the VMs scheduler the CPU is busy by switching between workload tasks.

V. Cloud System Setup

A small-scale cloud setup environment has been emulated using XenServer hypervisor and XenCenter for basic management. Java applications and Linux Bash scripts are

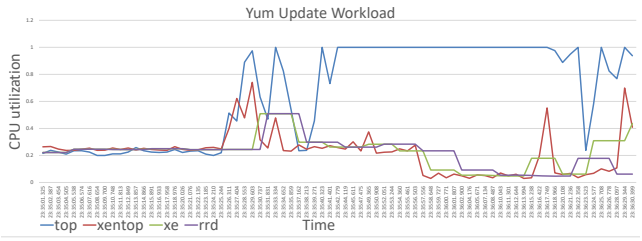

Fig. 9: Yum Command Workload (2s Sampling)

developed to collect logs from the cloud environment. The physical infrastructure of our work includes two identical Dell workstations with high speed processor 4 cores of Intel Xeon 3.6 GHz speed, connected by Linksys Ethernet switch with port speed 100Mbps. Each Dell server runs custom version of Linux OS (CENTOS5.6) which supports Xen kernel 2.6.32.43-0.4.1.xs1.8.0.835.170778xen. The network connection is limited by the switch speed capabilities, which is only 100Mbps. Cloud pool includes both the host physical servers, a master node is configured for virtualization pool control management, which represents an interface of outside management via XAPI API ("xe"command). This server has all the virtualization configuration files and monitoring reports from physical host server and the VMs guests machines. Four VMs running Linux Guest OS are deployed in the small setup, which are running different kinds of applications.

VI. Conclusion

The data source of the logs, the transportation protocol used in communication, and the message format are not considered in most of the monitoring tools available in the literature. By using latest technology in message encoding (Google Protocol Buffer) with network socket communication protocol, we achieved an excellent reduction in message size. This is very important in cloud management system to reduce network overhead.

Selecting "xentop"as the best data source (considering accuracy and performance perspectives) with good sampling rate provides us an accurate reading for cloud system. Sampling rate between 0.5s to 3s gives us an accurate and lower data redundancy. Our future work will focus on logs correlation and aggregation, to trigger log readings from data sources with minimum overhead to update cloud management system (logs change tracker engine). Cloud elastic models relies on the changes in cloud environments, which demands a system status anticipation that can be figured out from logs values.

VII. Acknowledgment

The authors would like to acknowledge the financial support provided by Natural Sciences and Engineering Research Council (NSERC)'s CRD Grant in collaboration with Cistech Limited, Canada.

References

[1] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, "Dynamic resource management using virtual machine migrations," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 34–40, September 2012.

[2] M. Mohamed, M. Amziani, D. Belad, S. Tata, and T. Melliti, "An autonomic approach to manage elasticity of business processes in the cloud," *Future Generation Computer Systems*, vol. 50, pp. 49 – 61, 2015, quality of Service in Grid and Cloud 2015.

[3] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, Sept 2009.

[4] openstack. Monitoring & metering. [Online]. Available: https://www.openstack.org/software/project-navigator

[5] opennebula. Large deployments. [Online]. Available: http://docs.opennebula.org/5.4/deployment/references/one_scalability.html#monitoring

[6] eucalyptus. Build your own private cloud with eucalyptus. [Online]. Available: http://opensourceforu.com/2014/03/build-private-cloud-eucalyptus/

[7] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of cloud monitoring tools: Taxonomy, capabilities and objectives," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2918 – 2933, 2014.

[8] A. Datt, A. Goel, and S. Gupta, "Analysis of infrastructure monitoring requirements for openstack nova," *Procedia Computer Science*, vol. 54, pp. 127 – 136, 2015.

[9] S. A. D. Chaves, R. B. Uriarte, and C. B. Westphall, "Toward an architecture for monitoring private clouds," *IEEE Communications Magazine*, vol. 49, no. 12, pp. 130–137, December 2011.

[10] H. Huang and L. Wang, "P&p: A combined push-pull model for resource monitoring in cloud computing environment," in *2010 IEEE 3rd International Conference on Cloud Computing*, July 2010, pp. 260–267.

[11] W.-C. Chung and R.-S. Chang, "A new mechanism for resource monitoring in grid computing," *Future Generation Computer Systems*, vol. 25, no. 1, pp. 1 – 7, 2009.

[12] R. Sundaresan, T. Kurc, M. Lauria, S. Parthasarathy, and J. Saltz, "A slacker coherence protocol for pull-based monitoring of on-line data sources," in *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.*, May 2003, pp. 250–257.

[13] R. Sundaresan, M. Lauria, T. Kurc, S. Parthasarathy, and J. Saltz, "Adaptive polling of grid resource monitors using a slacker coherence model," in *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, 2003, pp. 260–269.

[14] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive push-pull: disseminating dynamic web data," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 652–668, Jun 2002.

[15] M. Lin, Z. Yao, and T. Huang, "A hybrid push protocol for resource monitoring in cloud computing platforms," *Optik - International Journal for Light and Electron Optics*, vol. 127, no. 4, pp. 2007 – 2011, 2016.

[16] J. Shao, H. Wei, Q. Wang, and H. Mei, "A runtime model based monitoring approach for cloud," in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010.

[17] Y. Su, J. Yin, Y. Chen, and Z. Feng, "A high performance transport protocol for jtangmq," in *2009 Second International Symposium on Information Science and Engineering*, 2009, pp. 268–272.

[18] google. Developer guide. [Online]. Available: https://developers.google.com/protocol-buffers/docs/overview

[19] Citrix. General architecture. [Online]. Available: https://xenserver.org/overview-xenserver-open-source-virtualization/download.html

[20] XAPI. Xapi architecture. [Online]. Available: http://xapi-project.github.io/xapi/architecture.html

[21] ——. General architecture. [Online]. Available: http://xapi-project.github.io/getting-started/architecture.html

[22] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[23] Xenproject. Xapi rrd. [Online]. Available: https://wiki.xenproject.org/wiki/XAPI_RRDs

[24] W. von Hagen, *Professional Xen Virtualization*. Birmingham, UK, UK: Wrox Press Ltd., 2008.

[25] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 8th ed. Wiley Publishing, 2008.