An Efficient Data Dissemination Approach for Cloud Monitoring

Xingjian Lu, Jianwei Yin, Ying Li*,
Shuiguang Deng, and Mingfa Zhu

College of Computer Science and Technology,
Zhejiang University, 310027 Hangzhou, China
{zjulxj,zjuyjw,cnliying,dengsg}@zju.edu.cn, brucezmf@gmail.com

Abstract. Cloud computing brings dynamic resource scalability, pay-per-use billing model and simplified developing platforms, however, the monitoring of cloud today is still confronted with the flexibility, scalability, efficiency and performance problems, especially when the scale of cloud platform is being constantly expanding recent years. In this paper, we first present an efficient and intelligent monitoring architecture for cloud platform based on Data Distribution Service(DDS) and Complex Event Processing(CEP), in order to cope with these challenging issues. Then we mainly focus on the monitoring data dissemination, give more details on how DDS is used in this architecture and propose a comprehensive data delivery algorithm to achieve better accuracy and efficiency.

Keywords: Cloud Monitoring, Data Distribution Service, Complex Event Processing.

1 Introduction

As one of the hottest topics in current internet systems, cloud computing has transferred the delivery of IT services to new level that brings comfort of traditional utilities such as water, electricity to its users by dynamically scaling the service provision. Such dynamic scalability and service level agreement (SLA) negotiability of cloud computing result in a strong demand for monitoring.

Resource monitoring, which has been widely used for software optimization, profiling, performance evaluation, etc [1], is the premise of many major operations such as fault detecting, network analysis, job scheduling, and load balancing in cloud systems [2]. Organizations that are using right mix of technologies for cloud monitoring are more likely to enjoy following business benefits: prevention and resolution of performance issues in a timely manner, ability to support changes in business demand, ability to optimize spending decisions, etc [3].

However, monitoring the cloud at runtime is very challenging. Firstly, much more monitoring concerns need to be covered in clouds than in traditional software system, and individual monitoring schemas and mechanisms need to be

* Corresponding author.

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 733–747, 2012.

designed and implemented respectively, due to the heterogeneity of components in the cloud. An integral cloud monitoring system should cover all the concerns, for satisfying the needs of different roles in the cloud.

Furthermore, each monitoring technique needs to consume some computing resources to take effect. Therefore, it will lead to some undesired runtime overhead to a running cloud. It is a challenging issue to keep such overhead within an acceptable range. Although the weaker the monitoring ability is, the lower the runtime overhead is, enough monitoring ability is still required to ensure the healthy operation for a running cloud. Consequently, how to balance the tradeoff between monitoring ability and runtime overhead is one of the most important issues in cloud monitoring system.

Lastly, due to the large number of services and end users in cloud, monitoring applications have to process a massive amount of runtime information for sending alerts or triggering some actions once something noteworthy happens. Usually, this information is provided in a steady stream of separate events which are detected by certain monitoring sensors. As a result, an efficient and robust communication infrastructure is required, for facilitating the dissemination of monitoring events with high throughput and low latency. Additionally, it will also need to apply real-time intelligence to the management of your cloud infrastructure through making automated decisions. For example, we can anticipate upcoming peak loads and provide the necessary capacity in advance to avoid performance slowdowns through cloud monitoring.

Therefore, in order to deal with these difficulties, an efficient and intelligent monitoring architecture for cloud platform is first proposed in this paper. In this architecture, an efficient and robust data dissemination framework is implemented to transmit the monitoring information reliably with high throughput and low latency based on Data Distribution Service (DDS)[5]. An intelligent cloud action platform is also developed to deal with infinite dynamic event stream based on Complex Event Processing (CEP) [6]. So it can filter out the meaningful information from the event flood to support decision making.

Then, in this paper, we mainly focus on monitoring data dissemination, after describing how DDS is used in this monitoring architecture, an extended comprehensive data delivery algorithm Papx is proposed to achieve better accuracy and efficiency based on the theory of temporal locality. Through this algorithm, the agent of our monitoring architecture can perform well on the balance between runtime overhead and monitoring capability with the adaptive updating frequency regulation.

The organization of this paper is as follows: section 2 gives an overview of the efficient and intelligent monitoring architecture for cloud platform. Then in section 3, details of how DDS is used in this architecture and the concrete implementation of Papx are presented. After that, section 4 shows the experimental evaluation results of the proposed algorithm. In section 5, relevant work concerning cloud monitoring is introduced. Finally, we summarize conclusions.

2 Efficient and Intelligent Monitoring Architecture for Cloud Platform

The scale of cloud computing platform has being constantly expanding recent years. According to reports, Google cloud computing platform already has more than one million servers. Amazon, IBM, Microsoft, Yahoo and other companies each also has hundreds of thousands of servers for their cloud computing. Besides the powerful computing and storage capacity, this kind of fairly large scale also brings some challenging issues for cloud monitoring.

One one hand, how to transmit the huge monitoring data to the server with high throughput and low latency is one of the most challenging issues, especially when it deals with thousands to millions physical servers. On the other hand, due to large amount of real-time data will be generated in this large scale cloud monitoring system, how to extract user required information from these confused data to provide strong support for decision making is also one of the most challenging issues in cloud monitoring.

In order to deal with these challenges, an efficient and intelligent monitoring architecture for cloud platform is proposed based on DDS and CEP in this section. As described in Fig. 1, different types of monitoring facilities are contained in a monitoring agent to collect runtime information from entities of each level of the cloud in timely manner. Then these runtime information will be encapsulated to events, so as to be delivered to the server by DDS efficiently and timely. After that, on the one hand, for facilitating manual control, analysis and display, monitoring server will save these data in database and system logs for persistency and then show to cloud operators, service developers and end users through different views. On the other hand, the cloud action platform will do a variety of complex checking and statistics to trigger related operations through CEP, in order to provide intelligent decisions for cloud management system.

Why and how we use DDS to transmit the huge monitoring information will be described more details in section 3. Due to space limitation, we just give an overview of the cloud action platform in this paper. The core component of the cloud action platform is a CEP engine, which can coordinate and refine the simple events to abstract the complex event for intelligent decision according to the monitoring rules and event schemas [8].

In addition, for reducing the coupling of event rules and decision actions and making the code reading and maintain easy, a portal that can help users configure the complex events based on customized monitoring rules is also developed in this cloud action platform. In this way, users or the third-party systems can define their own complex events conveniently, and they can also modify the rules dynamically when users' requirements or system runtime status changed.

3 Efficient Data Dissemination for Cloud Monitoring

As the number of nodes in clouds reaches a high value, vast runtime information will be send to monitoring server. It is likely to cause network congestion and

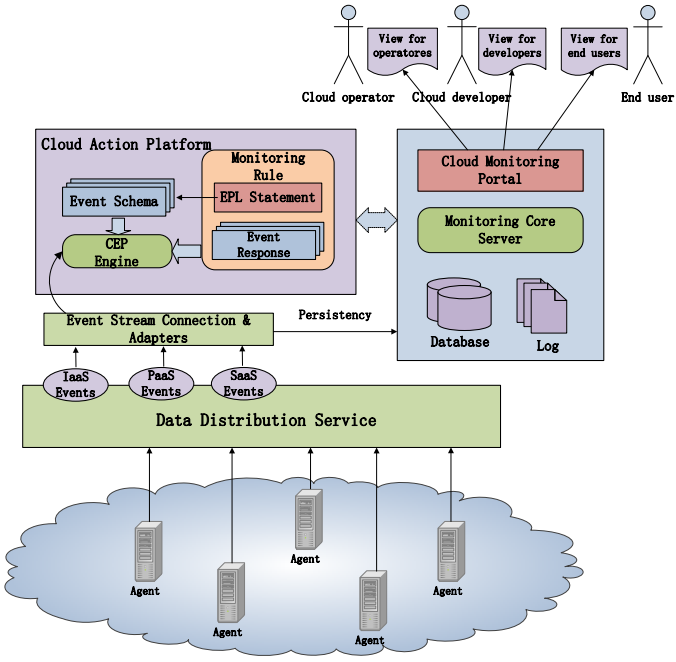


Fig. 1. Efficient and Intelligent Monitoring Architecture for Cloud Platform

make system throughput decline. For neutralizing the impact of this challenge, DDS will be used to delivery these huge monitoring data. Furthermore, an extended data delivery algorithm that can deal with the balance between runtime overhead and monitoring capability will be made to cope with this challenge too.

3.1 Data Dissemination Based on DDS

DDS is an emergent platform-independent standard that defines a data centric publish/subscribe interaction paradigm. It particularly addresses the needs of real-time applications that require deterministic information exchange, low memory footprints and high robustness requirements.

Design of the Data Model. Entities required to be monitored in cloud system can be classified into five categories: hardware, virtualization, middleware, application and interaction [4]. However, due to the tradeoff between efficient data transfer and flexible interpretation of these data, the data of each entity are not published as a whole in our cloud monitoring architecture. We divide the monitoring data of each entity into three categories: *EntityBasicInfo*, *EntityStatus* and *EntityEvent*. Note that the entities of interaction can be subsumed to a specific *EntityEvent*.

The category of $EntityBasicInfo$ contains the basic information of each monitored entity. These basic information are often published to initialize the entity and rarely to be modified during the runtime. Below an example is provided for the data structure for virtual machine basic information. Each attribute is defined with a data type and a name.

```
struct   VMBasicInfo   {
    long vm_id;       long pm_id;
    int cpu_amount;   int memory_size;
    int disk_size;    string owner;
}
```

The category of *EntityStatus* defines all the runtime status information for each entity. This information come from various monitoring metrics and will be published periodically. As an example, the data structure for virtual machine status is provided below.

```
struct   VMStatus   {
    long vm_id;       string status;
    double cpu_util;  double disk_util;
    double memory_util;
}
```

The event data occurs in a running cloud will be pushed to the server immediately. Furthermore, this kind of events occurs as not frequently as the status updates, so we make it independent from the data type of entity status. There maybe serval kinds of event data types for one entity, while each entity has only one data type for the basic information and status. For example, there are *create*, *start*, *user_login*, *shutdown*, *delete* events for some specific application. Due to the different causes and participants of these events, all of them will be modeled as a data type individually in our cloud monitoring architecture.

Design of the Topic Structure. Topics, which hold one specific type of object defined by one data type, play an important part when designing a distributed publish/subscribe system [7]. In a DDS application, publishers write to topics while subscribers read from them. For our cloud monitoring architecture, a lot of topics will be defined, and each of them is bound to one data type. So we classified them into five categories which are described in Fig. 2.

In the centre of this figure, data topic categories are displayed and the corresponding data types are provided in brackets. The arrow directions indicate whether a participant is a publisher or subscriber with respect to a certain topic. When the monitoring system initializes, agents will publish data into the topic category *BasicInfoInitiation* to register the entities. If required you can publish data into the topic *BasicInfoModification* to modify these basic information. Then during runtime, all the status information are published periodically into the topic category *EntityStatusUpdate* to update the runtime information for

each entity, while the event information are published into the topic category *EntityEventReport* to report these kinds of information in timely manner. In addition, server can publish the command data into the topic category *Command* to control or configure the agent dynamically.

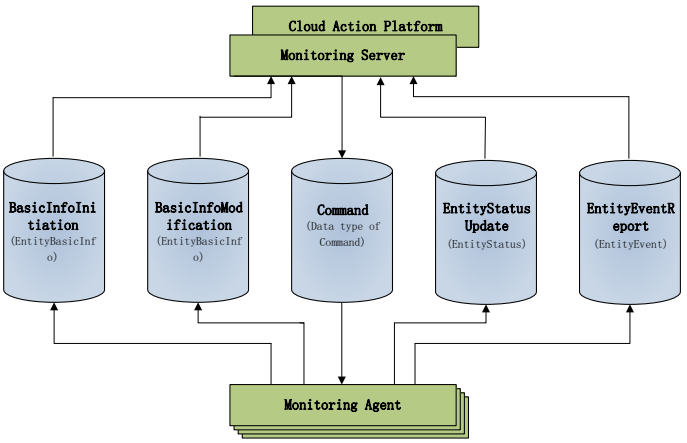


Fig. 2. Topic structure of cloud monitoring system

3.2 Comprehensive Data Delivery Algorithm

As one of the most challenging issues of cloud monitoring, the balance between runtime overhead and monitoring capability has caught the attention of researchers. In addition to dynamically adding or subtracting monitoring facilities of the agent to achieve this balance, a comprehensive data delivery model that combines the pull-based model and push-based model is proposed to deal with cloud monitoring in [2], for decreasing updating times and costs.

However, the Pap algorithm proposed in [2] is inadequate in some respects. First, User Tolerant Degree (UTD) proposed to describe how tolerant a user is to the status inaccuracy, depends on specific application environment and can not vary with the current runtime status dynamically. On the other hand, the increased or decreased pull interval in the pull algorithm is fixed and can not be adjusted according to the current change degree of pulled value.

In order to cope with these deficiencies, we extend this algorithm based on the theory of temporal locality to achieve better accuracy and efficiency in our monitoring architecture. Compared with the Pap algorithm, the extended PapX algorithm can efficiently capture the significant change of monitored values and dynamically adjust UTD and pull interval, for not losing the important updates and improving the accuracy of monitored values.

The extended PapX model consists of two mutual exclusive algorithms: Push algorithm and Pull algorithm. By comparing current change degree of monitored values with dynamic user tolerant degree, the Push and Pull models are alternated adaptively. Before introducing the details of this algorithm, let's have a look at the assumptions and definitions in the following.

Change Degree (CD), defined in Eq. (1), describes the extent of change for monitoring status value between a producer and the corresponding consumer at certain time point.

$$CD(t) = \frac{|P(t) - C(t)|}{Max - Min} \tag{1}$$

Where $P(t)$ denotes the status value of the producer at time t, while $C(t)$ represents the value maintained in the consumer at time t. Max and Min are the maximal and minimal possible value of status separately.

Additionally, in order to capture the change of monitored resource status in recent period, we maintain a sliding window of information about previous updated values for each resource status. Assume $A_0, A_1, ..., A_i, i \leq N$, represents the successive updates to the server in this window, whose size is N. Then the average amount of change (Avg_AC), which describes the average amount of change for the resource status in the sliding window, can be defined as:

$$Avg\_AC = \frac{\sum_{i=1}^{N}(|A_i - A_{i-1}| * i)}{\sum_{j=1}^{N} j} \tag{2}$$

The Dynamic User Tolerant Degree (d_UTD), defined in Eq. (3), describes how tolerant a user is to the status inaccuracy in current sliding window, when taking the average change degree of resource status in current sliding window as adjustable parameters. The value of d_UTD is initialized to user defined UTD and is dynamically calculated. If current change degree of the monitored resource status is larger than the d_UTD, the value will be updated.

$$d\_UTD = UTD \times (1 - \frac{Avg\_AC}{Max - Min}) \tag{3}$$

The core idea behind d_UTD is to decrease the value of UTD, for not losing the important updates, when status changes significantly during current sliding window. Since according to the theory of temporal locality, the larger the status change is, the more attention we should pay on these updated values.

Different from the fixed incremental of pulling interval in [2], the server periodically pull from the agents with a dynamic rate in our PapX algorithm. The pulling rate is adaptively determined based on last pull interval and current damping factor of pull interval, which depends on current average change degree in the window and the user predefined initial incremental value in our algorithm, so the dynamic pull interval (DPI) can be defined as:

$$DPI(t) = \begin{cases} \lfloor DPI(t_0) + STEP \times (1 - \frac{Avg\_AC}{Max - Min}) \rfloor, if \Delta \leq 0, \\ \lfloor DPI(t_0) - STEP \times (1 - \frac{Avg\_AC}{Max - Min}) \rfloor, if \Delta > 0. \end{cases} \tag{4}$$

$$\Delta = CD(t) - d\_UTD(t) \tag{5}$$

Where $DPI(t_0)$ is the last pull interval at time t_0, $STEP$ presents the user predefined incremental of pulling interval. Fig. 3 and Fig. 4 show details of the Push and Pull algorithm separately. In order to avoid Push and Pull operations concurrently happen in a same period, the two operation identifiers, $isPulled$ and $isPushed$ are set to be mutual exclusive to reduce updating times.

```
1   WHILE TRUE
2     set Pull operation identifier isPulled←FALSE waiting for Push_interval
3     IF isPulled equals to TRUE during Push_interval
4       update status information (s_now) that Server currently holds,
5       update the values in the sliding window(can be modeled as a Length
        Fixed Queue)
6     ELSE //check whether need to Push
7       get sensor's current value (sensor_now) at Agent,
8       calculate the value of CD(t), Avg_AC, d_UTD according to Eq. (1), (2),
9       (3) separately
10      IF CD(t)>d_UTD
11        isPushed←TRUE, s_now←sensor_now,
12        push s_now to the Server
13      ENDIF
14    ENDIF
15  ENDWHILE
```

Fig. 3. PapX-Push Algorithm

When the value of d_UTD is relatively small, the Push method dominates, because the condition at line 10 in Fig. 3 is easily to be met, and the Push operations are frequently triggered. On the other side, although the Pull algorithm is trying to minimize Pull interval's value, the PULL_INTERVAL_MIN blocks this trend when Pull interval becomes very small (line 20 to line 22 in Fig. 4).

Similarly, when the value of d_UTD is relatively large, the Pull-based method will dominates. Only when the value of d_UTD is relatively moderate, none of Push and Pull dominates and both of them act frequently. More details about the evaluation and comparison between our extend PapX algorithm and the Pap algorithm proposed in [2] will be described in Section 4.

4 Evaluation

The balance between updating times and accuracy of monitored values plays a very important role on reducing costs and improving efficiency of cloud monitoring. In this sectioon, we evaluate the key performance indexes of the proposed PapX algorithm through experimental based tests.

```
1   Initialize Pull operation's initial interval: PULL_INIT_INL,
    minimal possible interval: PULL_INL_MIN and maximal possible interval:
    PULL_INL_MAX, initial incremental interval: STEP
2   Pull_interval←PULL_INIT_INL
3   WHILE (TRUE)
4     set Push operation identifier isPushed←FLASE
5     waiting for Pull_interval
6     IF isPushed equals to TRUE
7       update status information (s_now) that Server currently holds,
8       update the values in the sliding window (can be modeled as a Length
      Fixed Queue)
9     ELSE
10      isPulled←TRUE, Pull the Agent
11      update s_now,
12      update the sliding window
13    ENDIF
14    calculate the value of CD(t) according to Eq. (1)
15    calculate the value of d_UTD according to Eq. (2) and (3)
16    calculate the value of DPI(t) according to Eq. (4)
17    IF CD(t)≤d_UTD
18      Pull_interval=min{DPI(t), PULL_INL_MAX}
19    ENDIF
20    ELSE IF CD(t)>d_UTD
21      Pull_interval =max{DPI(t), PULL_INTERVAL_MIN}
22    ENDIF
23    s_last←s_now
24  ENDWHILE
```

Fig. 4. PapX-Pull Algorithm

4.1 Experimental Environment

In this experiment, we choose two PCs as a transmission pair to evaluate the performance of our proposed PapX algorithm and the Pap algorithm proposed in [2]. One PC plays as a Producer and the other plays as a Consumer. Each PC is equipped with two Pentium(R) Dual-Core CPU E5200@2.50GHz, 2 GB memory and Ubuntu Release 10.0.4(lucid). To simplify the experiment, we use only the CPU load percentage to test performance of the two models.

This experiment aims for a high accuracy and low intrusiveness data transmission for cloud monitoring. So we analyze and evaluate the two algorithms from the two aspects. For better comparing the accuracy, we define Inaccuracy Degree (ID) to denote the degree of inaccuracy between the value holds on the server and the real value collected by the agent, the expression can be described as follows:

$$ID(t) = \frac{1}{et - st} \times \int_{st}^{et} [C(t) - P(t)]^2 dt \tag{6}$$

where st and et represent the start and end time of monitoring separately. $C(t)$ denotes the value that the server holds at time t, while $P(t)$ denotes the value collected by agent at time t.

4.2 Experimental Analysis

For facilitating repeating the experiments to analyze the efficiency and accuracy of the two algorithms, we first collected and saved 1000 times of updating. Then in later experiments, the Push interval of the agent was set to 10s, and the server's PULL_INIT_INTERVAL, PULL_INTERVAL_MIN, and PULL_INTERVAL_MAX were set to 5s, 3s and 12s, respectively. In addition, the STEP of Pull interval increment was set to 1s.

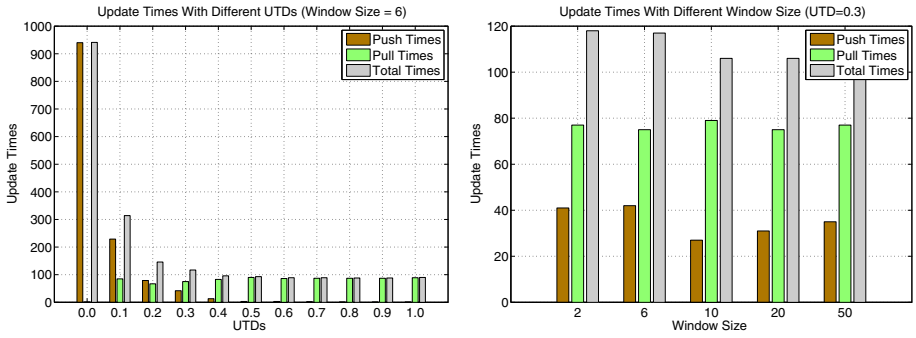
Table 1 describes the comparative results of PapX and Pap algorithm under different window size and UTD. In this group of experiments, we set the presentative values 2, 6, 10, 20, and 50 for window size. And the value of UTD varies from 0 to 1 with an incremental interval of 0.1. For each cell of this table, the value before "/" is the result of Pap algorithm and the value after "/" is the result of PapX algorithm. Due to the limitation of space, the comparative results when UTD is 0.3 and 0.7 are ignored in this table.

Table 1. Comparative results of PapX and Pap algorithm

		0.1	**0.2**	**0.4**	**0.5**	**0.6**	**0.8**	**0.9**
2	Updates	319/310	154/152	96/99	90/93	88/88	87/88	87/88
	Inaccuracy	0.12/0.10	0.23/0.22	0.31/0.30	0.35/0.34	0.35/0.31	0.39/0.36	0.37/0.35
6	Updates	319/314	154/146	96/96	90/93	88/89	87/88	87/88
	Inaccuracy	0.12/0.12	0.23/0.24	0.31/0.29	0.35/0.32	0.35/0.32	0.39/0.36	0.37/0.34
10	Updates	319/310	154/151	96/100	90/93	88/88	87/88	87/88
	Inaccuracy	0.12/0.13	0.23/0.28	0.31/0.30	0.35/0.34	0.33/0.32	0.39/0.36	0.37/0.36
20	Updates	319/323	154/155	96/97	90/93	88/88	87/88	87/88
	Inaccuracy	0.19/0.13	0.23/0.22	0.31/0.30	0.35/0.33	0.35/0.32	0.39/0.35	0.37/0.36
50	Updates	319/326	154/158	96/97	90/94	88/88	87/88	87/88
	Inaccuracy	0.12/0.15	0.23/0.22	0.31/0.33	0.35/0.33	0.35/0.32	0.39/0.36	0.37/0.36

Now, we will analyze these data in table 1. First, we fix the window size, and vary UTD to reveal the relation between updating number and UTD. Fig. 5(a) describes the result when window size is fixed to 6 and UTD varies from 0 to 1. Overall, the total updating number decreases with UTD rising, and the Push operations' number drops dramatically, while the Pull operations' number grows slightly. The reason for this phenomenon is that the rate of Pull operations' number increasing is much less than the rate of Push operations' number decreasing. The figure also proves our analytical result described in section 3.1. That is when UTD is relatively low, most of the updating operations are push, and when UTD is relatively high, the number of Pull operations is dominant.

Also we observe the number of Push operation is not 0 but a small value when the UTD is 1 in our PapX algorithm, it is because the positive feedback effect of



(a) (b)

Fig. 5. Updating number of PapX at different (a)UTDs and (b)window size

dynamic UTD to the rare dramatically changes of monitoring values. In order not to lose the important updating, the PapX algorithm proposed in this paper needes to capture and transmit the significant changes even the UTD is relatively high. In addition, as UTD increases, the number of pull operation increases slowly until it to be stabilized. During this process, little fluctuations occur due to the responding result of dynamic pull interval DPI(t) to the dramatically change of monitoring information.

Fig. 5(b) shows the experimental result of PapX algorithm with different window size and a fixed UTD 0.3. From this figure, we can see that overall the differentiation of the results of different window size is small. The main reason for this phenomenon is that the average change degree of the sliding window is calculated according to the weighted moving average of the change of each two adjacent monitored values, so the influence of the window size is weakened. This is consistent with the theory of temporal locality, since the latest change of these monitored values often has the greatest impact on the eventual result.

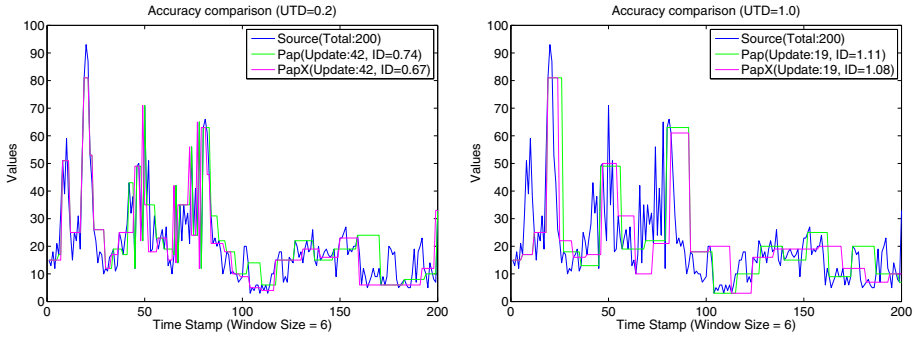
From table 1, we also find the updating number of PapX algorithm is a little bigger than the Pap algorithm. However, the accuracy of PapX is much better than Pap. We tried to give an intuitionistic comparison of the accuracy for the two algorithms in one figure. Unluckily, the amount of the data is so large that the differences of them are not evident in one figure. So in the following experiments, we select the first 200 updates to compare the accuracy of them. Since the window size has a little influence on the eventual result, we fix it to 6.

When UTD is 0, users can not tolerant the deviation of monitored values, so the two alogorithms degenerate to the pure push algorithm, and the inaccuracy of them are both 0. When UTD is relatively low, take 0.2 (Fig. 6(a)) for example, the PapX algorithm has evident superiority than Pap algorithm on accuracy. Although the updating number of PapX and Pap algorithm are both 42, the inaccuracy degree of PapX is 0.67, while Pap is 0.74. This phenomenon is mainly caused by the dynamic UTD. When significant changes of monitored values occur, the dynamic UTD of PapX algorithm will decrease to do more push operations, so the accuracy is improved evidently in PapX.

When UTD is relatively high, take 1.0 (Fig. 6(b)) for example, the push operations of the two algorithms are not triggered in this example. However, the Pap algorithm just modifies the pull interval according to the constant incremental, while the PapX algorithm can modify the pull interval dynamically according to the change degree of monitored values. So when significant change occurs, the pull interval of PapX algorithm may decrease to a lower value immediately, and improve the accuracy of monitored values. As described in Fig. 6(b), although the updating number of PapX and Pap algorithm are both 19, the inaccuracy degree of PapX is 1.08, while Pap is 1.11.

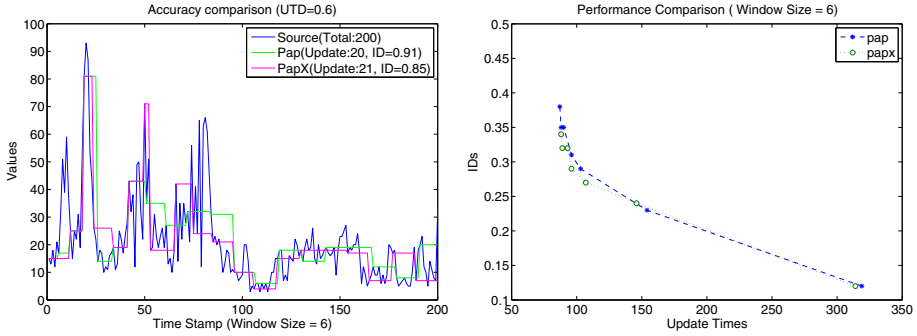
When UTD is relatively moderate, as described in Fig. 7(a) (UTD=0.6), PapX algorithm also has evident superiority than Pap algorithm on accuracy. PapX algorithm just updates one more time and the inaccuracy degree decreases from 0.91 to 0.85. The reason for this phenomenon is mainly the combined influence by dynamic UTD and pull interval described before when significant changes occur.

Through the above logical analysis and experimental comparison, we conclude that the proposed adaptive PapX algorithm can work efficiently with the change of user requirements and monitoring status for cloud monitoring. When compared with the Pap algorithm, PapX has a higher degree of intelligence, since it can efficiently capture the mutations of monitored values to decrease the number of updating and increase the accuracy of monitored values. Fig. 7(b) shows the comparative result on updating number and accuracy for the two algorithms. From this figure, we can see that when the total updating number is low, the accuracy of PapX algorithm is much better than Pap algorithm under the same updating number. As the number of updating increases, the superiority of PapX algorithm on accuracy decreases slowly until the performance of them are analogous.



(a) (b)

Fig. 6. Accuracy comparison between PapX and Pap when (a) UTD=0.2 and (b) UTD=1.0, window size is 6



(a) (b)

Fig. 7. (a) Accuracy comparison between PapX and Pap when UTD=0.6, window size is 6; (b) Comparison on updating number and accuracy for PapX and Pap

5 Related Work

Well-known clouds in the industry all have their own monitoring system. The representative one is App Engine System Status Dashboard, which can be used to show how the applications work in Google App Engine [11]. In addition, a third-party tool CloudStatus [12] is also developed by Hyperic Inc. to monitor Amazon service and Google App Engine.

However, research work concerned with cloud monitoring is relatively less. Due to monitoring of cloud systems typically contains three main activities: collection, dissemination and processing of monitoring information, below we will focus the discussion of related work on them respectively.

Monitoring Data Collecting: A RESTful approach is proposed in [13] to monitor and manage cloud infrastructures. Entities in the cloud are modeled with REST in a tree structure. However, such an organization of monitored information is suitable for cloud infrastructure, and other entities in the cloud can not be modeled appropriately. Therefore, in [4], the authors propose a more universal runtime model for cloud monitoring (RMCM). All the raw monitoring data gathered by multiple monitoring techniques can be organized by this model to present an intuitive representation of a cloud.

Monitoring Data Dissemination: There are two basic data delivery models for communications between consumers and producers: the pull model and the push model [14]. However, a pure push or pull model is not suited for so many different kinds of virtualized resources in cloud system. So a hybrid resource monitoring model called P&P model is proposed in [2] for cloud system. Compared with this hybrid model, our extended model PapX can achieve better suitability and efficiency for cloud monitoring, due to the dynamic UTD and pull interval based on the theory of temporal locality.

For data dissemination methods, the standards-based QoS-enabled pub/sub platforms are promising approaches to build and evolve large-scale monitoring

systems. As an emergent standard for QoS-enabled pub/sub communication, DDS [5] attracts more and more attentions in mission-critical distributed real-time and embedded systems. Compared with the traditional pub/sub platforms, such as CORBA [15], SOAP [16], JMS [17], DDS perform significantly better and are well-suited for data-critical real-time systems [18]. Additionally, in [7], the authors discussed the applicability of DDS for the development of automated and modular manufacturing systems. As far as we know, we are the first to bring DDS into the data dissemination for cloud monitoring.

Monitoring Data Processing: Monitoring applications often involves processing a massive amount of data from a possibly huge number of data sources [19]. CEP [6] has evolved as the paradigm of choice to determine meaningful situations (complex events) for decision making by performing stepwise correlation over event streams in many domains, such as processing of environmental sensor data, trades in financial markets and RSS web feeds [19, 20]. In [21], a complex event language that significantly extends existing event languages to meet the needs of a range of RFID enabled monitoring applications is introduced first, then a query plan-based approach and some optimization techniques are used to efficiently implementing this language.

6 Conclusion

In this paper, an efficient and intelligent monitoring architecture for cloud platform is proposed to deal with the flexibility, scalability and efficiency challenges of cloud monitoring. In this architecture, an efficient and robust data dissemination framework is implemented to transmit the huge runtime information reliably with high throughput and low latency based on DDS. An intelligent cloud action platform is developed to provide decision making support for cloud management system based on CEP. In addition, an extended comprehensive data delivery algorithm Papx is also proposed to achieve better balance between runtime overhead and monitoring capability in this architecture.

Acknowledgment. This work was supported by National Science and Technology Supporting Program (No.2012BAH06F02, No.2011BAD21B02), Research Fund for the Doctoral Program by Ministry of Education of China (No.20110101110066), Science and Technology Program of Zhejiang Province (No.2011C14004), Zhejiang Provincial Natural Science Foundation of China under grant (No.LY12F02029), and National Technology Support Program under grant (No.2011BAH16B04).

References

1. Delgado, N., Gates, A., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Transactions on Software Engineering, 859–872 (December 2004)

2. Huang, H., Wang, L.: P&P: a Combined Push-Pull Model for Resource Monitoring in Cloud Computing Environment. In: IEEE 3rd International Conference on Cloud Computing (2010)
3. White Paper from ManageEngine. Four Keys for Monitoring Cloud Services (March 2010), http://www.manageengine.com
4. Shao, J., Wei, H., Wang, Q., Mei, H.: A Runtime Model Based Monitoring Approach for Cloud. In: IEEE 3rd International Conference on Cloud Computing (2010)
5. Object Management Group. Data Distribution Service (DDS) Brief (2011)
6. Wu, E., Diao, Y., Rizvi, S.: High-Performance Complex Event Processing over Streams. In: SIGMOD 2006, Chicago, Illinois, USA, June 27-29 (2006)
7. Ryll, M., Ratchev, S.: Application of the Data Distribution Service for Flexible Manufacturing Automation. Proceedings of World Academy of Sciency, Engineering and Technology 31 (July 2008)
8. Si-Tu, F.: Event-based Monitoring and Management of the Distributed System, M.Sc. Dissertation, Shanghai Jiao Tong University, Shanghai, P.R. China (2009)
9. Baldoni, R., Bonomi, S., Lodi, G., Querzoni, L.: Data Dissemination supporting collaborative complex event processing: characteristics and open issues. In: DD4LCCI 2010, Valencia, Spain (2010)
10. Bry, F., Eckert, M., Etzion, O., Riecke, J., Paschke, A.: Event Processing Languages. Tutorial in DEBS 2009 (2009)
11. Google App Engine. Google Inc., http://code.google.com/appengine/
12. Cloudstatus. Hyperic Inc., http://www.cloudstatus.com/
13. Han, H., Kim, S., Jung, H., Yeom, H.Y., Yoon, C., Park, J., Lee, Y.: A restful approach to the management of cloud infrastructure. In: Proc. IEEE International Conference on Cloud Computing, CLOUD 2009, September 21-25 (2009)
14. Chung, W.-C., Chang, R.-S.: Chang A new mechanism for resource monitoring in Grid computing. Future Generation Computer Systems 25, 1–7 (2009)
15. Krishna, A.S., Schmidt, D.C., Klefstad, R., Corsaro, A.: Real-time CORBA Middleware. In: Mahmoud, Q. (ed.) Middleware for Communications. Wiley and Sons, New York (2003)
16. Abu-Ghazaleh, N., Lewis, M.J., Govindaraju, M.: Differential Serialization for Optimized SOAP Performance. In: Proceedings of HPDC-13: IEEE International Symposium on High Performance Distributed Computing, Honolulu, Hawaii, pp. 55–64
17. Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K.: Java Message Service. Sun Microsystems Inc., Santa Clara (2002)
18. Xiong, M., Parsons, J., Edmondson, J., Nguyen, H., Schmidt, D.C.: Evaluating the Performance of Publish/Subscribe Platforms for Information Management in Distributed Real-time and Embedded Systems
19. Poul, N., Migliavacca, M., Pietzuch, P.: Distributed Complex Event Processing with Query Rewriting. In: DEBS 2009, Nashville, TN, USA, July 6-9 (2009)
20. Volz, M., Koldehofe, B., Rothermel, K.: Supporting Strong Reliability for Distributed Complex Event Processing Systems. In: Proceedings of 13th IEEE International Conference on High Performance Computing and Communications (HPCC 2011), Banff, Alberta, Canada, pp. 477–486 (September 2011)
21. Wu, E., Diao, Y., Rizvi, S.: High-Performance Complex Event Processing over Streams. In: SIGMOD 2006, Chicago, Illinois, USA, June 27-29 (2006)