An Effective Multivariate Control Framework for Monitoring Cloud Systems Performance

Ismail Hababeh[1*], Anton Thabain[2] and Sahel Alouneh[3]
[1,2,3] German Jordanian University - Faculty of Electrical Engineering and Information Technology
Amman, 11180 - Jordan
[1][e-mail: ismail.hababeh@gju.edu.jo]
[2][e-mail: anton.thabaine@hotmail.com]
[3][e-mail: sahel.alouneh@gju.edu.jo]
*Corresponding author: Ismail Hababeh

Received July 18, 2017; revised November 27, 2017; revised January 27, 2018; accepted April 11, 2018; published January 31, 2019

Abstract

Cloud computing systems' performance is still a central focus of research for determining optimal resource utilization. Running several existing benchmarks simultaneously serves to acquire performance information from specific cloud system resources. However, the complexity of monitoring the existing performance of computing systems is a challenge requiring an efficient and interactive user directing performance-monitoring system. In this paper, we propose an effective multivariate control framework for monitoring cloud systems performance. The proposed framework utilizes the hardware cloud systems performance metrics, collects and displays the performance measurements in terms of meaningful graphics, stores the graphical information in a database, and provides the data on-demand without requiring a third party software. We present performance metrics in terms of CPU usage, RAM availability, number of cloud active machines, and number of running processes on the selected machines that can be monitored at a high control level by either using a cloud service customer or a cloud service provider. The experimental results show that the proposed framework is reliable, scalable, precise, and thus outperforming its counterparts in the field of monitoring cloud performance.

Keywords: performance monitoring system, inter-process communication, machine data grid service, threading, performance object service, history data file

1. Introduction

Performance Monitoring System (PMS) greatly impacts a system's continuity and its business perspective [1]. A PMS explores system operating processes and investigates their functions in order to give system administrators an opportunity to discover systems faults, network problems, and communication disruptions [2-3]. Indeed, cloud system performance and security management require monitoring capabilities to guarantee system availability, scalability, and functionality [4].

PMS tasks vary from the simple, such as performing automated system inspections using open-source software, to the complex, such as physical component testing in order to enhance overall system performance. In addition, a PMS can investigate when a business reaches its ultimate support capacity and alert for that additional or new technology is needed [5-6]. Monitoring cloud computing systems is needed for continuous measuring, assessing, and improving the performance of the applications and their infrastructure behaviors [7]. The main objective of cloud monitoring systems is to obtain or provide the highest performance at the lowest cost [8].
Cloud monitoring system can be modeled in seven layers; facility, network, hardware, operating systems, middleware, application, and the user that can be monitored and controlled by either a cloud service customer or a cloud service provider [9]. At abstraction level, monitoring cloud system could be viewed at both high and low monitoring levels. A high-level monitoring represents the information on the status of real or virtual platform running cloud applications, where a low-level monitoring represents the status of the cloud physical infrastructure in terms of CPU speed, memory utilization, and network workload [10]. Moreover, cloud performance monitoring can be classified according to tests and metrics into two types; computation-based and network-based monitoring [11].

Basically, the knowledge gained from cloud monitoring is based on performance computation metrics that are considered a standard measures to assess cloud system performance [12]. Monitoring performance metrics then diagnose if the cloud system operates in a way that satisfies the objectives of its business, operations, and infrastructure functionality [7]. Such performance metrics support and help users, designers and administrators to better decide on suitable action(s) for any performance fault that might occur through running applications [13].

As the field of cloud computing is becoming more competitive, it is more important than before to ensure that the selected performance metrics provide the best performance-cost trade-off. However, there are many types of cloud systems so that it is not feasible to have a standard measure of performance metrics for different cases.

In literature, the approaches [9][14-16] present different methods for monitoring cloud systems. Each monitoring approach mainly focuses on a subset of performance features in order to manage huge amount of dynamic monitored data to achieve high cloud systems performance [14]. These monitoring approaches investigate specific cloud platforms by controlling services that assess the effective performance metrics at hardware, operating systems, application, and user layers [9]. The metrics considered by these approaches include CPU speed, virtual machine startup time, server and memory throughput, and storage utilization [16]. Performance metrics are often imply interactive data testing that can be evaluated in terms of availability, reliability, accuracy, extensibility, atomicity, adaptability and timeliness [15].

The metrics measurement accuracy is also considered a key issue in cloud performance, as it is necessary to perform the operations that make use of the monitoring information effectively and efficiently [17]. Therefore, cloud machines require time scheduling technique in order to have an accurate time stamping when switching between different monitoring machines.

Nevertheless, the current approaches assume a subsystem or rely on other agents for collecting, filtering and aggregating performance monitored data. This may violate the privacy and security when the monitored data classified as confidential. Moreover, performance metrics for fault tolerance and scalability are not considered by the current benchmark approaches. Consequently, the monitored systems should be able to manage and control a large number of cloud machines efficiently and effectively in a short measurement time.

To this end, we consider the effective and commonly used performance metrics that are broadly applicable to a wide range of cloud applications. The metrics should truly and accurately measure the hardware performance metrics such as CPU usage, RAM availability, number of cloud active machines and the number of running processes on the selected machine. CPU usage is defined as the percent operation of the cloud machine CPU during available time period of its functionality. RAM availability is described as the obtainable memory storage on specific cloud machine over a certain time period where the monitoring has occurred. The number of cloud machines represents the number of active cloud machines under monitoring. The number of running processes indicates how many processes are currently running on the active cloud machine. Cloud system users may be familiar with their system performance; however, some are not technically aware of how to use multivariate-benchmarks [18]. Benchmarks can therefore help in extracting performance information where components are embedded in the system that identifies the machines' metrics or distributes them into different structural components. However, the result is a large overhead, which add other challenges since the performance of various sets of machines needs to be determined [8 to 19].

Performance monitoring within a threaded [20] or a heavily parallelized environment is also a major challenge. It requires overlooking some features that are not needed or they have less impact on the decisions made by the cloud system user. In this scenario, the complexity should be hidden, and performance monitoring should be done in the background. Only important performance information that is familiar should be presented to the users, such as CPU, RAM, and PING [21]. This is in addition to the number of programs or processes that are currently running on the cloud system.

To determine and provide reliable performance information about a cloud system, it is not necessary to analyze and interpret the data in real time [22]. Therefore, in this research, the concept of data synchronization in parallel semi-real time systems [23] is proposed in the context of the use of data synchronization within a parallel threaded environment [24]. Semi-real time in this context means a millisecond based processing interval.

Benchmarking metrics and the complexity of the system task manager are given a high usage and are compartmentalized within our proposed system. The data is extracted from the operating system and then processed and presented to the user in a simplified GUI [25]. In such a system, the factors controlling processes or machine performance are not included.

Most cloud system users are already familiar with monitored operating systems; therefore, the proposed system has the potential to provide them with powerful new monitoring services that can easily define the performance metrics.

In this paper, we discuss the performance metrics that are needed to design a powerful integrated monitoring system. Such a system is designed to collect benchmark measurements to produce a comprehensive performance monitoring system. In addition, we developed a Performance Monitoring System as a Service (PMSaS) for collecting data from a cloud system that is controlled by heterogeneous operating systems. Our proposed monitoring system displays the performance results in terms of meaningful graphics, stores the graphical information in a database, and provides the data on-demand.

The rest of the paper is organized as follows: in Section 2, we review the previous related work in literature formonitoring cloud computing systems performance. Section 3 addresses our performance-monitoring framework. After that, we present and discuss the experimental results and performance evaluation of monitoring real time cloud systems in Section 4. Finally, conclusions and future research directions are presented in Section 5.

2. Related Work

In our research, we have reviewed studies, patents and methods that have touched on or thoroughly assessed computer performance in various ways. Among them, we have seen both ideas about injecting the system with benchmarks and concepts about specialized performance monitoring tools. Our goal of having a universal and easy to use system for both IT professionals and basic users has provided us with a general approach that will reduce and hide complexity by utilizing the systems' native tools. Within this context, we compare our performance monitoring ideas with those of the other studies in the field to provide logical and valuable insights to a system that simplifies this task as much as possible.

Performance Monitoring is defined as part of the evaluation of system performance for the purpose of determining the utilization of software and hardware resources, such as CPU and memory use [18]. The authors compare changes in system performance with that of the system's reference state to be able to investigate the behavior of current resources in a timely manner. However, extracting CPU and memory consumption is complex, which reflects performance information inconsistency.

The patent [26] concerning benchmarks discloses how to retrieve system performance information. This type of information helps when using benchmarks and micro-benchmarks that highlight the performance of separate parts or the entire machine. While this can be effective, the user may not have enough technical knowledge about such cases and is not able to use the injected micro-benchmarks. Instead, the user will rely on operating system applications, such as the task manager, to measure machine performance.

The complicated design of the task manager discussed in [27] is not needed as much for a large system as it is for a small one. It is arguably one of the most powerful performance monitoring tools, yet it is limited to one system due to its complexity. However, monitoring various levels of system performance can be achieved by choosing not to monitor some process variables. In addition, the simplification of the performance monitoring system requires that less information be collected than that provided by the task manager.

The Microsoft Azure Fabric Controller [28] is a multi-layer monitoring system that selects the centralized network architecture to improve cloud system efficiency. However, Azure does not support monitoring heterogeneous cloud infrastructures.

The Paradyn project [29] develops the capabilities of a dynamic instrumentation technique which generates performance profiles of unmodified resources. This approach includes the Dyninst API running operating system kernels and the MRNet multicast network. It achieves scalability by adjusting instrumentation granularity. However, implementation of threaded programs on untested platforms may cause some bugs due to platform dependency when using the dynamic instrumentation technique.

The open source performance monitoring system Nagios [30] enables establishments to recognize the status of their network devices and alert systems administrators to problems. However, in order to exert control over network components and services, difficult text-based configuration files are required. Moreover, a confusing GUI and lack of database connectivity make it hard to use, especially in parallel systems.

The infrastructure performance monitoring system boundary framework is presented in [31]. This framework collects large amount of data about unit system performance from heterogeneous running systems and generates diagrams in real-time for visualization. However, the boundary suffers from granular performance analysis such that the organization needs to determine its operational requirements.

Most research in the field of performance monitoring shows the existence of time and cost overheads that degrade system performance especially when dealing with systems with a large number of distributed nodes. In our approach, the demand for an efficient framework to monitor such distributed systems has been addressed.

3. Performance Monitoring Service Design

In this paper, we present an approach to a cloud performance monitoring system which consists of integrated performance components that are designed to serve as easily replaceable plugins, thus allowing the system to be highly dynamic, quickly configured and easily adjustable to any cloud system environment. Our approach is designed based on a three-tier structure that defines the interaction and role of each performance component with other components within the boundaries of its tier level and across outer levels. In the following sections, we describe the performance monitoring structure in details.

3.1 Performance Monitoring Service Structure

The performance monitoring service PMS structure described below is based on three functionality tiers; Top, Middle and Bottom. **Fig. 1** presents the functionality of PMS tiers and the operations that are assigned to each tier.

- The bottom tier contains many services that are installed on the clients cloud machines, each machine communicates with other machines through its own firewall system and local network router.
- When the client service is initiated from the client machine, it produces an XML file that holds the client machine performance information.
- The XML file is then transmitted by the client machine and passes through the local firewall system to the local network router in the bottom tier.
- The local network router transmits the XML file packets to the Internet Service Provider (ISP) Router through the World Wide Web.
- The ISP Router transmits the XML file to the local network router in the middle tier.
- The local network router in the middle tier transmits the XML file through the server firewall to the secured FTP cloud server that responds to the monitoring machine requests in the top tier.
- A timer is set for few seconds to idle the client service in order to repeat the process and to allow the cloud server to communicate with the monitoring machine.
- The monitoring machine in the top tier triggers a monitor service request that passes through the local firewall system to the local network router.
- The local network router in the top tier transmits the monitor service request to the ISP router.
- The ISP router will again transmit the monitor service request through the World Wide Web to the cloud server local router advancing to the firewall and reaching the server itself.
- The server response passes through the ISP router of the middle tier to the World Wide Web reaching out to the ISP router of the top tier, then to the local monitor router through the firewall back to the monitoring machine that issued the request.

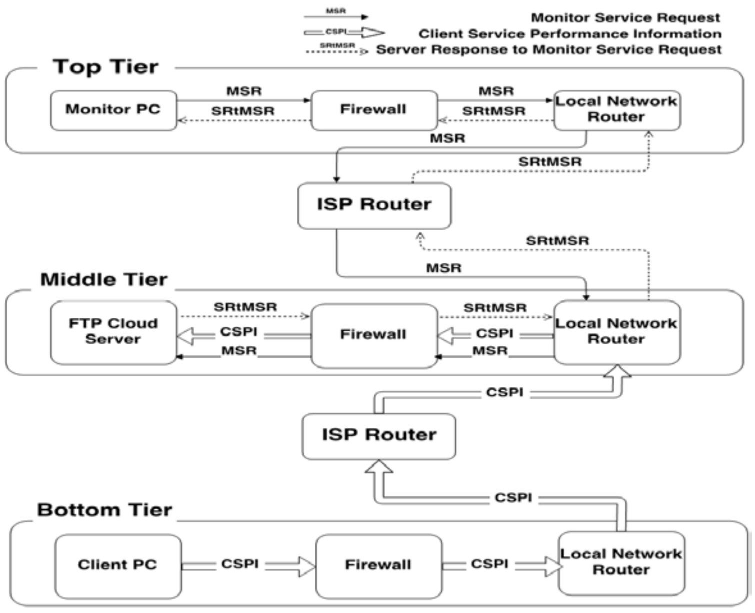

MSR Monitor Service Request
CSPI Client Service Performance Information
SRtMSR Server Response to Monitor Service Request
Top Tier
Monitor PC
MSR
SRtMSR
Firewall
Local Network Router
ISP Router
Middle Tier
FTP Cloud Server
CSPI
Bottom Tier
Client PC

Fig. 1. Performance Monitoring Service Functionality

The PMS tiers are illustrated in details as follows:

A. Top Tier

The top tier mainly consists of two components: The Monitor Service and the Desktop Application. The two components are defined as follows:

- **Monitor Service**

This service has the responsibility of retrieving information files from the server periodically and storing them inside a specified directory located on the monitoring machine. In addition, this service passes the information files to the desktop application [32] that initiated the call. Each machine-server call is triggered based on a desktop application request.

- **Desktop Application**

A graphical user interface (GUI) [33] presents the information from all clients to the monitoring administrator. This application spawns a background thread upon starting that is responsible for reading and parsing the received client files periodically. In addition, the GUI is updated with the parsed contents of all received files that are triggered through a call to the monitor service.

B. Middle Tier

The middle tier consists of a virtual machine running on a configured cloud server operating in a passive mode to allow the client to provide the server with both data and command ports, resulting in a simpler communication and connection environment. Moreover, this virtual machine allows the client firewall to configure itself and choose the specified ports on which the FTP connections will be established and run. In addition, the FTP server [34] has been configured to allow anonymous login, which authenticates any user connection to the server without the need to supply a username or password.

C. Bottom Tier

The bottom tier consists of the monitored client service that is responsible for collecting the information from the client machine, parsing this information into XML format [35-36], and then creating the XML file with the generated information. After the completion of the XML data file, it will be uploaded to the FTP server and saved as a history data file that keeps track of the latest updates on the monitored machines' performance information. The history data file is used to construct a graphical representation of monitored machine. The updating process of the history file is based on a queue data structure (FIFO First-In First-Out) [37]. The client service life cycle processes are run periodically as long as the machine and service are running. **Fig. 2** depicts the monitored client service life cycle processes in the bottom tier.

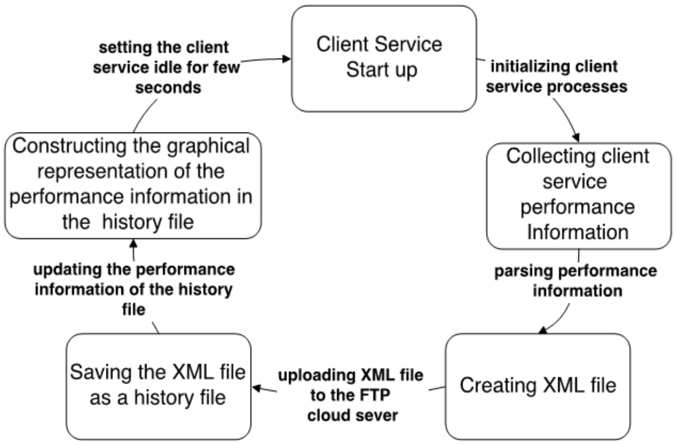

Client Service Start up
initializing client service processes
Collecting client service performance Information
parsing performance information
Creating XML file
uploading XML file to the FTP cloud sever
Saving the XML file as a history file
updating the performance information of the history file
Constructing the graphical representation of the performance information in the history file
setting the client service idle for few seconds

Fig. 2. Client Service Life Cycle processes

3.2 Communication Processes

In this section, we describe the PMS model and the framework of communicating

processes. Our goal is to make use of performance monitoring by extracting data directly from the monitoring machine's operating system. Our monitoring framework uses concurrent programming (threading) [38] to perform the performance computations. This can be done by instantiating dedicated threads to process specific information in order to produce the desired output, for example, parsing or upload operations.
The PMS is divided into multi-level components, and the overall system communication is then achieved through the following processes:

- **File Access**

  The PMS components work with files. The client rewrites the uploaded file or uploads a file that is retrieved from a GUI snapshot of the client file.

- **Inter-Process Communication**

  Inter-Process Communication (IPC) [39] is a communication protocol which allows different operating processes to communicate with each other and data transfer between processes. We used piping IPC [40] in our PMS, which works as a client/server system, using ports to which processes subscribe, and thus allowing complex data transfers.

3.3 Data Processing

The usage of generated XML data files, history files, and piping IPC in our PMS produces simple data and process communication on a higher level. Unified Modeling Language (UML) [41- 42] diagrams are used to illustrate the processes and their sequences within the performance monitoring system. **Fig. 3** shows the data flow processes of the PMS.

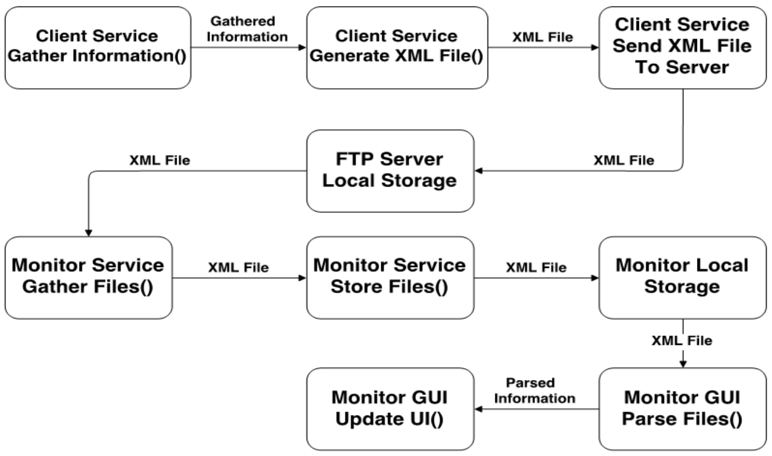

Client Service Gather Information()
Gathered Information
Client Service Generate XML File()
XML File
Client Service Send XML File To Server
XML File
FTP Server Local Storage
XML File
Monitor Service Gather Files()
XML File
Monitor Service Store Files()
XML File
Monitor Local Storage
XML File
Monitor GUI Parse Files()
Parsed Information
Monitor GUI Update UI()

Fig. 3. PMS Data Flow Processes

In this section, we propose four monitoring information algorithms to illustrate the data processes and their sequences within the performance monitoring system in cloud computing environment where set of virtual machines are created for several datacenters.

- **Performance Information Service Algorithm (PISA)**

  PISA algorithm collects the performance monitoring information and uploads it to the server. It checks the client service timer; if the service timer is off, then the monitoring data is generated in an xml file, encrypted and uploaded and stored in the server storage.

Performance Information Service Algorithm

```
Input: CPU percentage, RAM available (MB), Current OS processes, Machine name, Date
       and time
Output: Performance monitoring information xml data file
Begin
  IF Client Service Timer off THEN
      COLLECT Performance Monitoring Information
          GENERATE XML file
          ENCRYPT generated XML file
          PING server to check for availability
             IF server is available THEN
                   UPLOAD file
             END IF
             RESET service timer
  END IF
End
```

- **Machines Performance Object Service Algorithm (MPOSA)**

  MPOSA algorithm creates machines object list and displays it on client GUI. It opens the machine history file and compares the service last updated time with the received time of the requested service. Based on the previous result and according to the number of history points, machines object list is updated and displayed on the client GUI.

Machines Performance Object Service Algorithm

```
Input: Available XML file list on server
Output: Machines Performance Object list
Begin
  IF timer goes off or user requested an update service THEN
  PING server to check for availability
    IF server is available THEN
      REQUEST files from Server
         STORE files on Monitoring machine
         CREATE Machines Object List
         FOR each file saved
         READ xml file And CREATE Machine Object
         OPEN MACHINE HISTORY FILE
          IF Last Updated Time Stamp < Received Time Stamp THEN
      IF number of history points equals maximum number of history points allowed THEN
            DELETE oldest Node
           END IF
          INSERT received new machine object to the top of machines object list
         END IF
      ADD machine object to machines object List
      END FOR
  SEND Machines object list to GUI
  END IF
 END IF
End
```

- **Machines Data Grid Service Algorithm (MDGSA)**

MDGSA algorithm converts the available monitor service to data grid view row and updates the chart history points. The details of this algorithm are listed below.

Machines Data Grid Service Algorithm

Input: Machines Performance Object list
Output: Data Grid View Rows, Chart History Points
Begin

```
  PING monitor service to check availability
    IF monitor service is available THEN
         SEND request to machines object list
             RECEIVE response
             FOR each machine in response machines object list
                 CONVERT machine object to Data Grid View Row
                 ADD row to Data Grid View
             END FOR
        END IF
End
```

- **Populating Performance Information Charts Service Algorithm (PPICSA)**

PPICSA algorithm checks the updated history points in a machine history file and creates the performance information charts accordingly. The details of this algorithm are illustrated below.

Populating Performance Information Charts Service Algorithm

```
Input: Most recent history point in machine history file
Output: Populated history charts from the selected machines history file points
Begin
  IF a new machine is selected THEN
     CLEAR charts
        IF selected machine history file exists THEN
           READ xml file
           FOR each point in the history file
                 CREATE chart point for each machine performance attribute
                 ADD chart point to chart
           END FOR
        END IF
  END IF
End
```

- **Monitoring Time Cost Performance**

The time complexity in our approach is bounded by the time costs of collecting, converting and displaying performance metrics of the monitored machines. The time costs are described, defined, computed and improved as follows:

- The time costs of collecting, converting and displaying performance metrics from all monitored machines are defined and computed as follows:
- Let Tsearch represent the time cost to search for one performance information metric at one machine in order to display it on the monitoring screen.
- Let Tfetch represent the time cost to fetch one performance information metric at one machine.
- Let Tload represent the time cost to load one performance information metric at one machine from the cloud server.
- Let Texec represent the time cost to execute one performance information metric at one machine from the cloud server.
- Let Tcollect represent the time cost to collect the performance information of a certain metric at one machine. Then, Tcollect is computed as in equation 1:

$$Tcollect = Tserach + Tfetch + Tload + Texec \qquad (1)$$

- Let Tgraph represent the time cost to convert the collected performance information of one metric at one machine into a graph in the GUI.
- Let *n* represent the metric number in the monitored machine.
- Let Tmetric represent the time cost to display one metric performance information at one machine on the screen. Then, Tmetric is computed as in equation 2:

$$\text{Tmetric}(n) = \text{Tcollect}(n) + \text{Tgraph}(n) \qquad (2)$$

Consequently, the time cost to display the graphs of all performance metrics from all monitored machines on the screen is defined and simplified as follows:

- Let *n* represent the number of cloud system monitored machines.
- Let *m* represent the number of metrics in one machine.
- Let T_{MSR} represent the total time cost to perform the monitoring service request for all performance metrics at all machines. Then, T_{MSR} is computed as in equation 3:

$$T_{MSR} = \sum_{i=1}^{n} \sum_{j=1}^{m} (\mathrm{T}metric(i,j)) \qquad (3)$$

To this end, having one system that can provide all performance information metrics that the user requires for one machine on one screen will allow the user to monitor multi-metrics at all machines in much less time. Therefore, the time cost of searching for each metric at each machine is optimized in our proposed approach and done automatically for all metrics at all machines, thus improved the total time cost of monitoring service request. Accordingly, the improved total monitoring time cost for all performance metrics at all machines is defined as follows:

- Let Tsearch_all represent the total time cost to search for all performance metrics at all machines. Then, Tsearch_all is computed as in equation 4:

$$Tsearch\_all = \sum_{i=1}^{n} \sum_{j=1}^{m} (\mathrm{T}search(i,j)) \qquad (4)$$

- Let $T_{improvedMSR}$ represent the improved total monitoring time cost. Then, $T_{improvedMSR}$ is computed as in equation 5:

$$T_{improvedMSR} = T_{MSR} - \text{Tsearch\_all} \qquad (5)$$

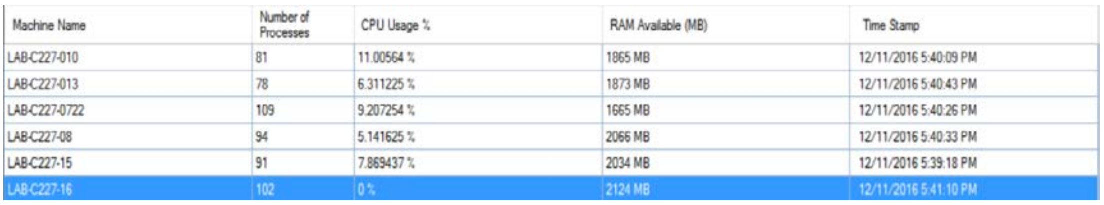
Therefore, the performance improvement in monitoring service request is defined according to the improved monitoring time cost that is computed as follows:

- Let PI represent the performance improvement achieved by our approach. Then, PI is expressed in equation 6:

$$PI = 1 - (T_{improvedMSR} / T_{MSR}) \qquad (6)$$

3.4 Performance Monitoring System Tool

The internal validity of our performance information services technique is tested on multiple machines in a cloud computing system. We developed a performance monitoring system tool that is able to test the PISA, MPOSA, MDGSA, PPICSA services on a set of performance history files obtained from different cloud machines. This tool is an open source, user-friendly graphical user interface that is divided into 3 main areas to evaluate cloud system performance. The proposed PMS tool is created following an approach that requires less user interaction while preserving as much visual feedback as possible. Moreover, it adds an important impact on extracting knowledge to the benefit of business decisions. **Fig. 4** shows the tool's GUI, which is divided into 3 different blocks; namely; control, information, and feedback.



Machine Name	Number of Processes	CPU Usage %	RAM Available (MB)	Time Stamp
LAB-C227-010	81	11.00564 %	1865 MB	12/11/2016 5:40:09 PM
LAB-C227-013	78	6.311225 %	1873 MB	12/11/2016 5:40:43 PM
LAB-C227-0722	109	9.207254 %	1665 MB	12/11/2016 5:40:26 PM
LAB-C227-08	94	5.141625 %	2066 MB	12/11/2016 5:40:33 PM
LAB-C227-15	91	7.869437 %	2034 MB	12/11/2016 5:39:18 PM
LAB-C227-16	102	0 %	2124 MB	12/11/2016 5:41:10 PM

Fig. 4. PMS GUI Blocks

The description of the GUI blocks that represent the performance metrics are detailed in the following sub-sections.

3.4.1 Control Block

The User Control Block consists of multiple user-interactive buttons that serve as the main process control component of the application, allowing the user to trigger system processes. This block consists of the following button controls encapsulated inside the GroupBox component:

- Start/Stop Service: Allows the user to control a service by starting or stopping it.
- Update Table: Triggers the information update processes, which will cause the program to execute all of its operations. This button is disabled if there is already an ongoing update process executing.
- Print: Starts the printing process of the form.
- Information: Produces a message box containing PMS project information.
- Exit: Closes the application.

3.4.2 Information Block

The information block consists of two components, named Control Machine and History Charts.

- Control Machine

This component holds a Data Grid View Control that shows the latest performance parameters parsed from the latest machine file retrieved from the server. The retrieved performance metrics are machine name, number of processes, CPU usage, available RAM, and date and time of the last performance check. **Fig. 5** presents the performance information generated from a control machine.

- History Charts

The History charts component includes four different metrics that are responsible for displaying the results of the performance information analysis of the control machine. The history charts metrics are defined as follows:

- Machine List: Holds the machine names for which the system has history files. Choosing one of these machines will result in updating the components information contained within its history file.

- Process Count: A chart that provides visual information about the number of processes running on the chosen machine. Here, the X-axis stands for the time frame in which the monitoring occurred and the Y-axis stands for the process count value.

- CPU Usage: The percent amount of CPU usage of the machine during the time interval available in its history file. The X-axis stands for the time frame in which the monitoring occurred and the Y-axis stands for the CPU usage value.

- RAM Available: The available RAM storage in mega-bytes on the chosen machine over the course of its recorded history file. The X-axis stands for the time frame in which the monitoring occurred and the Y-axis stands for the available RAM in mega-bytes (MB). **Fig. 5-a,b,c** shows all the PMS information components as they appear during run-time.

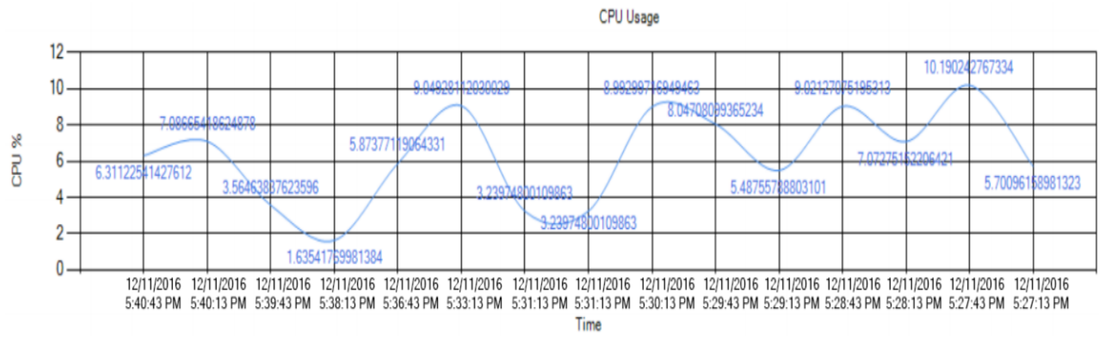


Fig. 5-a. CPU Information Metrics

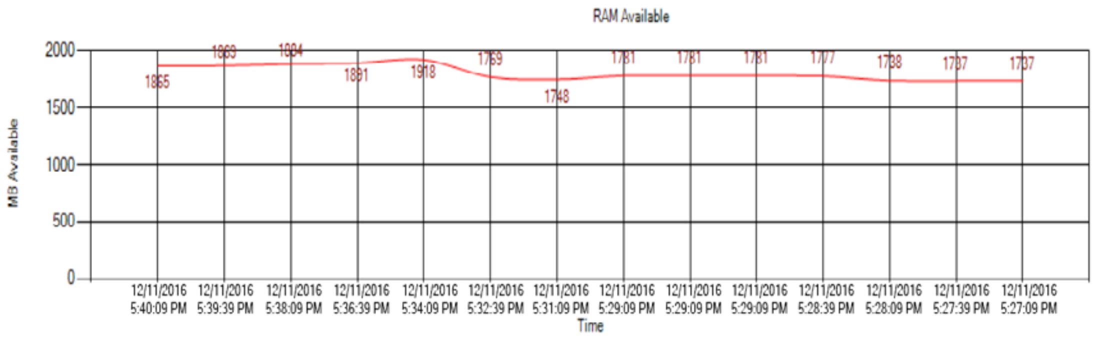


Fig. 5-b. RAM Information Metrics

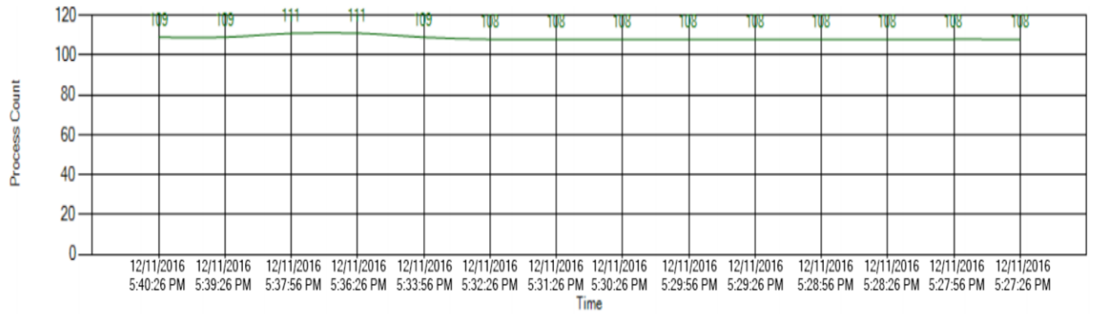


Fig. 5-c. Number of running processes on the active server

All components of the PMS information are updated immediately after an update operation finishes without regard to the operation triggers “User Clicked” or “Auto Update”.

3.4.3 Feedback Block

The feedback block of the performance analysis consists of two components: system information and a status bar.

A. System Information

The system information component serves as a feedback connection that holds the information from the PMS connections to the monitor service and the FTP server. This service has two components:

- The server connection status (connected, not connected).
- The service status (running, stopping, and stopped). Whereas the difference between the states (stopping and stopped) is defined such that stopping the service means finalizing its operations and de-allocating its resources, while the stopped state has already finished all of its operations and released all of its resources.

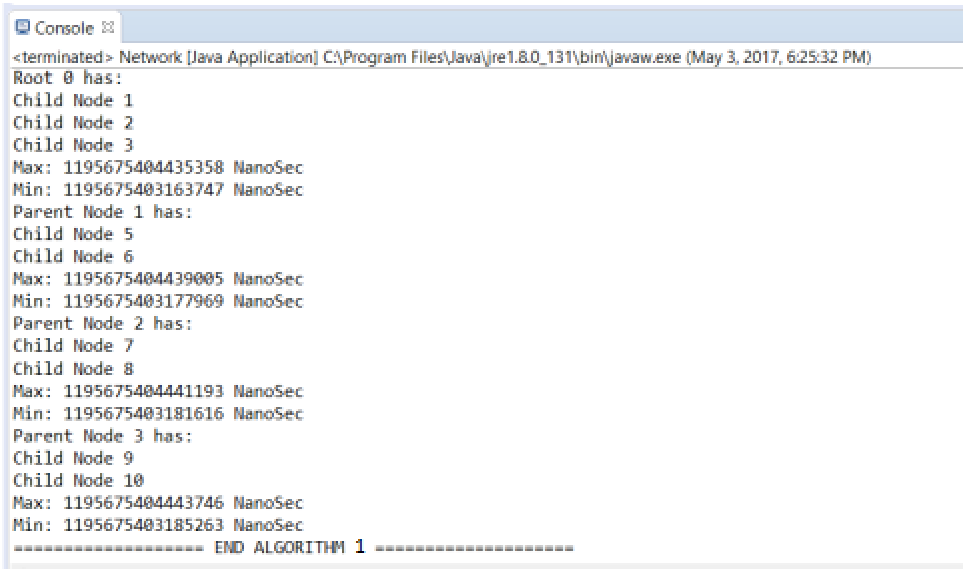
The only interactive control inside the Information block is the “Auto Update” check box that allows the system to automatically update the information about a semi-real time scheme it running or to stop it.

B. Status Bar

The bottom bar consists of dynamic controls: The information label and the progress bar that show the systems’ services status at run-time.

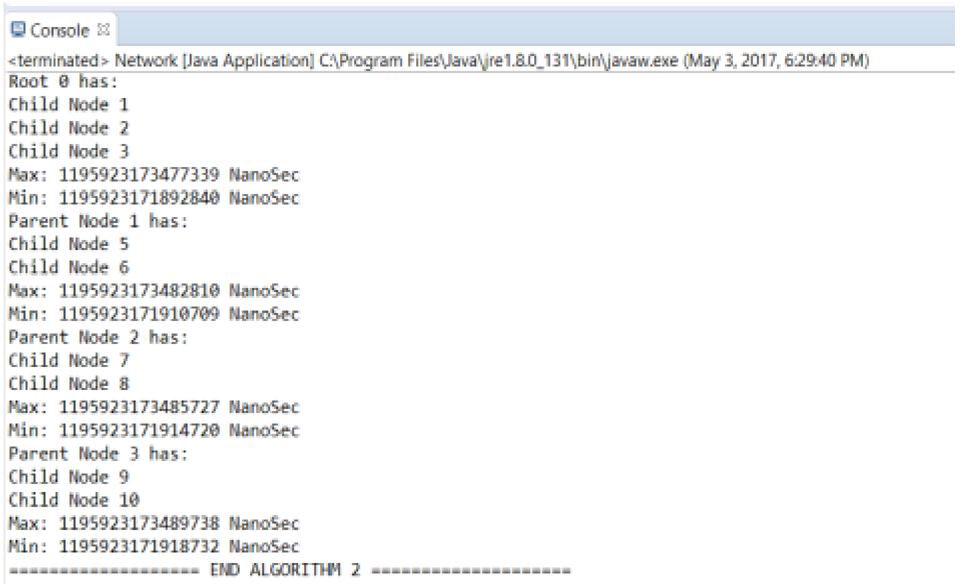
4. Experimental Results and Performance Evaluation

To validate our approach, we evaluated the performance techniques that are proposed by Fish et al. [27] and Dotti et al. [18], and compared their performance monitoring results with our technique. We run our monitoring tool under three algorithms that represents our proposed monitoring method and the methods introduced in [27] and [18] respectively. For simplicity, we assume the numbers of performance metrics are the same in all algorithms. Each algorithm is tested twice and the collected results are recorded as minimum and maximum time (time unit is: NanoSec). These collected results are required for extracting the processes on an active server. **Table 1** summarizes the average time considered for performance evaluations. **Fig. 6-8** depicts the experimental results of extracting monitoring time of three cloud system servers (parents 1, 2, 3) each controlled two child nodes; parent 1 controls child’s 5 and 6, parent 2 controls child’s 7 and 8, parent 3 controls child’s 9 and 10 respectively.



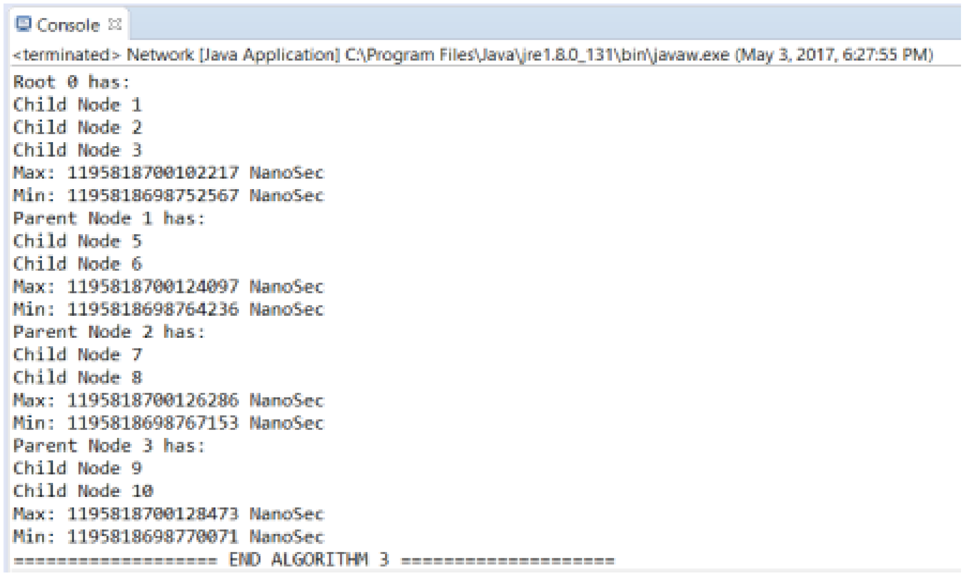
```
Console ⊠
<terminated> Network [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (May 3, 2017, 6:25:32 PM)
Root 0 has:
Child Node 1
Child Node 2
Child Node 3
Max: 1195675404435358 NanoSec
Min: 1195675403163747 NanoSec
Parent Node 1 has:
Child Node 5
Child Node 6
Max: 1195675404439005 NanoSec
Min: 1195675403177969 NanoSec
Parent Node 2 has:
Child Node 7
Child Node 8
Max: 1195675404441193 NanoSec
Min: 1195675403181616 NanoSec
Parent Node 3 has:
Child Node 9
Child Node 10
Max: 1195675404443746 NanoSec
Min: 1195675403185263 NanoSec
=================== END ALGORITHM 1 ====================
```

Fig. 6. Performance monitoring time required for our proposed method



```
<terminated> Network [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (May 3, 2017, 6:29:40 PM)
Root 0 has:
Child Node 1
Child Node 2
Child Node 3
Max: 1195923173477339 NanoSec
Min: 1195923171892840 NanoSec
Parent Node 1 has:
Child Node 5
Child Node 6
Max: 1195923173482810 NanoSec
Min: 1195923171910709 NanoSec
Parent Node 2 has:
Child Node 7
Child Node 8
Max: 1195923173485727 NanoSec
Min: 1195923171914720 NanoSec
Parent Node 3 has:
Child Node 9
Child Node 10
Max: 1195923173489738 NanoSec
Min: 1195923171918732 NanoSec
=================== END ALGORITHM 2 ====================
```

Fig. 7. Performance monitoring time required for Fish et al. [27] method



```
<terminated> Network [Java Application] C:\Program Files\Java\jre1.8.0_131\bin\javaw.exe (May 3, 2017, 6:27:55 PM)
Root 0 has:
Child Node 1
Child Node 2
Child Node 3
Max: 1195818700102217 NanoSec
Min: 1195818698752567 NanoSec
Parent Node 1 has:
Child Node 5
Child Node 6
Max: 1195818700124097 NanoSec
Min: 1195818698764236 NanoSec
Parent Node 2 has:
Child Node 7
Child Node 8
Max: 1195818700126286 NanoSec
Min: 1195818698767153 NanoSec
Parent Node 3 has:
Child Node 9
Child Node 10
Max: 1195818700128473 NanoSec
Min: 1195818698770071 NanoSec
=================== END ALGORITHM 3 ====================
```

Fig. 8. Performance monitoring time required for Dotti et al. [18] method

Table 1 shows the server performance monitoring average time required for the methods in comparison.

Table 1. Performance Monitoring Information Average Extraction Time

Server Number	**Our Proposed Method (nanosec)**	**Fish et al [27] (nanosec)**	**Dotti et al [18] (nanosec)**
Server 1	1136	6238	7211
Server 2	1135	7229	8356
Server 3	1134	8708	10071

The average extraction time is computed based on the number of processes that are extracted from the cloud servers and its machines (nodes). The proposed techniques in [27] and [18] consider a single active server processes under monitoring at a time unit (nanosecond), while in our technique we consider multivariate processes under monitoring on the active cloud server and its client machines. Therefore, much more processes are extracted and monitored by our technique in shorter time. The performance improvement in the previous experiment is computed according to equation 6 above. For example, the performance improvement of our approach over Dotti et al. [18] approach in server 1 is evaluated as follows:

$$PI_{(Server1)} = 1\text{- } (1136 / 7211) = 84.25\%$$

The performance improvements of our approach over both Fish et al. [27] and Dotti et al. [18] approaches are summarized in Table 2.

Table 2. Performance Improvement Comparison

Server Number	**Improvement % over Fish et. al. [27]**	**Improvement % over Dotti et. al. [18]**
Server 1	81.79%	84.25%
Server 2	84.30%	86.42%
Server 3	86.98%	88.74%

Based on the results generated by our approach, an enhanced performance monitoring system can be achieved while saving time, effort and power.

Fig. 9 shows the comparison of our proposed method with the methods introduced in [27] and [18] in terms of average time (nanosec) required for generating performance monitoring information.

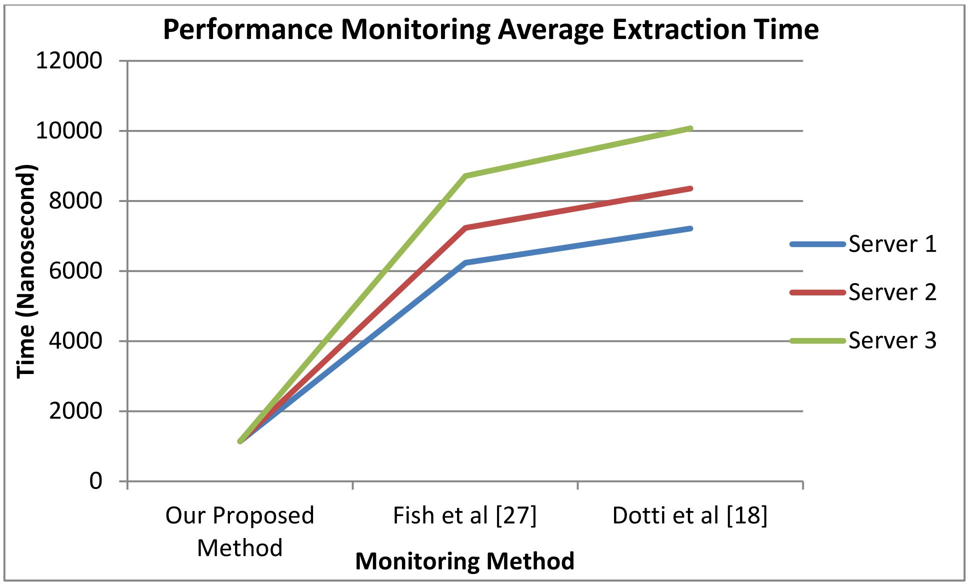


Fig. 9. Comparison of the Performance Monitoring Methods

This figure shows that more time is needed to extract performance information by approaches in [27] and [18] due to the high complexity of extracting performance metrics. Therefore, it is clear that our approach significantly outperforms its counterparts.

5. Conclusion

The data synchronization in a semi-real time for a control system in parallel and concurrent computational environment can be achieved. Integrating the various components of the performance monitoring system will empower the control systems of the real world applications. Such control systems are required for creating functional performance monitoring systems.
The benchmarks and the operating system task manager have high usage and are rather populated within our proposed performance monitoring system. The benchmark data is extracted from the operating system itself and is processed and given to the user within a simplified GUI, though within such a system, the controlling factors of processes or machine performance are not included.
There is potential to provide a powerful remote performance tool that can allow users to monitor and control cloud system performance easily. Most users are already familiar with task managers supported by different operating systems.

References

[1] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, and V. Bhatnagar, "An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art," *Computing*, 97(4), pp.357-377. 2015. Article (CrossRef Link)

[2] M. Riera-Guasp, J. A. Antonino-Daviu, and G. A. Capolino,"Advances in electrical machine, power electronic, and drive condition monitoring and fault detection: state of the art," *IEEE Transactions on Industrial Electronics*, 62(3), pp.1746-1759. 2015. Article (CrossRef Link)

[3] Z. Gao, C. Cecati, and S. X. Ding,"A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches," *IEEE Transactions on Industrial Electronics*, 62(6), pp.3757-3767. 2015. Article (CrossRef Link)

[4] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, 57(9), pp.2093-2115. 2013. Article (CrossRef Link)

[5] W. W. Eckerson, "Performance dashboards: measuring, monitoring, and managing your business," *John Wiley & Sons*. 2010. Article (CrossRef Link)

[6] X. Sales, and J. Carenys, "Case study on performance management: A comprehensive approach," *British Journal of Economics, Management & Trade*. Vol 3 no.2 pp 73-88. 2013. Article (CrossRef Link)

[7] Jain RK. "Art of Computer Systems Performance Analysis: Techniques for Experimental Design Measurements, Simulation and Modeling," *Wiley Computer Publishing, John Wiley & Sons*, Inc. 1992. Article (CrossRef Link)

[8] Frédéric Desprez, Eddy Caron, Luis Rodero-Merino, Adrian Muresan, "Auto-scaling, load balancing and monitoring in commercial and open-source clouds," *Cloud Computing: Methodology, System and Applications, CRC Press*, 2012. Article (CrossRef Link)

[9] J. Spring, "Monitoring cloud computing by layer, Part 1," *IEEE Security & Privacy*, 9 (2), pp 66–68. 2012. Article (CrossRef Link)

[10] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, A. Pescapè, "Broadband internet performance: a view from the gateway," *ACM SIGCOMM 2011 Proceedings*, 2011. Article (CrossRef Link)

[11] Aceto, Giuseppe, Alessio Botta, Walter De Donato, and Antonio Pescapè. "Cloud monitoring: A survey," *Computer Networks* 57, no. 9 . pp 2093-2115. 2013. Article (CrossRef Link)

[12] Hwang, K., Fox, G. and Dongarra, J, "Distributed and Cloud Computing," *Morgan Kaufmann Publisher*, 2012. Article (CrossRef Link)

[13] Desborough, Lane, and Randy Miller. "Increasing customer value of industrial control performance monitoring-Honeywell's experience," in *Proc. of AIChE symposium series. No. 326. New York; American Institute of Chemical Engineers*. 2002. Article (CrossRef Link)

[14] Rizwan Mian, Patrick Martin, Jose Luis Vazquez-Poletti, "Provisioning data analytic workloads in a cloud," *Future Generation Computer Systems*, 2013. Article (CrossRef Link)

[15] Hwang, Kai, et al. "Cloud performance modeling with benchmark evaluation of elastic scaling strategies," *IEEE Transactions on Parallel and Distributed Systems*, 27.1, pp.130-143. 2016. Article (CrossRef Link)

[16] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D.H.J. Epema, "A performance analysis of EC2 cloud computing services for scientific computing," *Cloud Computing, Springer*, pp. 115–131. 2010. Article (CrossRef Link)

[17] Y. Mei, L. Liu, X. Pu, S. Sivathanu, "Performance measurements and analysis of network I/O applications in virtualized cloud," in *Proc. of IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pp. 59–66. 2010. Article (CrossRef Link)

[18] A. Dotti, V. D. Elvira, G. Folger, K. Genser, S. Y. Jun, J. B. Kowalkowski, and M. Paterno, "Geant4 Computing Performance Benchmarking and Monitoring," *Journal of Physics: Conference Series (Vol. 664, No. 6, p. 062021). IOP Publishing*. 2015. Article (CrossRef Link)

[19] L. Shannon, E. Matthews, N. Doyle, and A. Fedorova, "Performance monitoring for multicore embedded computing systems on FPGAs," *arXiv preprint arXiv*:1508.07126. 2015. Article (CrossRef Link)

[20] C. D. Jones, L. Contreras, P. Gartung, D. Hufnagel, and L. Sexton-Kennedy,"Using the CMS threaded framework in a production environment," in *Proc. of Journal of Physics: Conference Series (Vol. 664, No. 7, p. 072026). IOP Publishing*. 2015. Article (CrossRef Link)

[21] A. Lister, "Fundamentals of operating systems," in *Springer Science & Business Media*, 2013. Article (CrossRef Link)

[22] A. I. Jehangiri, R. Yahyapour, P. Wieder, E. Yaqub, and K. Lu, "Diagnosing cloud performance anomalies using large time series dataset analysis," in *Proc. of IEEE 7th International Conference on Cloud Computing CLOUD* (pp. 930-933). June, 2014. Article (CrossRef Link)

[23] A. S. Radhamani and E. Baburaj, "Network Traffic Monitoring and Control for Multi core processors in cloud computing applications," *International Journal of Computer Information Systems and Industrial Management Applications*, 5 pp. 557-563. 2013. Article (CrossRef Link)

[24] V. Singh and A. Seth, "Approaches to Data Parallel Programming," *International Research Journal of Engineering and Technology (IRJET)*, Vol: 03 Issue: 05. 2016. Article (CrossRef Link)

[25] D. Licari, A. Baiardi, M. Biczysko, F. Egidi, C. Latouche, and V. Barone,"Implementation of a graphical user interface for the virtual multi-frequency spectrometer: The VMS-Draw tool," *Journal of computational chemistry*, 36(5), pp.321-334. 2015. Article (CrossRef Link)

[26] A. Blumenthal, M. Luedde, T. Manzke, B. Mielenhausen, and C. E. Swanepoel, "Measuring software system performance using benchmarks,". *U.S. Patent 7,546,598*. 2009. Article (CrossRef Link)

[27] J. Fish, D. R. Moulton, and K. Gray,"Graphical user interface with on board and off-board resources," *Bosch Automotive Service Solutions Inc*. U.S. Patent 9,299,197. 2016. Article (CrossRef Link)

[28] Last Accessed on July 7th, 2017. Article (CrossRef Link).

[29] W. R. Williams, X. Meng, B. Welton, and B. P. Miller,"Dyninst and MRNet: Foundational Infrastructure for Parallel Tools," *Tools for High Performance Computing, Springer International Publishing*. pp. 1-16. 2015. Article (CrossRef Link)

[30] K. Mehlhorn, "Data structures and algorithms 1: Sorting and searching," *Springer Science & Business Media*, Vol. 1, 2013. Article (CrossRef Link)

[31] 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*, 74(10), pp.2918-2933. 2014. Article (CrossRef Link)

[32] T. Fuand M. Gaurav,"Apparatus, systems and methods for deployment of interactive desktop applications on distributed infrastructures," *U.S. Patent Application*, No. 13/759,514. 2013. Article (CrossRef Link)

[33] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository," *Information and Software Technology*, 55(10), pp.1679-1694. 2013. Article (CrossRef Link)

[34] R. Mohammadi, S. Y. Nabavi, and S. M. Emam, "Analysis of FTP and Web Server Performance in Open Source Server Virtualization," *International Journal of Computer Science Issues (IJCSI)*, vol 13 no 5 2016. Article (CrossRef Link)

[35] W. Jackson,"Introduction to XML: Defining an Android App, Its Design, and Constants," *Android Apps for Absolute Beginners. A press*, Pp 101-130. 2014. Article (CrossRef Link)

[36] D. Nolan and D. T. Lang,"An Introduction to XML. In XML and Web Technologies for Data Sciences with R," *Springer New York*. (pp. 19-52). 2014. Article (CrossRef Link)

[37] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. Khan, and V.Bhatnagar," An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art," *Computing*, 97(4), 357-377. 2015. Article (CrossRef Link)

[38] O. Edelstein, E. Farchi, E. Goldin, Y. Nir, G. Ratsaby, and S. Ur,"Framework for testing multi-threaded Java programs," *Concurrency and Computation: Practice and Experience*, 15(3-5), pp.485-499. 2003. Article (CrossRef Link)

[39] P. D. Bain,"Inter-process communication in a multi-tenant environment," *International Business Machines Corporation*. U.S. Patent Application 14/842,926. 2015. Article (CrossRef Link)

[40] C. K. Hsieh, H. Falaki, N. Ramanathan, H. Tangmunarunkit, and D. Estrin,"Performance evaluation of android IPC for continuous sensing applications," *ACM SIGMOBILE Mobile Computing and Communications Review*, 16(4), pp.6-7. 2013. Article (CrossRef Link)

[41] S. Friedenthal, A. Moore and R.Steiner, "A practical guide to SysML: the systems modeling language," *Morgan Kaufmann*, 2014. Article (CrossRef Link)

[42] C. Larman," Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development," in *Pearson Education*. 2012. Article (CrossRef Link)

Ismail Hababeh received the bachelor's degree in computer science from University of Jordan, Amman-Jordan, the master's degree in computer science from Western Michigan University, Michigan – USA, and the PhD degree in computer science from Leeds Metropolitan University, Leeds-UK. He is currently with German Jordanian University at the Faculty of Electrical Engineering and Information Technology. His research interests span the areas of cloud computing, big data security, wireless communication networks, and systems performance.

Anton William Thabain received the BSc degree in Computer Science from the German Jordanian University, Amman - Jordan, in 2016. Currently, he pursuing the MSc and PhD from the Technical University of Cologne. His research interest includes the development of parallelized computer systems and computer performance algorithms in addition to Game AI development.

Sahel Alouneh is an Associate Professor of computer engineering and the Dean of Faculty of Electrical Engineering and Information Technology at the German Jordanian University. His research interests include computer networks, big data security, cloud computing, software security, MPLS security and recovery, Wireless networking security, Software testing, computer design and architecture.