A Big Data Framework for Cloud Monitoring

Saeed Zareian
School of Information Technology
York University
Toronto, Canada
zareian@yorku.ca

Marios Fokaefs
Dept. of Electrical Engineering and Computer Science
York University
Toronto, Canada
fokaefs@yorku.ca

Hamzeh Khazaei
Dept. of Electrical Engineering and Computer Science
York University
Toronto, Canada
hkh@yorku.ca

Xi Zhang
College of Computer Science
Sichuan University
Chengdu, China
zhangxi@stu.scu.edu.cn

Marin Litoiu
Dept. of Electrical Engineering and Computer Science
York University
Toronto, Canada
mlitoiu@yorku.ca

ABSTRACT

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Keywords

1. INTRODUCTION

With the advent of cloud technologies and the virtualization of computation resources, there is an increase in efficiency of developing, delivering and maintaining web software. Software providers can reserve, decommission and repurpose resources on demand and at will with minimal or no external effect to the application or the end-users. This flexibility has prompted cloud providers to invest in offering autonomic cloud management systems. Largely based on IBM's MAPE-K concept [4], autonomic management systems provide the capabilities to constantly monitor and analyze the performance and status of the deployed application and its supporting infrastructure and promptly react to changes that will negatively affect the application's behaviour. For example, an influx in incoming traffic may saturate the virtual machines and result in dropped requests and high response time.

A key component of any autonomic management system is *monitoring*. Monitoring is the module responsible for gathering all the metrics characterizing the performance and "health" of the deployed application. Metrics can be gathered on the infrastructure level, including CPU and memory utilization, network and disk throughput, and on the application level, including response time and availability among others.

Monitoring data indicates signs of big data properties in the following aspects:

SIGMETRICS
Copyright is held by author/owner(s).

- *Volume:* If we monitor dozens of metrics per application and we sample every few seconds or every few minutes, within an hour we may have generated a few gigabytes of data only for a single application. The rate and amount of data generated implies that the monitoring system should have high throughput and data storage capabilities to handle this data.
- *Velocity:* Apart from the rate with which monitoring data is generated, it is also imperative that this data is analyzed as fast as possible, so that the system can efficiently react to any negative changes. The whole concept of cloud adaptation is based on the fact that a system can reengineer its topology as fast as changes occur.
- *Variety:* With a number of monitoring agents, deployed on different levels and measuring on different rates, it is natural to have various formats and sizes of data samples. All these have to be synchronized and aggregated, so that they become available for analysis.
- Veracity: Accuracy and robustness of monitoring data is crucial in the success of an autonomic management system. False, inaccurate or badly-timed monitoring data may result in wrong adaptive actions and by extension in additional and unforeseen costs and lower software performance.

Given these properties, we propose our own monitoring component for autonomic cloud management systems. Our solution makes three distinct and novel contributions:

1. **Use of big data technologies:** We employ big data solutions to address the aforementioned challenges caused by the volume and velocity of monitoring data. More specifically, we use HBase to store the data, which is partitioned in a cluster of virtual data servers for efficient storage, faster retrieval and data integrity. Additionally, HBase is tightly coupled with highly efficient big data analytics including Hadoop and Apache Spark.
2. **Extendible, pluggable and layered architecture:** The component consists of various layers responsible

for measuring, aggregation, scheduling, storing and retrieving monitoring data. Every layer can be easily replaced by different technologies, since they are based on RESTful APIs for lightweight communication and decoupling. Moreover, its pluggable architecture allows for connecting various monitoring agents responsible for specific metrics or for different levels, including VM or application.

3. **BigQueue:** Because of the potentially large number of measuring agents and their different monitoring rate, their write requests to HBase needs to be orchestrated and synchronized. If all agents issued write requests independently, we would have too many simultaneous open connections to the database, which could cause big performance issues. For this reason, we introduce the *BigQueue* component, which serves as a write buffer; the agents commit their data to the buffer and after a time or volume threshold, BigQueue pushes the data from all agents to HBase. This way we have only one connection to the database.

The monitoring solution has been implemented as a component in our own MAPE-K implementation, called K-Feed [16], which is deployed in the SAVI testbed [10], a research cloud environment built on top of OpenStack. We conducted a set of experiments to evaluate the efficiency and general performance of our solution and of BigQueue, more specifically. Additionally, we identified another potential bottleneck in the fact that HBase uses one mater node for multiple worker nodes. Therefore, we repeated our experiments using Apache Accumulo[1], which uses a multi-master model, as the data store solution and compared the two deployments.

The rest of the paper is organized as follows. Section 2 reviews related works on cloud monitoring. Section 3 presents the proposed monitoring module, along with BigQueue and its functionality. In Section 4, we discuss the results of our experiments with HBase and Accumulo on enabling efficient monitoring. Finally, Section 5 concludes this work.

2. RELATED WORK

The area of this work is a point of interests in both academia and industry. Therefore, we will present the related works in two sections to focus well on academic and industrial solutions. Developing autonomic systems and investigating performance metrics has been the topic for a number of research works. In the SAVI testbed [10], there exists a project with the goal of monitoring the virtualized resources in the cloud such as storage and network [6]. In this project, called MonArch, Lin et al. proposed a new architecture for providing monitoring as a service (MaaS). This service is an integration of OpenStacks popular monitoring service, called Ceilometer, and some custom processes obtained from Apache Spark. Although in this work the architecture is extendable by adding new user agents, the main advantage of our work comparing to this project is its portability in over different datastores. In this work Cassandra is used as the datastore but in our paper, we compare different datastore environments and their performance. Yongdnong et al. [15], implemented their project to interact with multiple third-party monitoring software and read their output from their database and store them in MongoDB. In this way, they could merge the data and access them via an administrative query using an abstraction layer. Thus, they used whole deployment of monitoring services and instead of having their own monitoring component. Carvalho et al. [2] focused more on the discovery of particular resources that are shared in the Openstack cloud, called cloud slices, to find the allocated slices per user and then run monitoring on them. Therefore, their focus is nor portability and extendibility of the monitoring service, neither the monitoring big data ingestion. However, Smit et. al [12] designed a Monitoring-as-a-service (MaaS) framework called MISURE to show a proof of concept for leveraging the stream data processing for watching multiple monitoring sources with low overhead and high throughput. MISURE also provides custom endpoint for agents to push their monitoring data and it uses OpenTSDB as database. OpenTSDB uses HBase as its backed storage. However, in our work we will introduce a issue when we are receiving high throughput of data in HBase that is not addressed in this work. Meng et al. [8] tried to improve their previous work [7] to increase the efficiency of their MaaS solution. In their first work, they designed a statistical approach to detect the SLA violation using a window average of data. Then, in their subsequent work, they tried to integrate their concepts with cloud analogies. They evaluated their works using different standard workloads and showed that their solution could keep the SLA violation unlikely. Therefore, their work does not propose any architecture or software design for high throughput of data. Konig et al. [5] focused on real time metric data and monitoring query processing by defining a workflow to split the tasks related to query among the monitoring agents and fetch and combine the results. Therefore, their work lacks a central databases that is vital for complex monitoring queries. Anand [1], also defined another architecture for monitoring the cloud VMs. In his work, he installed agents on the VMs that push samples to a monitoring server; the performance of this architecture is not examined.

In industry, cloud computing has seen an increase in positive attention and endorsement from major players in the technology world over the recent years. Ceilometer [3] is designed to be a single point of contact for billing software to collect and report the OpenStack component resources usage. The main goal of Ceilometer is to be used by billing software in cloud. Thus, the analytic are as simple as min, max and sum function and the monitoring process performance is O(n) that n is the number of monitored resources. Monasca project [9], whose name is derived from Monitoring-at-Scale, designed to be multi-tenant and highly scalable compared to Ceilometer. Ceilometer polling method becomes slow when the number of VMs increases. This project is more independent from OpenStack and Ceilometer and can be deployed as a stand-alone Monitoring-as-a-Service in various cloud environments. Another feature of this open-source project is the anomaly and metric prediction service. Cloudwatch [11] is another project by Amazon to monitor applications, services and resources in its Amazon Web Services cloud environment. It has the same concepts that are discussed in previous sections. Agents gather the metric samples and store them in a central database to be accessed by the user or the alarm service. Nevertheless, it is a health monitor service in Amazon cloud environment and it is not portable to other cloud platforms.

[1] `https://accumulo.apache.org/`

3. HIGH THROUGHPUT CLOUD MONITORING MODULE

The proposed monitoring framework follows a layered architecture starting from the measuring agents down to HBase, our big data storage solution of choice. Figure 1 shows the overall stack of components in the framework. In the next sections, we will outline the characteristics of each one of them starting from the bottom, with particular emphasis and more details on the novel BigQueue component.

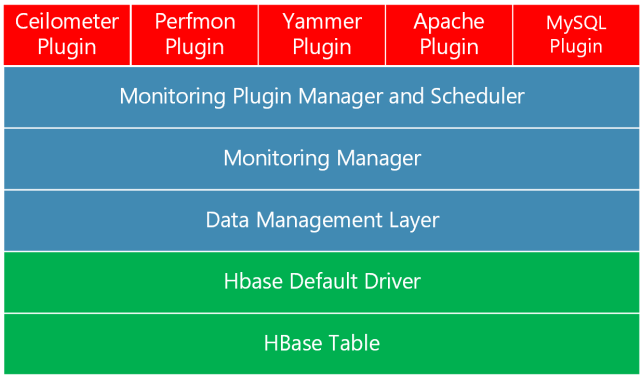

Figure 1: Monitoring framework stack of components.

3.1 General Architecture

3.1.1 HBase Table

HBase is a cluster-based distributed database system and part of the Hadoop software stack [14]. It can integrate very well with other software systems in the stack such as Pig, Hive, Mahout, and so on.

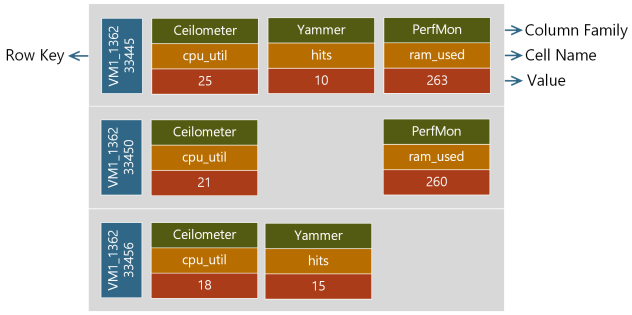

Figure 2: HBase schema.

Since HBase is a NoSQL database, it needs a targeted and well-specified design for its database schema to store the data in an effective way. The abstraction of the schema we designed for our solution is depicted in Figure 2. Each monitoring agent is considered as one column family. Therefore, for each source of monitoring such as Yammer, Ceilometer or any other pluggable monitoring services, there will be a new column family. Given that the Yammer agent is specific to one application and since we may have multiple applications on one VM, we can define multiple Yammer column families with the application id as prefix. Row keys are compound keys containing the VM's hostname and the Unix timestamp of the measurements in each row with the granularity of a second. Each cell represents the value of each metric sample. The cell name is the metric name.

3.1.2 HBase Default Driver

We used Cloudera[2] to deploy the HBase data storage cluster on the SAVI cloud. Libraries provided by the Cloudera manager are used by the Data Management Layer to connect to the HBase cluster, and then, through the HBase Java API, it can store and retrieve monitoring data.

3.1.3 Data Management Layer

This layer manages incoming data to HBase and outgoing data from the storage. Incoming data is managed by the BigQueue component to guarantee high throughput, as we will describe next. The layer is also responsible for making the necessary data transformation so that they can be provided as input to analytics software. Indicatively, in Apache Spark, we can directly connect HBase to Spark processing jobs using its Java resilient distributed dataset (RDD) abstraction layer to parallelize the data loading in a fault tolerant way.

3.1.4 Monitoring Manager

The Monitoring Manager is the backbone of the architecture as it is responsible for initializing the upper layers including the Monitoring Plugin Scheduler and connect them to the lower layers, such as the Data Management Layer. When deployed and configured along with the application to be monitored, the Monitoring Manager receives as input the cloud topology in terms of number and types of virtual machines and the software deployed in each VM, including the application itself, application and database servers, load balancers and so on.

Given the modular architecture of the framework and the API-enabled communication between the modules, the Monitoring Manager is independent from the data storage or the measuring agents. Therefore, regardless of whether these modules are changed or new ones are added, the Monitoring Manager does not need to change.

3.1.5 Monitoring Plugin Manager and Scheduler

This layer is responsible for organizing and orchestrating the various measuring plugins. The Plugin Manager receives the topology and software configuration from the Monitoring Manager and creates a list of the necessary agents to be deployed in each VM and each application to monitor machine-related metrics (e.g. Ceilometer and PerfMon) or application-level metrics (e.g. Yammer). This list is then passed to the Scheduler along with a default monitoring rate. Since each agent may be gathering metric samples at a different rate, then Scheduler needs to be aware of this information for each individual agent.

The Scheduler uses the Play Framework to set up safe and non-blocking threads for each plugin instance. In addition, the Scheduler sets each plugin to run with a small delay after the previous one so that we don't have network blocking. The final schedule of execution of the plugins is then returned to Plugin Manager.

[2] http://www.cloudera.com/content/www/en-us/documentation/archive/cdh/4-x/4-3-1/CDH4-Installation-Guide/cdh4ig_topic_20.html

The Plugin Manager instantiates and deploys all the individual agents and initializes them according to the predefined schedule. Afterwards, according to the set intervals, the Plugin Manager queries the agents for their metric samples, which then passes down to the Monitoring Manager, so that they are stored in the database.

3.1.6 *Monitoring Plugins*

Given the amount and variety of resources, like VMs and software, and metrics that need to be monitored, there is an increasing need for a number of monitoring agents. The pluggable nature of the proposed layered architecture allows for the integration of any number and any type of monitoring agent. However, since the agents cannot be seamlessly integrated with the framework, plugins are needed to be implemented first. Plugins can be perceived simply as middleware that specify an API for a given agent, which is exposed to the Plugin Manager, so that the latter can query the former for metric samples.

In the context of this work, and for the purpose of our experiments, we have implemented several plugins for a number of monitoring agents.

- **Ceilometer plugin:** This plugin uses a RESTful API to access OpenStack Ceilometer API. It connects through Keystone, the Openstack authentication service, and then sends a query to the Ceilometer endpoint URL. For each VM and each metric, a query is sent to the service and the results are collected. For collected metrics, normalization and conversion are performed to keep them as metric sample Java objects. The objects are pushed toward the Data Management Layer (i.e., BigQueue).

- **Perfmon plugin:** Perfmon (performance monitoring) is utilized to exploit the power of agents based on System Information Gatherer And Reporter (SIGAR) [**?**] API deployed in VMs. Each five seconds, the plugin sends a request via an SSH connection to the installed agent on each VM in parallel using threads. The response is a group of values for the metrics. The plugin pushes the metric sample objects to the BigQueue.

- **Yammer/Dropwizard API plugin:** Dropwizard introduces a library to have both JVM-related metrics and custom metrics. Custom metrics can be, for example, servlet execution time or number of web page hits. The values are represented in a URL endpoint in JSON format. The values are extracted by this plugin for each server periodically and pushed to the BigQueue as metric sample objects.

- **Apache Load Balancer plugin:** This plugin is inherited from the PerfMon plugin, but it also reads the Apache load balancer statistics page.

- **MySQL DB plugin:** This plugin is also inherited from PerfMon plugin, but it is specialized to read the MySQL-related statistics.

3.2 BigQueue Component

Considering the number of VMs and supporting software that a cloud application may consist of, along with the number of metrics each agent needs to monitor per unit of time, for example every second or every minute, we can easily understand the potentially high number of requests that the Monitoring Manager needs to handle and pass to the data storage system. For example, in the K-feed platform, there is a need to insert samples for around 150 different metrics per application and VM every 5 seconds, which is a predefined interval.

There are several challenges pertaining the task of pushing monitoring to a database.

1. Storing data in traditional and single-node database systems such as MySQL depends on a single bottleneck: server process throughput. Thus, for making our monitoring component scalable by having multiple database servers, we need to use a cluster-based solution. This was the original motivation behind using a distributed solution like HBase.

2. Although HBase is a multi-node storage, it has its own complexities such as storing on top of HDFS and also replicating data among the nodes. All connections from monitoring agents to commit their data are established through HBase master server. Therefore, the master node can become another bottleneck for the storage throughput. This led us to experiment with a multi-master NoSQL solution like Apache Accumulo [13] to explore whether such a setting will give us an improved throughput rate.

3. The default driver for HBase provides Java APIs to interact with the database system. The Cloudera APIs provide a Java interface that creates a thread and establishes a TCP connection to insert or delete data from the database. When it comes to large numbers of interactions, as in the case of monitoring agents, making new threads and TCP connections in parallel may block and slow down the operation. A potential solution can be to use a queuing system for the data flow. As a result, the monitoring service can collect information independently and push the data samples whenever there is a need for storage. This is the motivation behind the BigQueue component.

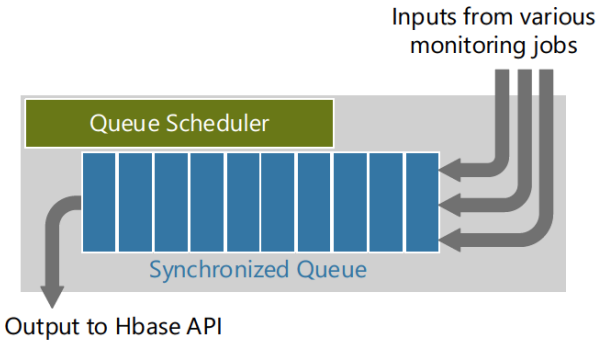

Figure 3: BigQueue architecture.

BigQueue facilitates the storage of metrics as depicted in Figure 3. Monitoring plugins can push metric samples as Java objects with the corresponding API to the internal queue. The internal queue used inside the component is designed to be thread-safe. This means multiple threads

can access the queue without considering the common challenges in simultaneously accessing the queue, such as overwriting each other's data. There are two policies to flush the queue and send the data out to HBase table. The first one is a threshold value defining the maximum number of metric samples to be kept. Any time the number of samples reaches the threshold value, the scheduler flushes out the whole queue via one single batch request to HBase. Therefore, there will be only one request for a large number of samples. The other policy is the time duration for keeping the samples. Every minute, the values are sent out to the HBase cluster to store persistently the data in the database. This is particularly useful when the system is not that busy, and the queue data has not reached the threshold value.

4. PERFORMANCE EVALUATION OF DATA MANAGEMENT LAYER

Due to layered architecture of the monitoring system, each layer can be replaced or modified with minimum effects on adjacent layers and virtually no effects on the others. Initially we setup the monitoring system with an HBase cluster; due to poor performance with default driver, we employed BigQueue in order to improve the performance of the data ingestion. As described in section 3.2 the problem stems from provisioning multiple concurrent connections by the Master node. As HBase is using a single master architecture, we mitigate the problem by implementing the BigQueue. Here we leveraged the layered architecture and substituted HBase with Apache Accumulo that is a multi-master columnar NoSQL datastore. The main intention for this was to see if BigQueue is still improving the data ingestion performance or no longer has a tangible effect. In the following sections, we compare the performance of default drivers and using the BigQueue in terms of insert operation delay for both HBase and Accumulo. Table 1 and 2 describe the specifications of the HBase and Accumulo clusters as well as virtual machines respectively.

Table 1: Specifications of HBase and Accumulo clusters.

Cluster Name	HBase	Accumulo
Gateway Node	N/A	1 (large)
Master Node	1 (xlarge)	2 (medium)
Data Node	4 (large)	4 (large)

Table 2: Virtual Machines (VM) specifications.

Name	xlarge	Large	Medium
vCPU	8	4	2
Disk (GB)	160	80	40
RAM (GB)	16	8	4

4.1 Apache HBase

In this section we investigate the effect of using the BigQueue component on the performance of inserting data in HBase. For this purpose, two experiments are designed. First, the data management layer (i.e., BigQueue) is bypassed, while, in the second experiment, the data management layer is utilized. The experiment starts from 300 rows per second and continues until the insertion of 10000 rows per second. In each iteration, we increase insertion rate by 100 rows per second.

4.1.1 *Storing Monitoring Data via Default Driver*

In the first experiment, we measure the performance of insert operation using default configuration of HBase. As shown in 4, the data insertion time increases linearly with the amount of rows that are being inserted. Also, as it can be seen, the insertion time increases up to 35 seconds to the maximum.

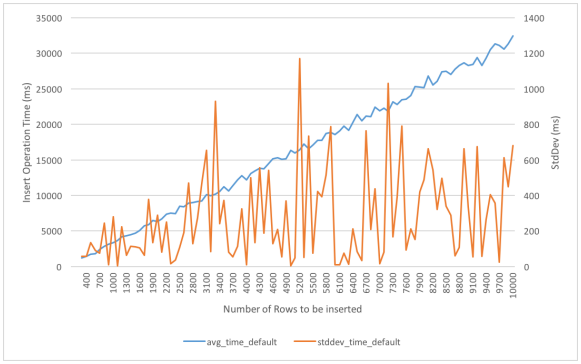

Figure 4: Operation delay for insert and the standard deviation for default driver.

4.1.2 *Storing Monitoring Data via BigQueue Component*

In this experiment, we measure the data insertion delay by leveraging the BigQueue component. Using the queue idea and batch processing for ingesting data into HBase master server, the overhead for making a new TCP connection to HBase master is reduced and, as a result, as can be seen in 5, the BigQueue component decreased the loading time almost 200 times.

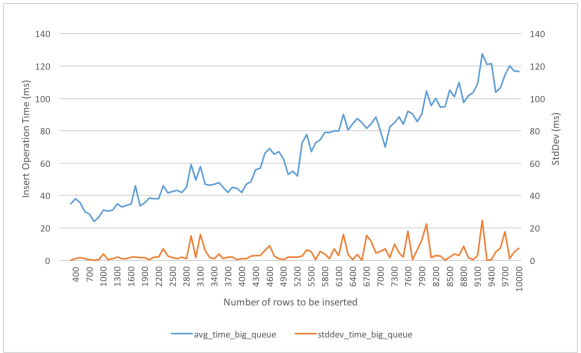

Figure 5: Operation delay for insert and the standard deviation using BigQueue.

Figure 5 clearly shows that for almost the same amount of data, using BigQueue component that provides batch loading makes the process more than 200 times faster. However, there is another issue revealed by these two experiments; the growth of data insertion time is inevitable because of

the data size, but its growth is not equal or even close to the growth of the first experiment. In Figures 4 and 5 we showed the average value of insert time for three iterations of the same experiment as well as the standard deviation on of the insertion time values. The standard deviation shows that without using BigQueue the insert operation will be more unstable and the time it takes can be more variable.

4.2 Apache Accumulo

In this section we investigate the effect of leveraging the BigQueue component on data ingestion performance in Accumulo. For this purpose, we repeated the two experiments that we carried out for HBase.

4.2.1 Storing Monitoring Data via Default Driver

As can be seen in Figure 6, Accumulo performance is much better than HBase under the default configuration and driver. The insertion delays increases slowly with increasing the pressure on the datastore step by step. Accumulo assigns a batch buffer to each plugin and keeps the connection alive until it closes explicitly by the client and flushes this buffer to the storage after a predefined time interval. We made the Accumulo to persistent the data after each step.

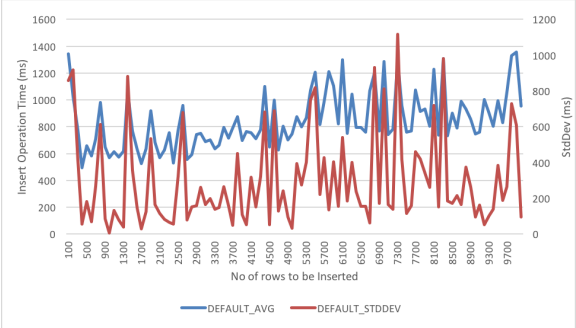

Figure 6: Operation delay for insert and the standard deviation for default driver.

4.2.2 Storing Monitoring Data via BigQueue Component

In spite of very good performance by Accumulo for concurrent data ingestion, still BigQueue manged to speed up the process up to 5 times (Figure 7). In this configuration, we used BigQueue and made it to push the data into a batch buffer in Accumulo after each step and then the batch flushes the data into storage immediately.

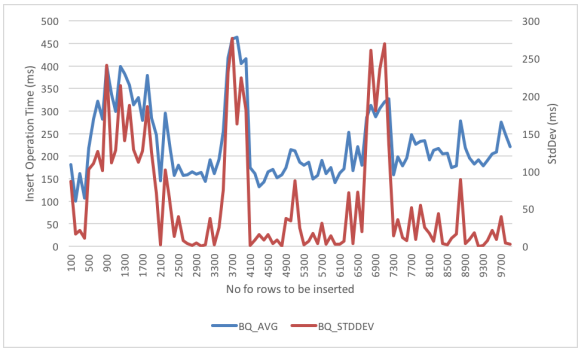

Figure 7: Operation delay for insert and the standard deviation for default driver.

5. CONCLUSIONS

6. ACKNOWLEDGMENTS

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] M. Anand. Cloud monitor: Monitoring applications in cloud. *IEEE Cloud Computing for Emerging Markets, CCEM 2012 - Proceedings*, pages 58–61, 2012.

[2] M. B. De Carvalho, R. P. Esteves, G. Da Cunha Rodrigues, L. Z. Granville, and L. M. R. Tarouco. A cloud monitoring framework for self-configured monitoring slices based on multiple tools. *2013 9th International Conference on Network and Service Management*, pages 180–184, 2013.

[3] O. Foundation. Ceilometer System Architecture. `http://docs.openstack.org/developer/ceilometer/architecture.html`, 2015.

[4] P. Horn. Autonomic computing: Ibms perspective on the state of information technology. 2001.

[5] B. Konig, J. Alcaraz Calero, and J. Kirschnick. Elastic monitoring framework for cloud infrastructures. *IET Communications*, 6(10):1306, 2012.

[6] J. Lin, R. Ravichandiran, H. Bannazadeh, and A. Leon-garcia. Monitoring and Measurement in Software-Defined Infrastructure. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 742–745, 2015.

[7] S. Meng, A. K. Iyengar, I. M. Rouvellou, L. Liu, K. Lee, B. Palanisamy, and Y. Tang. Reliable state monitoring in cloud datacenters. *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, pages 951–958, 2012.

[8] S. Meng, S. Member, L. Liu, and S. Member. Enhanced Monitoring-as-a-Service for Effective Cloud Management. *Journal of IEEE Transactions on Computers*, 62(9):1705–1720, 2013.

[9] Opesntack Foundation. Monasca Architecture. `https://wiki.openstack.org/wiki/Monasca{\#}Architecture`, 2015.

[10] SAVI. Cloud platform. `http://www.savinetwork.ca`, June 2015.

[11] A. W. Services. Amazon Cloudwatch Architecture. `http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/cloudwatch_architecture.html`, 2015.

[12] M. Smit, B. Simmons, and M. Litoiu. Distributed, application-level monitoring for heterogeneous clouds using stream processing. *Future Generation Computer Systems*, 29(8):2103–2114, 2013.

[13] The Apache Foundation. Apache accumulo. `http://accumulo.apache.org`, 2015.

[14] The Apache Foundation. Welcome to apache hbase. `http://hbase.apache.org`, 2015.

[15] H. Yongdnog, W. Jing, Z. Zhuofeng, and H. Yanbo. A Scalable and Integrated Cloud Monitoring Framework Based on Distributed Storage. *2013 10th Web Information System and Application Conference*, pages 318–323, 2013.

[16] S. Zareian, R. Veleda, M. Litoiu, M. Shtern, H. Ghanbari, and M. Garg. K-feed - a data-oriented approach to application performance management in cloud. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 1045–1048, June 2015.