Resource Monitoring Algorithms Evaluation For Cloud Environment

Mustafa M. Al-Sayed *mostafamcs@gmail.com*
Faculty of Computers and Information
Minia University
Minia, Egypt

Shrerif M. Khattab *s.khattab@fci-cu.edu.eg*
Faculty of Computers and Information
Cairo University
Cairo, Egypt

Fatma A. Omara *f.omara@fci-cu.edu.eg*
Faculty of Computers and Information
Cairo University
Cairo, Egypt

Abstract

Cloud computing is a type of distributed computing allowing to share many resources such as CPU, memory, storage ...etc. The status of these resources changes from time to time due to the dynamic adaptive ability of the cloud computing characteristics. Hence, the powerful and scalable monitoring algorithm is needed to monitor the status of these resources throughout the time. There are many models have been proposed for monitoring the distributed systems resources; the push-based, the pull-based, and the push/pull model. Most of the common monitoring systems are based on these models (e.g., Ganglia which based on push model and Nagios, which based on pull model). According to the work in this paper, a comparative study has been done to implement and evaluate these three models on the cloud environment. The implementation results showed that the push-based model outperforms the other two models due to its high scalability, stability, and efficiency.

Keywords: Cloud Computing, Resource Monitoring, Virtualization, Scalability.

1. INTRODUCTION

Recently, there is a rapid growth in the distributed computing system technologies. The main feature of these technologies is that they can easily provide a large amount of computing power and resources sharing [10]. Types of these systems are cluster, Grid and Cloud computing, which are allowed to access large amounts of computing power in a fully virtualized manner, through pool of resources and provide a single system view [7]. The main importance of these technologies is to deliver the Information Technology (IT) resources as utility [7]. So, the Cloud computing is a kind of new technology, and its main concept is to provide the services to the users through “pay-as-you-go” manner [8]. Therefore, the accurate monitoring of the resources which are consumed by the users is really importance issue because its effect on the system performance.

On the other hand, the Cloud computing is considered more complicated due to its heterogeneous and dynamic characteristics. Hence, the vital part of the Cloud computing system is to monitor the characteristics and the existence of the resources, services, computations, and other entities [1]. Therefore, the monitoring resources process is concerned with the collection of these resources information, which can be useful to manage many problems, such as job

scheduling, load balancing, event predicting, fault detecting, and fault recovery in the cloud computing, because the incorrect and unreal information would affect the performance of these problems [2]. So, a monitoring scheme that can collect and analyze the dynamic information for ensuring the stable participation of resources is needed. This monitoring scheme needs to be changed dynamically in real time to monitor the correct state of the information and to reflect the characteristics of the cloud resources [2]. When the time interval of the monitoring model is very short, the overhead of collecting information would be increased. The monitoring model, however, cannot keep a correct state of information in dynamic environments if the interval is very long [2].

There are many models have been proposed for monitoring the distributed systems resources; the push-based, the pull-based, and the push/pull model. Most of the common monitoring systems are based on these models (e.g., Ganglia which based on push model and Nagios, which based on pull model). Unfortunately, there is no any comparative study has been done to advice the most suitable to monitor resources on the Cloud environment among these three models. According to the work in this paper, a comparative study has been done to implement and evaluate these three models on the cloud environment. Based on the results of this study, the most suitable model for monitoring the resources on the Cloud environments will be chosen as the candidate model. In the future, this candidate model might be contributed to develop a new consistent and efficient model.

The paper organization is; the background and the related works are presented in sections 2 and 3 respectively. The comparative study among three monitoring resources algorithms based on the Push, the Pull, and the P&P models is presented in section 4. In section 5, The Performance Evaluation is presented. The conclusions and future work are presented in section 6.

2. BACKGROUNDS

There are some monitoring systems that are used in the large scale systems and Cloud computing environments. These monitoring systems can be classified into systems which based on pushing data from its point of collection and those which based on pulling data. Also, these systems can be classified into centralized and decentralized. The most common monitoring systems are Ganglia, and Nagios. Nagios represents a centralized system which pulls data from each monitored component. But, Ganglia is considered more decentralized and pushes data from its point of collection [13].

On the other hand, Ganglia is an open source distributed monitoring system, which has a simple hierarchal architecture, and depends on a multicast-based announce protocol to monitor the states of hosts [14] [15]. Also, Ganglia uses technologies such as XML, RRDTool (Round Robin Database), and XDR for data representation, data storage, and data transport, respectively.

Because of Ganglia characteristics, it satisfies high performance, robustness, good scalability, and low per-node overheads [13]. Ganglia is a popular grid monitoring system, which runs over 500 clusters around the world, and has been ported to nine different operating systems, and six CPU architectures [15]. Therefore, Ganglia monitoring system will be used as a criterion to differentiate among three monitoring resources models (Push, Pull, and hybrid).

Many types of Grid monitoring systems adopted the principle of Grid Monitoring Architecture (GMA) [16]. This principle contains three main roles; Producers (Working nodes), Consumers (Master node), and Registers. The Producers generate status information of monitored resources in its domain. The consumers use this information to solve many problems as mentioned above. One main purpose of the registers is to facilitate Consumers and Producers to find each other.

There are two basic models for the Consumers and the Producers interaction; the Pull-based model and the Push-based model [2]. In the Pull model, the Consumer pulls information from the Producers to inquire status by sending a message to a producer in order to request resources information. In the Push model, however, the Producers push the new resources' status to the

consumer when any updates are occurred at a Producer, under some trigger conditions and according to the decided policy by the cloud administrator based on the Service Level Agreement (SLA), which is an agreement between the provider and the client. The Cloud monitoring entities can also be modeled as Producers, Consumers, and Directory (locates producers and consumers) [3].

In the Pull model, the resource information is required when it is needed [2]. The more nodes or resources in cloud computing environment or any distributed environment, the greater overhead on master node occurs due to the increasing number of requests [4]. If the pulling is implemented depending on pull interval and this interval is small, this will cause high network consumption. However, if the pull interval is large, this causes the loss of important updates during this large interval. Hence, if the pull model has a high efficiency property, its consistency would be low [3]. So, a less transmission costs and a better efficiency would be obtained when the pull interval is proper.

In the push model, resources status information is pushed by the producer to the consumer under some trigger conditions. The Producer pushes the information of the current status of the monitored resource, if this information is different than the previous information by a specific value which called threshold [3]. The small threshold will cause a high transmission costs and the transmission of unnecessary updates. The large threshold may cause the loss of an importing updates. As the case in the threshold, the push interval time of the monitoring plays an important role in the quality and the communication overhead of the push model. A short interval time for pushing will increase the communication overhead of collecting information. A long interval time may result in the loss important updates. Hence, if the push model has a high consistency, its efficiency would be low [3]. So, a less transmission costs, a less network consumption, and a better efficiency would be obtained from the proper push threshold and push interval time. This keeps the consistency between the producer and the consumer.

A hybrid model, called Push-Pull (P&P), has been proposed [3]. The main features of the P&P model are the combination of the advantages of Push-based and Pull-based models, where the pushing deployed in the producer and the pulling deployed in the consumer and the two procedures are run simultaneously [3]. The P&P would satisfy a better performance as a result of its ability to switch between Push and Pull intelligently. The P&P model has been deployed to monitor the resources in the Cloud computing environment. This is achieved by deploying Push model on each working node and a Pull model on the master node. The two models run simultaneously [3].

Generally, the Pull model depends basically on the interval time between pulls. This interval can be static or dynamic by using one of forecasting methods to predicate the next pull interval. But the push model depends either on push interval time between pushes, trigger conditions, or both. Both models try to maximize the information precision with minimal network bandwidth consumption. This would be achieved by choosing a proper time interval, and a proper threshold. Although the Push, the Pull, and the P&P models are existed and implemented in the Grid and Cluster computing systems, there is no any attempt has been introduced to compare the performance of these models in the Cloud environment. In this paper, a comparative study has been done to evaluate the pull, push, and hybrid models to stand on the main deference points between these monitoring models.

3. RELATED WORKS

Foster, and et al [18] have concluded that Clouds and Grids share a lot Commonality in their vision, e.g., architecture and technology, so most of the resource monitoring mechanisms and platforms for Grid computing [19-23] have been customized for Cloud systems.

In an attempt to improve the Pull model and minimizing its updates, R. Sundaresan and et al [6] have proposed a set of pull-based algorithms using historical time series of information returned

from the different resources to estimate the time of the next information update. One of the crucial components of these algorithms is the estimator. They show that the different estimators, such as simple moving average and Exponential Weighted Moving Average (EWMA) perform best in different situations depending on a specific pattern of resource usage updates. This can be adaptively tuned to maximize freshness.

In an attempt to minimize unnecessary and useless updating massages, and maximize the consistency between the producer and consumer. Wu-Chun Chung and Ruay-Shiung Chang [5] have proposed GRIR (Grid Resource Information Retrieval), which is considered a new algorithm for resource monitoring in grid computing to improve Push model. They examined a set of data delivery protocols for resource monitoring in the push-based model, such as the OSM (Offset-Sensitive Mechanism) protocol, the TSM (Time-Sensitive Mechanism) protocol, and the hybrid ACTC (Announcing with Change and Time Consideration) protocol. This hybrid protocol is based on a dynamically adjusted update time interval and the consideration for early update when the change is larger than a dynamic threshold.

Han Fang-Fang, Peng Jun-Jie, Zhang Wu, and et al [8] have proposed a periodically and Event-driven Push (PEP) monitoring algorithm. This algorithm combines the advantages of the push and event-driven mechanism and simplifies the communication between the consumer and the producer without missing the important updating which would be happened during the push interval. This algorithm does not take a lot of computing resources and provide more adequate information.

Motivated by the complementary properties of Push model and Pull model, H. Huang and L. Wang [3] have presented a hybrid resource monitoring algorithm for Cloud computing called "P&P". This algorithm is considered a combination of Push and Pull Models for resource monitoring in the Cloud Computing Environment. The P&P model inherits the advantages of Push and Pull models. It can intelligently switch between Push and Pull models and adjust the number of updating according to the requirements of the users. This algorithm reduces the updating rate and maintains various levels of coherence in accordance with the users' requirements.

The resources monitoring model should be adaptive as much as possible in order to avoid a negative impact of monitoring activities, and then provides an accurate and efficient resource monitoring system. There are several studies have faced such issues by tuning the amount of monitored resources and the monitoring frequency [24-28]. For instance, Park and et al. [24] have proposed a monitoring algorithm based on Markov Chains to analyze and predict resource states, in order to adaptively set a suitable time interval to push monitoring information. An adaptive measurement algorithm has been proposed, to monitor resource usage patterns, where the past measurement history are used to update the measurement frequency dynamically [17]. This algorithm has relatively achieved accurate patterns and reduces monitoring overhead considerably.

4. THE COMPARATIVE STUDY IMPLEMENTATION

According to the work in this paper, a comparative study has been done among three monitoring resources algorithms, which are based on push model, pull model, and hybrid model. These three algorithms are Announce with Change and Time Consideration (ACTC) algorithm which based on the push model [5], the Adaptive Polling of Grid Resource Monitors using a Slacker Coherence algorithm which based on the pull model [6], and the Pull-Push (P&P) algorithm which based on the hybrid of push and pull models [3].

The evaluation parameters which are considered in this comparative study are [1] [4]:

- ***Efficiency***; communication overhead measured as the number of updates, which are exchanged among system components, per time unit. It must be as small as possible.
- ***Quality***; a small delay when detecting a threshold crossing.

- ***Scalability***; Quality degrades and communication overhead increases linearly with increasing the system size.
- ***Low intrusiveness***.

The implementation of the ACTC, the Adaptive Polling, and the P&P algorithms will be discussed in details in the following sections.

4.1 The ACTC Monitoring Algorithm

The ACTC algorithm is based on push model and combined the advantages of the conditional Announce-Absolute-Change (AAC) scheme [5] and Announce-Dynamic-Interval (ADI) scheme. Along with the updating information based on the dynamic time interval (DTI), if the status information change of one resource becomes larger than a dynamic threshold (d_threshold), the change is updated immediately even if the timer has not expired. In Announce-Time-Interval (ADI) scheme, the time interval of sending updating message to master node is adjusted based on the moving average of time intervals between status information changes. Let T(0), T(1), …, T(i) be the time when status of monitored resources changes. The Dynamic Time Interval (DTI) is calculated as the following equation [5]:

$$DTI = \frac{1}{NC} \times \sum_{i=1}^{NC} (T(i) - T(i-1)) \quad \text{..... (1)}$$

Where, NC is the number of changes, which is initialized by 0.

In the conditional Announce-Absolute-Change (AAC) scheme, the status information change is announced to the master node only when the change between the current and previous value is greater than a dynamic threshold (d_threshold), which is initialized to zero and is dynamically calculated as the following equation [5]:

$$\text{d\_threshold} = \frac{1}{NA} \times \sum_{i=1}^{NA} AVCi \quad \text{..... (2)}$$

Where, NA is the number of announcements (updates) and AVCi = |Ai – Ai-1|, where A0, A1… Aj represents the successive updates to the master node. Let d_threshold = 0, and A0=C0, where C0, C1… Ci are the values of the monitored resource when status change occurs.

The ACTC algorithm is composed of the combination between the AAC, and the ADI algorithms to become one algorithm. The pseudo code of this algorithm is depicted in Figure (1).

```
1.  ACTC_model(){
2.  define list of successful_updates //to be used in calculating the d_threshold
3.  while(true)
4.      //--push based on dynamic threshold------
5.      if (change_degree > d_threshold )
6.         Update  successful_updates list
7.          d_threshold ← get_Dynamic_Threshold(successful_updates);
8.          DTI ← get_Dynamic_Interval(successful_updates);
9.          timer ← DTI;
10.         push current update to master node
11.     //--push based on Dynamic time interval--
12.     else if (is timer terminated?)
13.         if (change_degree > MINThreshold  )
14.             Update  successful_updates list
15.             d_threshold ← get_Dynamic_Threshold(successful_updates);
16.             DTI ← get_Dynamic_Interval(successful_updates);
17.             timer ← DTI;
18.             push current update to master node
19.  //end loop
20. }// end ACTC algorithm
```

FIGURE 1: The pseudo code of the ACTC Algorithm which is based on the push monitoring model and is composed of AAC and ADI algorithms.

According to The ACTC pseudo code, the ACTC algorithm runs as a probe at each Producer. This algorithm monitors resources according to changes of status of the monitored resources comparing with the previous statuses. The Producer will push an update to the Consumer, when the timer terminated (line 12). This timer has been calculated, based on equation (1), from the previous intervals of successive updates (lines 8, 16). Also, The Producer will push update, when status information change became greater than d_threshold value (line 5), which is calculated according to equation (2) as in lines (7, 15).

4.2 The Adaptive Polling Algorithm

The Adaptive polling algorithm, which has been explored by R. Sundaresan and et al [[4], is based on the pull model. In this algorithm, the consumer polls the probe of Producer ***i*** after a time interval $\mathbf{t_i}$. When the value of the received update from Producer ***i*** (status information) is greater than the previous one with a significant amount, the master node will decrease the polling interval to become $\mathbf{t_i*(1 - d)}$, where **d** is the damping factor, beside a minimum limit to avoid extreme situation. But, if there isn't any significant change, the consumer will increase this interval to become $\mathbf{t_i*(1 + d)}$, beside a maximum limit to the polling interval. The pseudo code of this algorithm is depicted in Figure (2).

```
1.   adaptivePolling() {
2.   while (true)
3.     if (is Poll_interval  terminated ?)
4.       current_value   ← Send poll message to Producer i
5.       if ( |current_value  - previous_value | > change_degree)
6.          Poll_interval ←Poll_interval * (1 - d)
7.       else if ( |current_value - previous_value | <   change_degree)
8.          Poll_interval ←Poll_interval * (1 + d)
9.    // end while loop
10. } // end adaptive Polling algorithm
```

FIGURE 2: The pseudo code of the Adaptive Polling Algorithm which is based on the pull monitoring model, depending on the damping factor **d**.

At the termination of the poll interval time of Producer ***i*** (line 3); the Consumer will send a poll message to this Producer (line 4). **change_degree** is used to decide if the change in the status is significant or not (lines 5, 7) where **change_degree** is set based on the user requirements.

4.3 The Push-Pull Model (P&P) Algorithm

To inherit the complementary properties and the advantages of the Pull and Push models, H. Huang and L. Wang [3] have proposed the P&P algorithm, which is considered an amalgamation of Pull and Push models. The P&P model switches between the two models and adjust the number of updating according to the users requirements. The switching between the Pull and the Push models is based on the comparing of the change degree in status information of the monitored resources with the users' requirements as defined in the following equation:

$$change\_degree = \frac{|P_i - C_i|}{MAX - MIN} \leq UTD \quad \ldots\ldots\ldots (3)$$

Where, **change_degree** describes the change between the current status of Producer ***i***, which is defined as $\mathbf{P_i}$ and the last Status information that the consumer received, which is defined as $\mathbf{C_i}$. MAX and MIN represents the maximal and minimal possible value of the status.

This hybrid model depends basically on the user requirements, which is defined as UTD[1]. The P&P algorithm has three possible cases based on the value of UTD [3]:

1- Dominating push-based when UTD is relatively small,

2- Dominating pull-based when UTD is relatively large, and

[1] The requirements of users are expressed by the concept of User Tolerant Degree (UTD)

3- None dominating when UTD is relatively moderate.

The P&P algorithm consists of P&P-Push algorithm and P&P-Pull algorithm. The pseudo code of the P&P-Push and P&P-Pull algorithms are depicted in Figures (3, 4) respectively. The P&P_Push algorithm runs at the Producer and the P&P-Pull algorithm runs at the Consumer concurrently in a mutually exclusive manner[2]. The two algorithms will be switched between Push and Pull according to UTD value and status information changes of monitored resources.

When UTD approaches 1, the Pull operation dominates. Therefore, when UTD equals to 0, all Pull operations are forbidden, and the P&P algorithm becomes pure Push model. Similarly, When UTD approaches 0, the Push operation dominates. Therefore, when UTD equals to 1, all Push operations are forbidden, and P&P model become pure Pull model [3]. Hence when UTD is relatively moderate, none of Push and Pull dominates. This situation is just in the middle of the above two cases, and both Push and Pull methods act frequently.

```
1.   P&P_Push(){
2.   while(true)
3.      set Pull operation identifier isPulled ← false
4.      waiting for the termination of the Push_interval
5.      if isPulled equals to true during Push_interval
6.        push current update to master node
7.      else if( | current_value  - previous_value | / (MAX - MIN) ≥ UTD )
8.        isPushed ← false
9.        push current update to master node
10.  //end while
11.  } //--end P&P  Push algorithm----
```

FIGURE 3: The pseudo code of P&P_Push Algorithm which runs at the Producer part.

```
1.   P&P_Pull() {
2.   while (true)
3.     foreach producer in master_domain
4.        Set Push operation identifier isPushed  ← false
5.        waiting for Pull_interval
6.        If isPushed equals to true during Pull_interval
7.           update current status information (current_value)
8.        else
9.           isPulled ← true
11.          current_value   ← Send poll message to Producer i
10.       change_degree←|current_value  - previous_value|/(MAX - MIN)
11.       if (change_degree  ≤ UTD)
12.          Pull_interval ←Pull_interval + increased_Pull_interval
13.       else
14.          Pull_interval ← Pull_interval - decreased_Pull_interval
15.       if (Pull_interval > PULL_INTERVAL_MAX)
16.          Pull_interval ← PULL_INTERVAL_MAX
17.       if(Pull_interval < PULL_INTERVAL_MIN)
18.          Pull_interval ← PULL_INTERVAL_MIN
19.       previous_value ← current_value
20.  //end while
}// end P&P_Pull operation
```

FIGURE 4: The pseudo code of P&P_Pull Algorithm which runs at the Consumer part.

5. THE PERFORMANCE EVALUATION

The implementation of the Announce with Change and Time Consideration (ACTC), the Adaptive Polling of Grid Resource Monitors using a Slacker Coherence, and the Pull-Push (P&P) algorithms is introduced.

[2] If Pull occurs, push was abandoned in the same period, and vice versa. This is achieved by setting isPushed and isPulled identifiers to be mutual exclusive.

5.1 The Implementation Environment

In our experimental environment, two PCs are used to build a private cloud computing experimental platform, which include PC as master node (the Consumer Pc) and PC as working node (the producer PC). The Consumer and the Producer PCs are connected together using 10/100Mbps Desktop Switch. The two PCs are installed on Linux operating systems by using Ubuntu 11.10 with Linux kernel 3.0 and open source opennebula3 [11], which is used as a cloud computing platform. The Consumer PC is equipped with a Genuine Intel(R) Core(TM) 2 Duo CPU 2.0 GHz with 2.0 GB memory, and the Producer PC is equipped with a Genuine Intel(R) Core(TM) Duo CPU 1.83 GHz with 1.0 GB memory. The monitoring programs for these algorithms are written by using C/C++ language. Also, the multithreading programming and socket programming with C/C++ language are used. There are many resource parameters, which can be used to measure resource status, such as CPU, physical memory, virtual memory, disk space and network equipment data. These parameters can be obtained from the "/ proc" file in the system documents [9]. To simplify the experiments, only one of these resource parameters, i.e. the CPU load percentage, is used to evaluate the performance of the three monitoring algorithms. To evaluate the performance metrics of these monitoring algorithms, the data is gathered through two hours. This data has been used as a constant input to the monitoring algorithms. The gathered data patterns about CPU usage through two hours are shown in Figure (5).

On the other hand, high accuracy and low intrusiveness are considered important metrics for distributed monitoring systems [1]. Also, efficiency, quality, scalability, and robustness are considered the main goals of any resource monitoring protocol which should be achieved at any large distributed system [4]. So, the Implementation results of the three monitoring algorithms will be analyzed and evaluated based on these parameters.

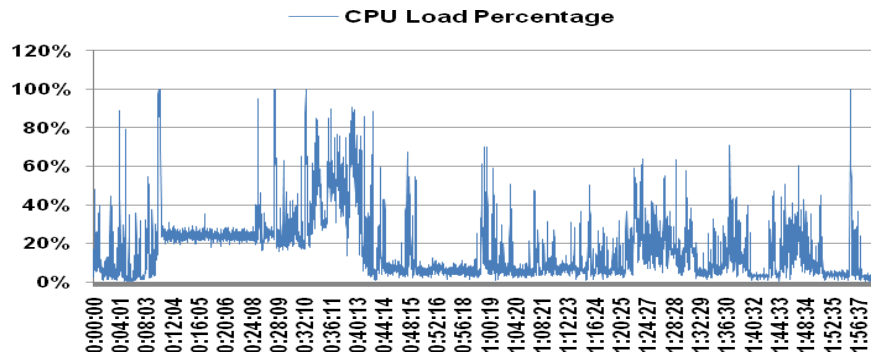

FIGURE 5: the actual usage percentage of the CPU measurements which is gathered for about two hours.

5.2 Evaluation Metrics

To evaluate the performance the three monitoring algorithms, two metrics are used. The First metric has been used to evaluate the accuracy caused by each algorithm. This metric called Standard Deviation (SD). The value of this metric has been calculated by using the following equation:

$$SD = \sqrt{\frac{\sum_{i=1}^{N}(A_i - G_i)^2}{N}} \quad \ldots\ldots\ldots (4)$$

[3] OpenNebula.org is an open-source project developing the industry standard solution for building and managing virtualized enterprise data centers and IaaS clouds.

Where A_1, A_2, …, A_N and G_1, G_2, …, G_N are the points on the graph of the actual measurements and the graph of the Generated measurements respectively. Note that when the generated graph exactly matches the actual graph, SD is zero. The second metric has been used to evaluate the communication overhead caused by each algorithm. This metric called Overhead. The value of this metric has been measured by using the following equation:

Overhead = *no. of updates to Consumer + no. of poll messages from Consumer* … (5).

In the case of pure push algorithms, number of the poll messages equal to zero.

5.3 The Experimental Results

Three groups of experiments have conducted in our environment. The first experiment has been done to compare accuracy, consistency, and the communication overhead of the three monitoring algorithms. This group of experiments has been simulated by using a standard input file, which contains 7210 actual CPU measurements which have been collected every second from the PC which was subjected to a mixture of workload during two hours). The actual and the generated measurements of the monitoring algorithm have been used for calculating SD metric as shown in equation (4).

To avoid runaway events in extreme situations, the upper and lower limit factors of the three monitoring algorithms have been defined [3]. These factors are push/pull intervals, and the threshold (change degree between the current value and its previous). Three seconds has been set as a minimum push/pull interval, 12 seconds as a maximum push/pull interval, 10% as a minimum threshold, and 40% as a maximum threshold. Also, a damping factor of 25% is used in the case of the adaptive polling algorithm and 10% as a moderate change degree for UTD factor in the P&P algorithm.

The second group of experiments has been done for comparing the accuracy, the consistency, and the communication overhead of the three monitoring algorithms to Ganglia monitoring system as a criterion. To conduct this group of experiments, both gmond and gmetad have been run on the Producer PC to monitor the resources of this PC and the Consumer PC to obtain data from the gmond, respectively. The Producer PC hosted a simple LAMP server and ran Wordpress which was subject to a minor load (for about 50 minutes; read every 5 seconds) using the Apache JMeter load testing tool. The other three algorithms (ACTC, Adaptive polling, and P&P algorithm) ran as mentioned before on the two PCs in the same time of running the gmond on the Producer PC.

gmond and the three algorithms ran under the same circumstances. Thirty seconds has been set as a maximum push/pull interval, 1% as a minimum threshold, and 40% as a maximum threshold. Also, a damping factor of 25% is used in the case of the adaptive polling algorithm and 1% as a moderate change degree for UTD factor in the P&P algorithm.

The third group of experiments has been conducted to evaluate the effect of increasing the number of the Producers (physical/virtual machines) on the accuracy degradation degree, and the communication overhead degree of the three algorithms. This effect has been evaluated using small number of Producers (e.g., Two PCs with four Producers, and two virtual machines (VM^4), opennebula command line interface has needed to create VMs, delete VMs… etc [12]) and large number of the Producers (e.g., Two PCs with 300 Producers). Also, 1 sec is used as updating period for the sensor of the producer PC. This sensor has continued to work about two hours with 7,210 times of updating totally.

5.3.1 The Quality and the Efficiency Results

The factors that have a great effect on the quality and the communication overhead of the concerned monitoring algorithms have been examined in the first group of the experiments. The

[4] VM with 1 GHz CPU and 256M memory.

implementation results of the first group of experiments for each algorithm will be discussed in details in the following sections.

5.3.1.1 The ACTC Implementation Results

The quality and the communication overhead of the ACTC algorithm have been affected by changing the minimum threshold. The minimum threshold causes the trade-off between the quality and the communication overhead. As shown in Figures (6), the increasing of minimum threshold degrades the quality, which has been represented by SD, sublinearly while the communication overhead has been decreased.

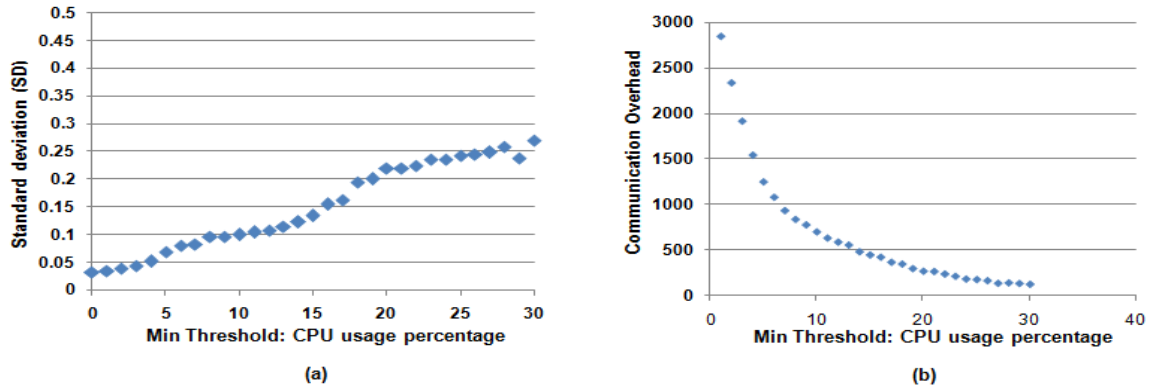

FIGURE 6: The effect of Min threshold on (a) the Quality, which is represented by SD, and on (b) the Communication Overhead of ACTC algorithm.

5.3.1.2 The P&P Implementation Results

The quality and the communication overhead of the P&P algorithm have been affected by the value of the UTD. As shown in Figures (7), the increasing of the UTD degrades the quality logarithmically but decreases the communication overhead (overhead ≈ 7210 / $UTD^{0.46}$).

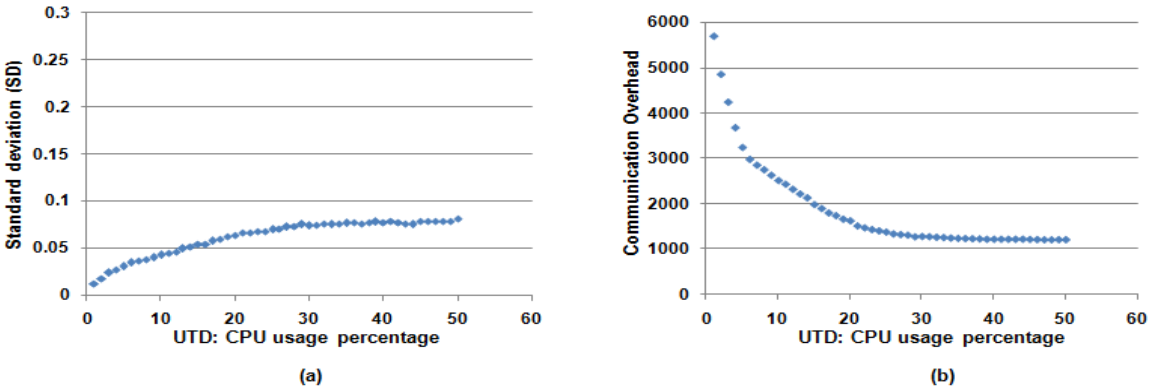

FIGURE 7: The effect of UTD on (a) the Quality, which is represented by SD, and on (b) the Communication Overhead of P&P algorithm.

5.3.1.3 The Adaptive Polling Implementation Results

The quality of the adaptive polling algorithm hasn't been affected by the value of the change_degree as shown in Figure 8(a). On the other hand, changing the value of the change_degree causes a great effect on the communication overhead at beginning and followed by a constant performance as shown in Figure 8(b).

The damping factor (d) has a constant effect on the quality and the communication overhead caused by this algorithm (see Figure (9)). But, the maximum pulling time interval has a

logarithmical effect on the quality as shown in Figure 10(a), and a power effect on the communication overhead as shown in Figure 10(b).

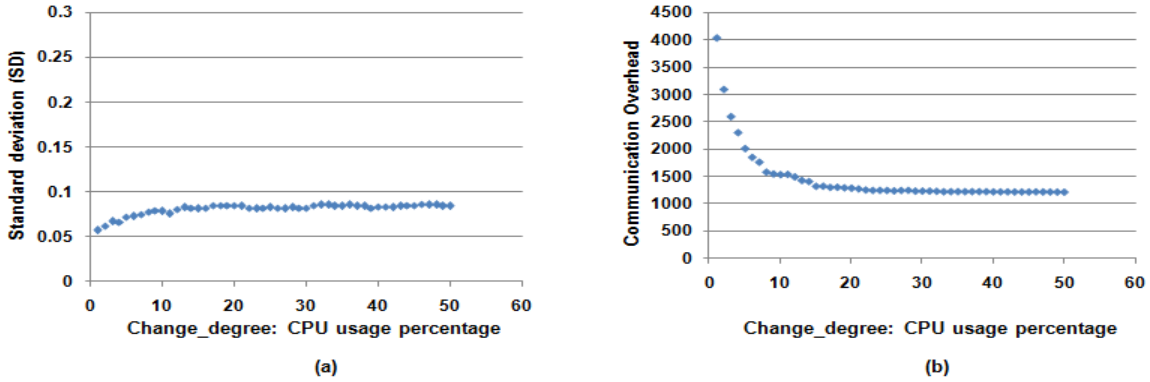

FIGURE 8: The effect of Change_degree on (a) the Quality, which is represented by SD, and on (b) the Communication Overhead of the adaptive polling algorithm.

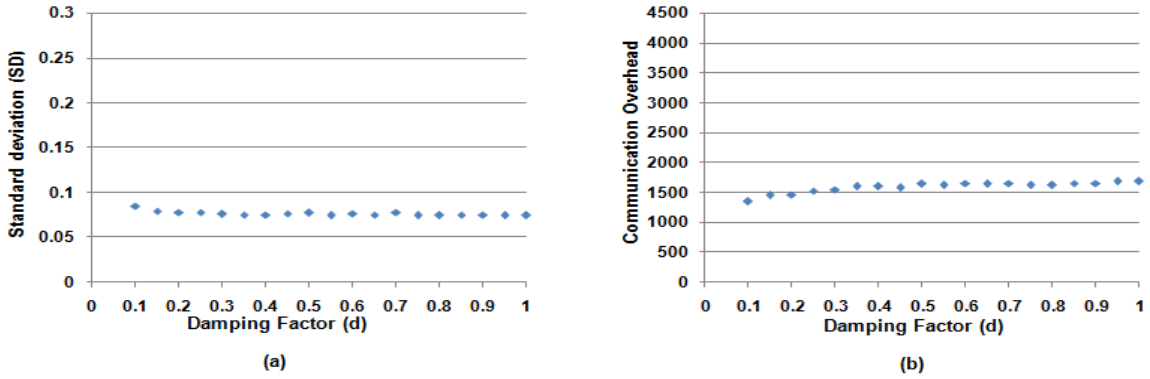

FIGURE 9: The effect of the damping factor "**d**" on (a) the Quality, which is represented by SD, and on (b) the Communication Overhead of the adaptive polling algorithm.

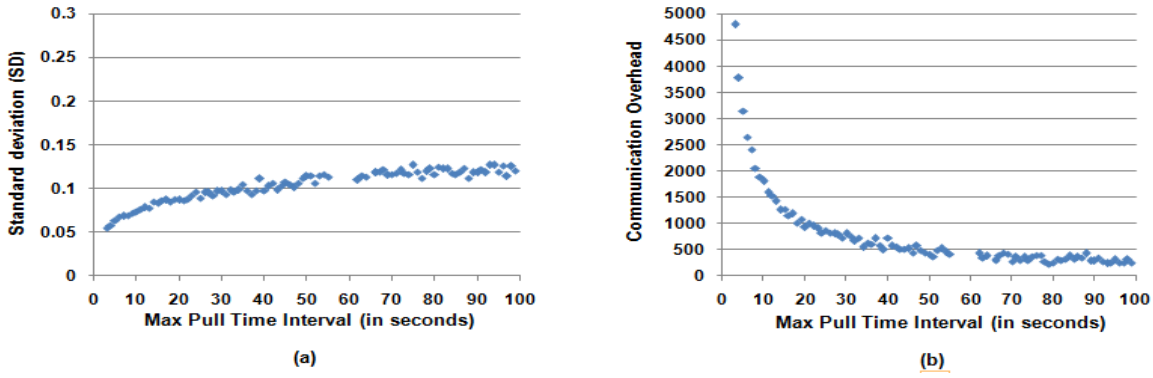

FIGURE 10: The effect of increasing the Max Time Interval on (a) the Quality, which is represented by SD, and on (b) the Communication Overhead of the adaptive polling algorithm.

According to the implementation results, it is found that for each monitoring algorithm, a specific factor would affect its quality and efficiency according to the monitoring way. Generally, the implementation results indicate the following:

- The constant value of the Maximum Pulling Time Interval and the change degree between the current and the previous monitored values have the main effect on the Quality and the Efficiency of the Adaptive Polling algorithm, because the Adaptive Polling has relied on the pulling time interval that changes increasingly or decreasingly based on the value of the change degree.
- The constant value of the UTD has the greatest effect on the Quality and the Communication Overhead of the P&P algorithm. This effect is due to both the P&P-Push and P&P-Pull sections based on the UTD.
- The Quality and the Efficiency of the ACTC algorithm has been affected by the constant value of the Minimum Threshold.

5.3.1.4 Monitoring Algorithms Evaluation

Two groups of experiments have been conducted to evaluate the accuracy and the efficiency of these three algorithms. One of these groups has been conducted for comparing three algorithms together using data set for two hours. Another group has been conducted by using Ganglia monitoring system as a differential criterion.

As shown in Figure 11(a), after comparing three monitoring algorithms together, all the three algorithms have a good quality, where their SD ranges from 4% for P&P to 10% for ACTC. This means that the P&P algorithm has the best quality followed by the Adaptive Polling with 8% and then the ACTC. The P&P algorithm has the highest quality, because it sends a large number of updates to the Master node. But, the more updates are sent, the more communication overhead is caused. So, the P&P has high communication overhead than the ACTC algorithm (see Figure 11(b)). Hence, the ACTC has a good quality, and the best efficiency.

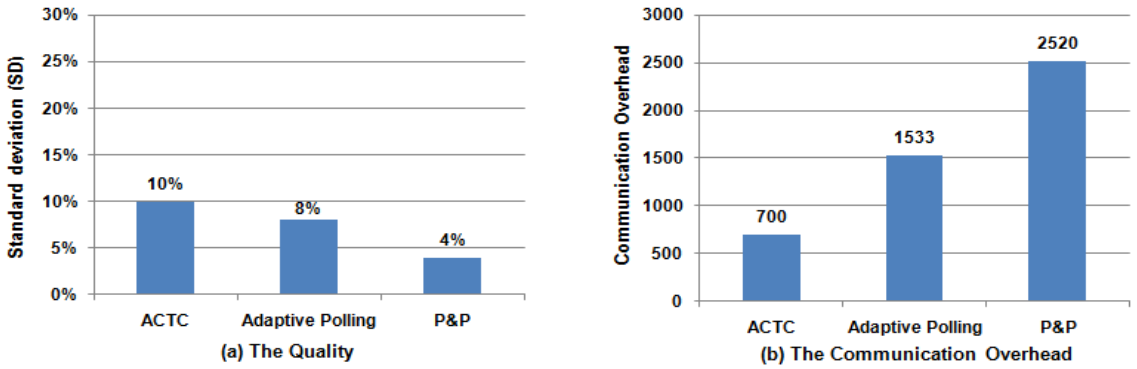

FIGURE 11: Comparison of the three monitoring algorithms under the same circumstances in terms of (a) the Quality, which is represented by SD, and (b) the Communication Overhead.

As shown in Figure 12(a), after comparing three monitoring algorithms together using Ganglia as a differential criterion, the two algorithms (P&P and ACTC) have a good accuracy compared to ganglia, where their SD is less than 10%. But, the Adaptive Polling algorithm achieved the worst accuracy, where its SD is more than 15%. The P&P algorithm achieves a good quality, because it sends a large number of updates to the Master node. But, the more updates are sent, the more communication overhead is caused. So, the P&P has high communication overhead than the ACTC algorithm (see Figure 12(b)). Hence, the ACTC algorithm has a good quality and a good efficiency compared to ganglia and the other two algorithms (P&P and Adaptive Polling).

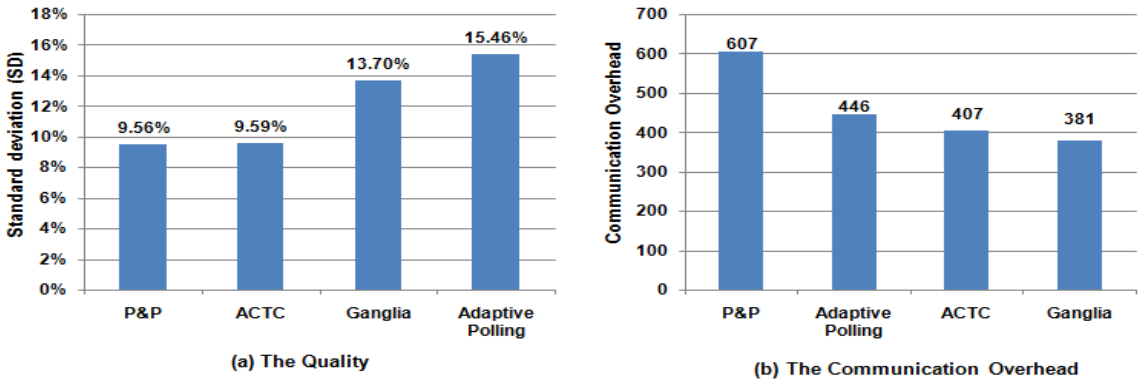

FIGURE 12: The performance evaluation of the three monitoring algorithms using Ganglia as a differential criterion in terms of (a) the Quality, which is represented by SD, and (b) the Communication Overhead.

5.3.2 The Scalability Experimental Results

The third group of the experiments has been conducted to detect the relation between increasing the number of the Producers and the quality degradation degree of three monitoring algorithms. The first experiment of this group has been conducted in the small environment by running 1, 2, 3, and 4 Producer processes on one VM, two VMs, one PC and two VMs, and two PCs and two VMs respectively. According to the results of this experiment, the communication overhead of three algorithms has increased linearly by increasing the number of the Producer processes as shown in Figure 13(a). Hence, the three algorithms are scalable algorithms with the small environments.

The third experiment of this group has been conducted in the large environment by running 1, 50, 100, 300 Producer processes respectively on the Producer PC. According to the results of this experiment, only the communication overhead of the ACTC algorithm has increased linearly by increasing the number of the Producer processes. As shown in Figure 13(b), comparing with the other two algorithms, the ACTC is the most scalable algorithm beside the stability of its quality and accuracy. As shown in Figure 13(b), comparing with the other two algorithms, the ACTC is the most scalable algorithm beside the stability of its quality and accuracy with any environment (the small and the large environments).

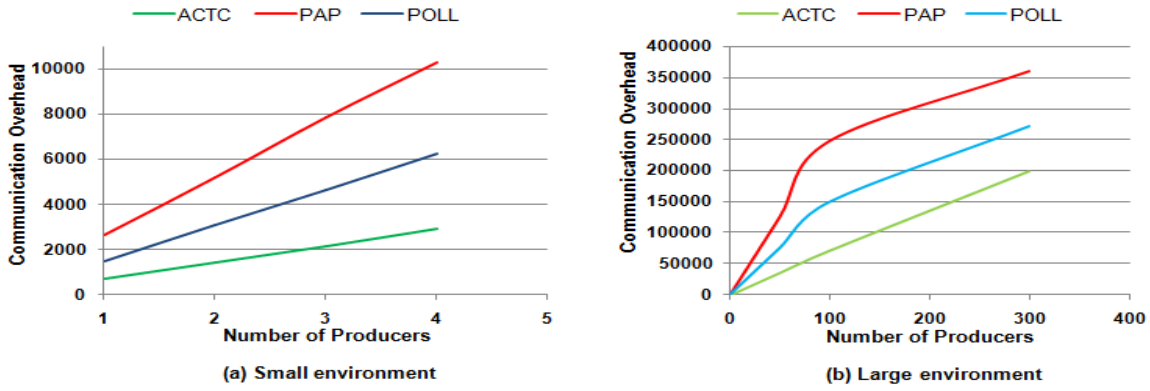

FIGURE 13: The scalability of the three monitoring algorithms under the same circumstances in (a) A Small environment and in (b) A large environment, which had a large number of Producers up to 300 Producers.

According to the evaluation results, the effect of increasing the number of the Producers' updates (total received updates by the Master node) is depicted by the following; the Adaptive polling and P&P algorithms have an excellent degree of quality with the small numbers of the Producers. But, this quality would be degraded with the large number of Producers, because the two algorithms

depend on pulling the status information from the Producer. In the pull, two messages are sent; one message from Master to Producer and one from Producer to master. So, with the large number of Producers, the network consumption will increase and the load on the Master will also increase. Hence, the total number of missed updates will increase. But in the case of the ACTC algorithm, the algorithm depends only on the push model, which sends the important updates only. The Master node receives updates from the Producers without sending anything to them. So the increasing the number of Producers won't have a considerable effect on the load on the Master node and this will decrease the communication overhead. Hence, the quality remains stable.

According to the results of the first and the third group of the experiments, we can conclude the following:
The most important characteristics of the ACTC algorithm are:

- The lack of intrusiveness; due to the lack of the pull operations,
- The high efficiency; due to the communication overhead increases sub-linearly with increasing the resources to be monitored, while the quality and the accuracy stay stable,
- The high scalability degree, and robustness; as, the ACTC algorithm continues to work after adding or removing any Producer without failure.

The most important characteristics of the P&P and Adaptive Polling algorithms comparing to the ACTC algorithm are:

- The intrusiveness; the P&P algorithm has a non-negligible degree of the intrusiveness. But, the intrusiveness dominated in the case of the Adaptive Polling algorithm.
- The low efficiency; due to the communication overhead and the network bandwidth consumption increases nonlinearly with increasing the system size,
- The instable quality; with the large numbers of the resources.
- Scalable only with the small environments, and robustness; as the P&P algorithm continues to work after adding or removing any Producer without failure.

According to the experiments results, it is found that the ACTC algorithm achieves the main goals which should be achieved by any monitoring algorithm used for monitoring the resources in the large distributed systems as mentioned in [4].

6. CONCLUSIONS AND FUTURE WORK

A comparative study has conducted to decide which of the three monitoring algorithms (ACTC, Adaptive polling, and P&P algorithm) are the most suitable for the cloud computing and the large distributed systems. The results show that the ACTC algorithm is the most suitable one, where it achieves a good quality and efficiency degree compared to Ganglia monitoring system, and a small communication overhead degree. Also, the ACTC algorithm is a scalable monitoring algorithm due to the communication overhead increases sub-linearly while the quality doesn't change with increasing the system size. Also, we can conclude that the monitoring algorithms that play good with the large distributed systems are the ones that based on the pure push model. But, the monitoring algorithms, which based on the pull model, consume the bandwidth of the network and increase the load on the Master node.

In the future work, the effect of increasing the history size of the updating series on the quality degree of the ACTC algorithm and the network bandwidth consumption will be studied. Also, introducing the Markov Chain model (MCM) as a predictor to improve the ACTC algorithm. Finally, we plan to conduct these experiments in a large scale Cloud computing environment.

7. REFERENCES

[1] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, M. Wong (May 2009), "Resource Monitoring and Management with OVIS to Enable HPC in Cloud Computing Environments," 23rd IEEE International Parallel & Distributed Processing Symposium (5th Workshop on System Management Techniques, Processes, and Services). Rome, Italy, PP 1 – 8.

[2] J. Park, K. Chung, E. Lee, Y. Jeong, H. Yu (May 2010), "Monitoring Service Using Markov Chain Model in Mobile Grid Environment," the 5th international conference on Advances in grid and pervasive computing (GPC 2010). Hualien, Taiwan, PP 193-203.

[3] H. Huang, L. Wang (Jul 2010), "P&P: A Combined Push-Pull Model for Resource Monitoring in Cloud Computing Environment," 3rd IEEE International conference on Cloud Computing (CLOUD 2010). Miami, FL, PP 260 – 267.

[4] F. Wuhib, R. Stadler (2011). Distributed Monitoring and Resource Management For Large Cloud Environments. KTH R. Inst. of Technol., Stockholm, Sweden. PhD Thesis.

[5] W. Chung, R. Chang (2009), "A New Mechanism For Resource Monitoring in Grid Computing," Future Generation Computer Systems - FGCS 25, PP 1-7.

[6] R. Sundaresan, M. Lauria, T. Kurcy, S. Parthasarathy, J. Saltz (June 2003), "Adaptive Polling of Grid Resource Monitors Using a Slacker Coherence Model," The 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03). Seattle, Washington, PP 260-269.

[7] Cloud Computing: Principles and Paradigms, Edited by R. Buyya, J. Broberg and A. Goscinski © 2011 John Wiley & Sons, Inc.

[8] H. Fang-fang, P. Jun-jie, Z. Wu, L. Qing, L. Jian-dun, J. Qin-long, Y. Qin (Oct 2011), "Virtual Resource Monitoring in Cloud Computing," Journal of Shanghai University, 15, PP 381-385.

[9] J. Ge, B. Zhang, Y. Fang (2010), "Research on the Resource Monitoring Model Under Cloud Computing Environment," The International Conference on Web Information Systems and Mining (WISM'10). Sanya, China, PP 111-118.

[10] B. S. Ghio, (March 2012). Project of a SDP prototype for Public Administrations and private networks. Faculty of Mathematics, Physics and Natural Sciences, University of Genoa. Master of Science in Information Technology.

[11] http://opennebula.org/about:about

[12] http://opennebula.org/documentation:archives:rel2.0:vm_guide.

[13] J. S. Ward & A. Barker (2012), "Semantic Based Data Collection for Large Scale Cloud Systems", DIDC '12 Proceedings of the fifth international workshop on Data-Intensive Distributed Computing Date, New York, USA.

[14] Ganglia: http://ganglia.sourceforge.net.

[15] R. Bhatnagar & J. Patel (2013), "Performance Analysis of A Grid Monitoring System - Ganglia." International Journal of Emerging Technology and Advanced Engineering 3(8): 362-365.

[16] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski (August 2002), "A Grid Monitoring Architecture," The Global Grid Forum Draft Recommendation (GWD-Perf-16-3).

[17] M. Wu & X.H. Sun. (2006), "Grid harvest service: a performance system of Grid computing," Journal of Parallel and Distributed Computing, 66(10): 1322-1337.

[18] I. Foster, Y. Zhao, I. Raicu, and S. Lu. (2008), "Cloud computing and Grid computing 360-degree compared", Grid Computing Environments Workshop, Austin, TX, PP 1-10.

[19] M.L. Massie, B.N. Chun, and D.E. Culler (2003), "The ganglia distributed monitoring system: design, implementation and experience," Parallel Computing, 30(7): 817-840.

[20] Nagios, available in: http://www.nagios.org.

[21] H. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu (2003), "MonALISA: a distributed monitoring service architecture," Computing in High Energy and Nuclear Physics 2003 Conference Proceedings (CHEP03), California, USA.

[22] A. Cooke, A.J.G. Gray, L. Ma, and W. Nuttetal (2003), "R-GMA: an information integration system for grid monitoring," Proceedings of the 11th International Conference on Cooperative, Information Systems, Catania, Sicily, Italy , PP 462–481.

[23] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G.L. Rubini, G. Tortone, and M.C. Vistoli (2005), "GridICE: a monitoring service for grid systems," Future Generation Computer Systems 21(4): PP 559–571.

[24] J.S. Park, H.C. Yu, K.S. Chung, and E.Y. Lee (2011), "Markov chain based monitoring service for fault tolerance in mobile cloud computing," IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA), Biopolis , PP 520–525.

[25] G. Katsaros, R. Kübert, and G. Gallizo (2011), "Building a service-oriented monitoring framework with REST and nagios," IEEE International Conference on Services Computing (SCC), Washington, DC, PP 426–431.

[26] S. Clayman, R. Clegg, L. Mamatas, G. Pavlou, and A. Galis (2011), "Monitoring, aggregation and filtering for efficient management of virtual networks," Proceedings of the 7th International Conference on Network and Services Management, Paris, PP 1–7.

[27] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf (2010), "Monalytics: online monitoring and analytics for managing large scale data centers," Proceedings of the 7th International Conference on Autonomic Computing, Ser. ICAC '10, ACM, New York, NY, USA, PP 141–150.

[28] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf (2011), "A flexible architecture integrating monitoring and analytics for managing large-scale data centers," Proceedings of the 8th International Conference on Autonomic Computing, Ser. ICAC'11, ACM, New York, NY, USA, PP 141–150.