Anatomy of Cloud Monitoring and Metering: A case study and open problems

Ali Anwar*, Anca Sailer†, Andrzej Kochut†, Ali R. Butt*

*Virginia Tech, †IBM Research – TJ Watson

{ali,butta}@cs.vt.edu, {ancas,akochut}@us.ibm.com

Abstract

Microservices based architecture has recently gained traction among the cloud service providers in quest for a more scalable and reliable modular architecture. In parallel with this architectural choice, cloud providers are also facing the market demand for fine grained usage based prices. Both the management of the microservices complex dependencies, as well as the fine grained metering require the providers to track and log detailed monitoring data from their deployed cloud setups. Hence, on one hand, the providers need to record all such performance changes and events, while on the other hand, they are concerned with the additional cost associated with the resources required to store and process this ever increasing amount of collected data.

In this paper, we analyze the design of the monitoring subsystem provided by open source cloud solutions, such as OpenStack. Specifically, we analyze how the monitoring data is collected by OpenStack and assess the characteristics of the data it collects, aiming to pinpoint the limitations of the current approach and suggest alternate solutions. Our preliminary evaluation of the proposed solutions reveals that it is possible to reduce the monitored data size by up to 80% and missed anomaly detection rate from 3% to as low as 0.05% to 0.1%.

1. Introduction

The cloud computing model has emerged as the de facto paradigm for efficiently providing infrastructure, platform, and application services for IT industry. As a result, vendors such as IBM, Amazon, and RedHat offer cloud based solutions to optimize the use of their data centers. Customers of these cloud providers, particularly those building their critical production businesses on cloud services, are interested in detailed monitoring data to track in real time the health of their thousands of service instances. Netflix, for instance, collects tens of thousands of metrics per microservice every 1 to 5 seconds [1]. With service management datasets growing at a rate of 10 Billion+ records/day, typical monitoring and analysis tools break or require significant additional capacity for storage and computation.

Furthermore, the charge model most sought after by the customers of those providers is the fine-grained pay-per-use, where users are charged for the amount of specific resources, e.g., volume of transactions, CPU usage, etc., consumed during a given time period [18]. The cloud service providers, looking to maintain the competitive advantage by effectively adapting to versatile charging policies, have started to promote pay-per-use. However, usage based pricing brings a new set of service management requirements for the service providers, particularly for their revenue management [27]. The finer-grain metering requires monitoring of service resources and applications at appropriate level to provide useful information about the resource consumption that is to be charged for. This may result in collecting significantly large amounts of metered data. Additionally, this metered data needs computational resources to be processed in order to perform revenue management specific tasks.

The resource capacity requirement for such non-revenue generating systems such as monitoring and metering fluctuates largely with the service demand (e.g., the number of service instances), the service price policy updates (e.g., from single metric based charge to complex multi-metric based charge), the resolution of the system behavior exposed (e.g., from higher-level aggregations to individual runaway thread), while their unit cost changes depending on the operational infrastructure solution (e.g., on premise, traditional outsourcing or IaaS). A crucial challenge for the cloud service providers and their customers is how to control the quickly escalating data size of their service management, and implicitly its costs, in order to profitably remain in the race for the cloud market.

Hence there is a clear need to understand and improve the design of underlying architecture provided by open source

ApSys'15, July 27-28, 2015, Tokyo, Japan.

http://dx.doi.org/10.1145/2797022.2797039

cloud solutions, such as OpenStack, to efficiently carry out monitoring and metering tasks. In this paper, we present a deep analysis of how OpenStack facilitates monitoring and metering. We specifically focus on how the monitoring data is collected by OpenStack and by analyzing the collected data we identify open problems in OpenStack. Furthermore, we also present solutions for those open problems and discuss the challenges therein.

2. OpenStack – Ceilometer Background

In OpenStack, Ceilometer [3] is the module that provides an infrastructure to collect detailed measurements about resources managed by OpenStack. The main components of ceilometer can be divided into two categories, namely agents, e.g., compute agents, central agents, etc., and services, e.g., collector service, API service, etc. The compute agents poll the local `libvirt` daemon to fetch resource utilization of launched VMs and emit this data as AMQP [2] notifications on the message bus called Ceilometer bus. Similarly, central agents poll the public RESTful APIs of OpenStack services, such as Cinder and Glance, to track resources and emit this data onto the OpenStack's common message bus called Notification bus. On the other hand, the collector service collects the AMQP notifications from the agents and other OpenStack services, and dispatches the collected information to the metering database. Finally, the job of the API service is to present aggregated metering data to the billing engine. In Ceilometer, resource usage measurement, e.g., CPU utilization, Disk Read Bytes, etc., is done by meters or counters. Typically there is a meter for each resource being tracked, and there is a separate meter for each instance of the resource. It is important to note that the lifetime of a meter is decoupled from the associated resource, and a meter continues to exist even after the resource it was tracking has been terminated [3]. Each data item collected by a meter is referred to as a "sample," and consists of a timestamp to mark the time of collected data, and a volume that records the value. The polling interval between two events is specified in the pipeline.yaml file and can be adjusted according to the cloud provider requirements. Once configured, the same polling interval is used to monitor all the instances launched in that particular setup, unless the cloud provider manually changes it.

3. Open Problems in Cloud Monitoring

In this section we present our analysis of how monitoring and metering has been implemented in OpenStack and pinpoint existing problems.

3.1 Constant polling frequency

The frequency at which samples are collected for a certain meter is called the polling frequency for that meter. OpenStack allows cloud service providers to manually configure the polling frequency for different types of meters, however, once configured it remains constant unless updated manually. Hence, cloud service providers end up collecting large amounts of close to identical samples often carrying information of low significance about the usage or state of the tracked resource. This results in high storage volume and increased computational resource requirements to first collect and then process the collected data in view of metering, incident, or problem management purposes, to name a few. To understand the problem and potential solutions, lets assume a concrete case where the utilization of a certain resource remains constant at a specific value for 5 hours. Assuming the polling frequency is per second, we collect in total 18000 samples. From the metering point of view this same information would have been inferred from data collected according to a polling frequency of one or a few samples [1] per hour. If we scale this calculation to hundreds of metrics on thousands VMs in a typical cloud setup, the problem increases by many folds.

Category	# of VMs	Duration
AP	737	3 months
Africa	400	8 days
Australia	999	3 months
EMEA	1223	3 months

Table 1: Range of IBM production servers used in our study.

The logical question to ask is how prevalent is in a typical cloud environment this scenario where the resource utilization remains unchanged? To answer this question we collected and analyzed the data from 3359 machines launched in geographically distributed IBM production servers. Table 1 shows the range of IBM production servers used in our study, in Asia Pacific, Africa, Australia, and Europe. The data was collected over a period of 3 months with a sample collected after every 15 minutes.

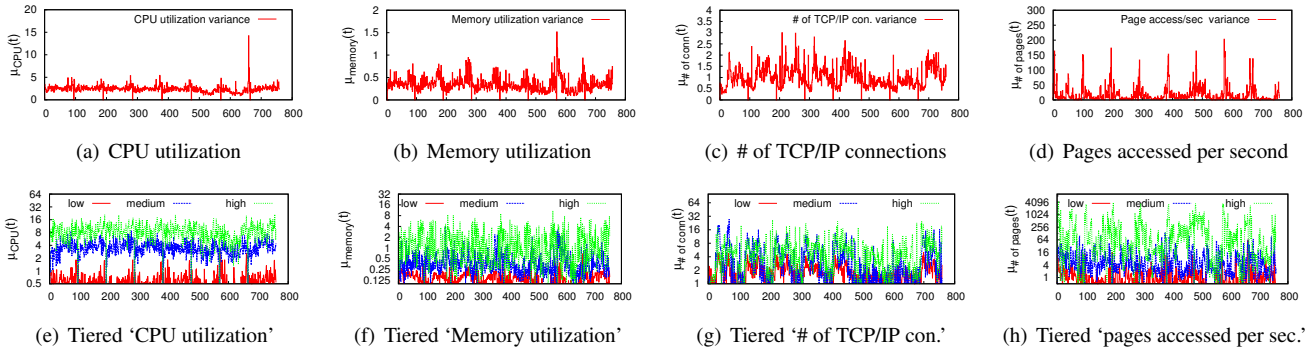
$$\left|\frac{dRvm_i(t)}{dt}\right| = |Rvm_i(t) - Rvm_i(t-1)| \quad (1)$$

$$\mu_R(t) = \frac{\sum_{i=1}^{N}\left|\frac{dRvm_i(t)}{dt}\right|}{N} \quad (2)$$

To study the variance of resource utilization, we calculated the mean of absolute rate of change, $\mu(t)$, for different monitored resources of randomly picked 338 VMs from all the regions. We analyzed two different kinds of meters: i) Meters used to directly monitor the infrastructure usage like CPU and memory utilization, and ii) Meters used to track the load imposed by VMs on the physical infrastructure like number of TCP/IP connections established by VM and pages accessed per second from the disk. We first calculated the absolute value of the rate of change at time *t* in each resource usage (e.g., CPU utilization, memory utilization, number of TCP/IP connections, pages accessed from disk/sec. etc.), for each VM *i*, as shown in equation 1. Then we average across all VMs as shown in equation 2.

Figure 1(a) and 1(b) show $\mu(t)$ of the CPU and memory respectively for the last 8 days of our collection of data. We

[1] Cloud providers may be interested in collecting more samples to properly track the health of the monitored resource. We will address this case later.



(a) CPU utilization (b) Memory utilization (c) # of TCP/IP connections (d) Pages accessed per second

(e) Tiered 'CPU utilization' (f) Tiered 'Memory utilization' (g) Tiered '# of TCP/IP con.' (h) Tiered 'pages accessed per sec.'

Figure 1: Mean of absolute rate of change in utilization of resource R, i.e. $\mu_R(t)$, for studied VMs; and tiering of VMs in three different clusters based on rate of change in utilization for resource R, i.e. $\left|\frac{dRvm_i(t)}{dt}\right|$.

found over this period of 8 days the instantaneous variation of the resource usage to be less than 5%. We repeated this evaluation on the two other type of meters, '# of TCP/IP connection' and 'pages accessed/sec'. We found the same trend for meter monitoring as shown in figure 1(c) and 1(d). This behavior hold true in average across VMs and the 3 month of data in our collection. Overall we found less than 5% of VMs having sudden variation in the tracked resource utilization.

An additional observation is that for the 5% of VMs having sudden variation in resource utilization, the polling was not frequent enough to properly capture the evolution of the change. Hence, if on one hand decreasing the polling frequency can be beneficial for the majority of the VMs, on the other hand it is advantageous to identify those VM that benefit from an increased polling frequency compared to the default in order to better capture their behavior in view of modeling it.

3.2 Global polling frequency for all VMS

For each meter, Ceilometer uses the same polling frequency globally across all the VMs launched in an OpenStack setup. Tiering based on variation in resource utilization can enable a cloud service provider to monitor and collect samples from each tier at a different polling frequency, hence allowing to track at a lower polling frequency the resources having less resource usage variation. Let us consider the scenario where a metric exhibits 3 types of behaviors across a set of VMs as follows: high variance in utilization of monitored resource; predictable or medium variance; and low variance.

Currently, the cloud providers are limited to monitor this resource by sampling it at one and the same frequency in all VMs, which would be the polling frequency to capture a predefined volume of changes in the utilization of that resource, e.g., 98%,. This limitation of choice in the sampling frequencies leads to resources in the low variance tier to be over sampled by being monitored at a too high frequency, whereas the resources in the high variance tier are under-sampled.

To further understand the characteristics of our data, we manually divided the VMs into three different tiers by calculating mean rate of change in resource utilization for each of the 338 VMs. We defined thresholds by dividing the range between maximum and minimum found values in three equal tiers for each of the four resources. Following the levels of variance in $\mu_R(t)$ of the four monitored resource, we found that for each meter used to track these resources, more than 80% of the VMs fall in the tier with low variance; 15% were found in the tier with medium variance and 5% in the ties with high variance. Figure 1(e) and figure 1(f) show the results for tiered CPU and tiered memory utilization, respectively. Similarly, figure 1(g) and figure 1(h) show the results for tiered '# of TCP/IP connections' and 'pages accessed/sec', respectively. This analysis shows that in case of un-tiered sampling, 80% of resources were monitored at a frequency higher than needed to capture their changes, whereas 5% of the resources were monitored at a frequency smaller than needed to capture their changes.

3.3 Lack of policy based monitoring

To enable custom, tier based polling frequencies, OpenStack should support a policy based data monitoring. A monitoring policy defines the sampling rules for a given metric profile. Currently, Ceilometer lacks this feature, thus prohibiting the cloud providers from defining metric profiles and associated differentiating policies.

The applications hosted by a system dictate their monitoring and data retention requirements. For instance, the monitoring requirements for desktop clouds are different from those for HPC application or for MapReduce jobs. Similarly, monitoring and data retention requirements for charging purposes are different from those for system health check. Metrics with higher (/lower) monitoring data resolution requirements should be able to configure in their profile that their data is critical (/not critical) and hence an conservative (/reduced) sampling and storing policy can be associated to them. Another advantage of policy based sampling is that it enables cloud providers to separate those resources for which prompt anomaly detection is required.

Furthermore, what monitoring data needs to be collected and how monitoring data is collected highly depends on usage of these monitoring data. Our goal is to enable the cus-

Metric Profile				Metric Policy
Critical	Dependency root	Used for usage based charging	Independent metric	
✓	∗	∗	∗	Conservative sampling and storage
✗	✓	∗	∗	Conservative sampling and storage
✗	✗	✓	∗	Conservative sampling and aggregated storage [a]
✗	✗	✗	✓	Per tier sampling and conservative storage
✗	✗	✗	✗	Per tier sampling and aggregated storage

Table 2: Example of mapping between metric profile and metric policy (∗ means either of ✓or ✗).

[a] Replaced by conservative storage in case the provider is required to keep evidence of the raw metered data for financial regulations.

tomization of metric profiles by allowing the providers to characterize each metric in terms of what it is used for (e.g., used for charging the tenant as part of the usage based price definition, used for health check etc.), or in terms of its importance in the inventory (e.g., belongs to VM hosting critical or non-critical applications), in terms of its precedence in the application flows (e.g., belongs to a leaf or root item in the dependency graph), or in terms of its dependence on other metrics (e.g., independent or correlated metric). These criterias, while extendable, represent key configuration items to be defined in the metric profile. Based on the metric profile, each metric is associated a metric policy. A polling policy can be as naive as collecting and storing only the data from the last hour, day or even week initially at full granularity and then aggregate over time; or it can be as complex as collecting data, storing it, analyzing it and then fine tune how to capture and store it in a more efficient way.

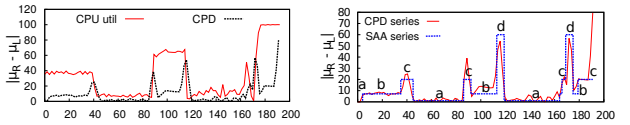
4. System Design

In this section we present a methodology which enables the cloud providers to customize their service management monitoring system for a policy based data monitoring such that each tier of systems with similar monitoring data behavior and business needs gets its VMs monitored according to the same dedicated polling policy, different from the other tiers.

4.1 Metric profiling

Our solution allows cloud providers to specify configuration items such as: (i) REST URLs of the usage calculation classes for the usage based pricing -these classes typically calculate the transformation of the raw metered data into the charged unit of measure, or the maximum or sum of the metered values, or more complex metric aggregations; (ii) inventory and interdependencies between servers, applications, network devices, software, configuration files, operating systems and other IT infrastructure components expressed as graphs, xml files or spreadsheets [10]; (iii) event correlation engine API [11].

The proposed solution programmatically and periodically accesses the specified sources of data to automatically populate or update the profile of each metric. A mapping between each item in the profile and its corresponding sampling rule is to be maintained manually by the cloud provider. For instance, a metric that is critical or belongs to root item in



(a) CPD (b) SAA

Figure 2: Change Point Detection and Symbolic Aggregation Approximation on CPU utilization of a VM over period of two days.

the dependency graph requires a more conservative sampling and storage policy compared to a metric that is non-critical, belongs to a leaf item in the dependency graph or is not used for usage based charging.

Table 2 illustrates a potential mapping between metric profiles and metric policies. A conservative sampling implies using the default high frequency sampling of the monitoring system; a conservative storage means recording all the collected data samples; a per tier sampling denotes a reduced sampling frequency as inferred by our solution for the behavior of the metrics in that tier; an aggregate storage means applying a corresponding aggregation rule e.g., keeping the independent metric only in case of correlated metrics, or aggregating as indicated by the usage calculation class for a metered metric, etc., and then recording only the value of the aggregate.

4.2 Tiered polling frequency

Once configured and enabled, the service management monitoring system, e.g., Ceilometer, polls the systems metrics initially at a regular, default frequency. We first identify the metrics suitable for aggregated storage and apply the aggregation technique on the fly. For instance, for metrics characterized as correlated to another metric, we compare the new sample to the last stored sample and store the new one in the database only if its value is different than the last one. The metrics used for usage based charging will be applied the usage calculation logic (e.g., sum, max, min) and the result will update the current value without creating a new entry in the database.

Then we identify the metrics suitable for tiered sampling. Each sample of data collected from the cloud environment is a data point in the time series of that metric on a given configuration item. After sufficient data is collected, e.g., for a few weeks, we apply a Change Point Detection (CPD) analysis algorithm [12] on their time series to identify the number and timing of the changes that occur in each metric. Figure 2(a) shows an example of CPD applied on a CPU uti-

lization of a VM over period of two days. We subsequently apply Symbolic Aggregate Approximation [7, 21] on the change point time series to convert the data into a discrete format of a sequence of symbols with a small alphabet size as shown in figure 2(b). The strings obtained, e.g. 'abcacbdacdbc' in given example, encode the change behavior of the metrics selected for a reduced sampling. Thus, one symbol corresponds to the flat, monotone segments of invariable behavior, while a few other symbols indicate when the metric changed. Therefore, we are interested to collect precise data around the timing of the change occurrences which correspond to changes in the metrics statistical properties, while collecting samples during the monotone stretches of consecutive identical symbols is of less interest. When these segments of unchanged performance keep for many hours, it is unnecessary to poll and store data at 1 to 5 seconds interval. In consequence, a few samples only are to be collected during those segments of unchanged performance.

Furthermore, we notice that large groups of metrics have similar change point time series, with spikes corresponding to seasonal periods of the day and of the week/week-end. To reduce the number of metric policies to maintain, we split the sequences in segments corresponding to weekly periods (Monday to Sunday) and group them. We employ the Structural Similarity algorithm [7, 21, 26] to cluster the symbol segments generated above into tiers of metrics with similar sequences. A metric having all its weekly segments in one group indicates that its weekly pattern is stable across the analyzed weeks. We filter out of the tiers all the metrics with segments scattered in different clusters. We infer then the polling frequency for the remaining metrics in a tier from their weekly sequence of symbols as follows:

- For each isolated occurrence of a spike symbol in any metric segment, data is collected from the timing of the beginning of the spike until the end of the spike (e.g., every second during a minute).
- For the segments with unchanged performance, data is collected hourly only if no isolated spike has already triggered data collection during that hour.

The signature of the polling timing identified in each tier represents the sampling metric policy of that tier. As these policies are made available by our solution, the service management monitoring system can switch to from the default policy and start making an efficient usage of storage and computational resources.

Online Analysis: as a metric behavior or monitoring policy evolves over time, its classification in a particular tier may become unsuitable for the new sampling and storage requirements. A metric policy update will directly trigger the reclassification of the metric into the default, full sampling tier, where the process of classification described above will be re-applied. However, a metric behavior update is not directly signaled unless it is monitored. To this end, we compare the samples collected for tier based sampling to the most recent average of the past collected values corresponding to the metrics sequence symbol generated by the Symbolic Aggregate Approximation for that particular time in the sequence. If the difference is bigger than a threshold (e.g., 10%), we place the metric into the default, conservative sampling tier, for reclassification. An additional goal is to identify and use in the meter profile those configuration items that have a reduced sensitivity to the changes in the monitored environment, and hence a limited potential of causing the metrics to oscillate between the tiers.

5. Preliminary Results

We used Python and R [8] for this preliminary evaluation of our solution. The data analyzed was collected from IBM production servers over a period of 3 months (Table 1 and §3.1).

Policy	Data Reduction		Miss Detecion Rate
	Non Aggressive	Aggressive	
`Cons_Sam + Cons_Stor`	0 %	0 %	3 %
`Cons_Sam + Aggr_Stor`	61.17 %	70.17 %	3 %
`Tier_Sam + Cons_Stor`	72.7 %	76.8 %	0.05 - 0.1 %
`Tier_Sam + Aggr_Stor`	76.32 %	80.04 %	0.05 - 0.1 %

Table 3: Data reduction and miss detection rate under different policies.

For evaluation purposes we compared default or conservative sampling and storage with tiered sampling and aggregated storage. The policies introduced in §4.1 and Table 2, were defined based on two procedures, one aggressive and the other non aggressive. In the aggressive approach, i) we set a higher threshold for aggregated storage, and ii) we aggressively reduce the polling frequency when collecting monitoring data from tiers with low variance in the resource utilization. For each policy, we measured the reduction in the collected data size, as well as the missed anomaly detection rate. The missed anomaly detection rate was calculated by comparing the data collected for each policy with the anomalies found by examining the system logs collected for the same time period using sysstat utilities [9]. The sysstat data was collected at a frequency double than the maximum polling frequency used to collect the monitoring samples. We defined anomaly as a missed sample having a sudden increase or decrease in utilization as compared to its adjacent samples. Furthermore, for the policies involving metered data we enforced that we store enough samples so that metering tasks can be successfully performed in following revenue calculation stages.

Table 3 illustrates the results of our evaluation. Notice that by storing aggregated data instead of conservatively storing all samples, we obtained a 60% to 70% reduction in data collected for monitoring purpose. The decrease in data size was due to storing only samples which either conveyed useful information about the current health of the monitored resource or were required for charging purposes. The missed anomaly detection rate for conservative sampling was found 3% as the default polling frequency was not high enough to track the changes in VMs having sudden variations in resource utilization.

Next, when tiered sampling was enabled with conservative storing of every sample, we were able to reduce the data size by 72% to 76% whereas missed anomaly detection rate was found to be only 0.05% to 0.1% due to higher polling frequency used for the set of VMs having sudden variations in resource utilization. Further evaluation revealed that 99.99% of the anomalies were from the tier for which reduced the polling frequency was used. Hence missed anomaly detection rate can be further reduced by using less aggressive approach.

Finally, when applying both tiered sampling and aggregated storage, we obtained up to 80% reduction in data size. Missed anomaly detection rate remained the same as in the case of tiered sampling with conservative storing, since shifting from conservative to aggregated storage without changing the sampling policy does not affect the anomaly detection.

6. Potential Impact and Current Work

We estimated the storage savings by considering an average object size of a sample of 1024 bytes [4]. This size is due to the information related the resource usage plus the additional fields, e.g., instance_id, timestamp, resource_id, user_id, project_id, etc. If a single VM produces 100 counters per second and storage costs $0.07 per GB then a rough estimate of savings for an environment of 1000 VM, per year can be calculated as following:

0.8 x $0.07 per GB / month x 100 samples / sec x 1024 bytes x 60 sec / min x 60 min / hour x 24 hours / day x 30 days / month / 10 ^ 9 bytes / GB = $14.864 / VM / month x 1000 VMs = $14,864 / environment / month, which accumulated over one year results in a saving of $1,159,392 / environment / year [2].

It is important to note that in some cases (e.g. metering data for charging) cloud providers are bound by SLA to keep customer data for as long as 3 to 5 years. Therefore, our solution is beneficial to both tenant (for the savings) and provider (for the competitive advantage).

On-going efforts We are looking for new criterias of evaluation besides the missed anomaly detection that could shed additional light on the impact of our policy based monitoring on the service management functionality. Also, the current solution attributes fixed weights to the profile items in the different policies, while a change in the environment, e.g., an incident, may increase the relevance of leaf systems detailed monitoring data and hence require an augmented weight to be reflected in an increased polling frequency. Furthermore, we are evaluating our solution in real time scenarios. Our plan is to have our solution integrated with IBM production servers to directly evaluate it by comparing it with existing more conservative setups used to collect monitoring data. This will help in gaining more confidence on our approach and carve the aspects that lead to positive results. We are also investigating the effects of using different window sizes for CPD analysis, and performing more fine grained symbolic approximation of data generated by CPD, on the resource requirements for cloud management. This evaluation will provide to the cloud providers further control on the monitoring setup as it will enable them to perform informed tradeoffs between the degree of monitoring and the cost associated with the monitoring setup.

[2] The amount of stored data increases each month and can be represented by an arithmetic progression. Hence, $n\frac{(a_1+a_n)}{2}$ was used to calculate the cost accumulated over period of 12 months.

7. Related Work

The focus of several recent works [13–15, 19, 25] is on providing an efficient and scalable cloud monitoring setup, however, these works do not consider or discuss reduction in collected monitoring data. Similarly, some other works in distributed state monitoring such as [20, 24] are either to study the problem of employing distributed constraints to minimize the communication cost or to ensure trustworthiness of resource accounting [17, 22]. While these works either provide communication efficient detection or verifiable resource accounting, we study the lower level problem on efficiently collecting monitoring data for both anomaly detection as well as cloud metering purposes. Recently, there has also been work done to evaluate the idea of using Time-Series storage for the metering data but unlike us the goal of this work is limited to reducing the avgObjSize of each sample [5, 6]. Volley [23] proposes violation likelihood based state monitoring for datacenters and perhaps is closest to our work as it also utilizes node-level adaption algorithms to minimize monitoring cost. Our work differs from Volley in a sense that we not only study the effect of controlled monitoring on anomaly detection but also study its impact on cloud metering. Finally, some works make assumption on value distribution[16] in clustering VMs for multi-cloud systems, while our approach makes no such assumption.

8. Conclusion

In this paper, we presented an analysis of how monitoring data is collected by OpenStack. By analyzing the characteristics of the collected monitoring data, we identified several open problems and presented alternative solutions. Our preliminary evaluation using actual data from IBM production servers reveals that it is possible to reduce the monitoring data size upto 80% and missed anomaly detection rate from 3% to as low as 0.05% to 0.1%.. We believe that we have looked deeply into how monitoring subsystem provided by open source cloud solutions collect data and this paper will lead to lots of discussion on how these subsystems can be improved.

Acknowledgments Thanks to the anonymous reviewers and our shepherd, Jia Wang, for their valuable feedback. This work was sponsored in part by the NSF under CNS-1405697 and CNS-1422788 grants.

References

[1] A Microscope on Microservices. http://goo.gl/1BXHRH.

[2] Advanced Message Queuing Protocol. https://www.amqp.org/.

[3] Ceilometer Quickstart. http://goo.gl/LEYBiM.

[4] Ceilometer Samples and Statistics. http://goo.gl/u2hBKe.

[5] OpenTSDB as a metering storage for OpenStack Telemetry. https://goo.gl/3I6L6J.

[6] Rethinking Ceilometer metric storage with Gnocchi: Time-series as a Service. https://goo.gl/68jXaO.

[7] Symbolic Aggregate approXimation. http://goo.gl/30X8f1.

[8] The R Project for Statistical Computing. http://www.r-project.org/.

[9] The SYSSTAT utilities. http://goo.gl/21243V.

[10] Tivoli Application Dependency Discovery Manager. http://goo.gl/tO8a4b.

[11] Writing rules for the state correlation engine. http://goo.gl/Z5McMT.

[12] M. Agarwal, M. Gupta, V. Mann, N. Sachindran, N. Anerousis, and L. Mummert. Problem determination in enterprise middleware systems using change point correlation of time series data. In *IEEE/IFIP NOMS*, 2006.

[13] A. Anwar, A. Sailer, A. Kochut, C. O. Schulz, S. Alla, and A. R. Butt. Cost-aware cloud metering with scalable service management infrastructure. In *IEEE CLOUD*, 2015.

[14] A. Anwar, A. Sailer, A. Kochut, C. O. Schulz, S. Alla, and A. R. Butt. Scalable metering for an affordable it cloud service management. In *IEEE IC2E*, 2015.

[15] A. Brinkmann, C. Fiehe, A. Litvina, I. Luck, L. Nagel, K. Narayanan, F. Ostermair, and W. Thronicke. Scalable monitoring system for clouds. In *IEEE/ACM UCC*, 2013.

[16] C. Canali and R. Lancellotti. Automatic virtual machine clustering based on bhattacharyya distance for multi-cloud systems. In *International workshop on Multi-cloud applications and federated clouds*. ACM, 2013.

[17] C. Chen, P. Maniatis, A. Perrig, A. Vasudevan, and V. Sekar. Towards verifiable resource accounting for outsourced computation. In *ACM SIGPLAN Notices*, 2013.

[18] R. Iyer, R. Illikkal, L. Zhao, D. Newell, and J. Moses. Virtual platform architectures for resource metering in datacenters. *ACM SIGMETRICS*, 2009.

[19] X. Jiang and X. Wang. out-of-the-box monitoring of vm-based high-interaction honeypots. In *Recent Advances in Intrusion Detection*, pages 198–218. Springer, 2007.

[20] S. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla. Efficient constraint monitoring using adaptive thresholds. In *IEEE ICDE*, 2008.

[21] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

[22] M. Liu and X. Ding. On trustworthiness of cpu usage metering and accounting. In *IEEE ICDCSW*, 2010.

[23] S. Meng, A. K. Iyengar, I. M. Rouvellou, and L. Liu. Volley: Violation likelihood based state monitoring for datacenters. In *IEEE ICDCS*, 2013.

[24] S. Meng, A. K. Iyengar, I. M. Rouvellou, L. Liu, K. Lee, B. Palanisamy, and Y. Tang. Reliable state monitoring in cloud datacenters. In *IEEE CLOUD*, 2012.

[25] W. Richter, C. Isci, B. Gilbert, J. Harkes, V. Bala, and M. Satyanarayanan. Agentless cloud-wide streaming of guest file system updates. In *IEEE IC2E*, 2014.

[26] P. Siirtola, H. Koskimäki, V. Huikari, P. Laurinen, and J. Röning. Improving the classification accuracy of streaming data using sax similarity features. *Pattern Recognition Letters*, 32(13):1659–1668, 2011.

[27] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 2008.