P&P: a Combined Push-Pull Model for Resource Monitoring in Cloud Computing Environment

He Huang and Liqiang Wang
Department of Computer Science
University of Wyoming
{hhuang1, lwang7}@uwyo.edu

Abstract

Cloud computing paradigm contains many shared resources, such as infrastructures, data storage, various platforms and software. Resource monitoring involves collecting information of system resources to facilitate decision making by other components in Cloud environment. It is the foundation of many major Cloud computing operations. In this paper, we extend the prevailing monitoring methods in Grid computing, namely Pull model and Push model, to the paradigm of Cloud computing. In Grid computing, we find that in certain conditions, Push model has high consistency but low efficiency, while Pull model has low consistency but high efficiency. Based on complementary properties of the two models, we propose a user-oriented resource monitoring model named Push&Pull (P&P) for Cloud computing, which employs both the above two models, and switches the two models intelligently according to users' requirements and monitored resources' status. The experimental result shows that the P&P model decreases updating costs and satisfies various users' requirements of consistency between monitoring components and monitored resources compared to the original models.

1. Introduction

Cloud computing paradigm makes huge virtualized compute resources available to users as pay-as-you-go style. Resource monitoring is the premise of many major operations such as network analysis, management, job scheduling, load balancing, event predicting, fault detecting, and fault recovery in Cloud computing. Cloud computing is more complicated than ordinary network owing to its heterogeneous and dynamic characteristics. Hence, it is a vital part of the Cloud computing system to monitor the existence and characteristics of resources, services, computations, and other entities [1].

S. Zanikolas and R. Sakellariou present the taxonomy of existing Grid monitoring systems [4]. It indicates that some of the current Grid monitoring systems demonstrate high performance in specific contexts. For example, the widely-used distributed monitoring system, Ganglia [3], is such a tool that can monitor compute resources in Clusters and Grids.

However, monitoring resource in Cloud computing is different from the same task in Cluster and Grid computing [2]. In Cloud computing, the users are exposed to different levels of virtualized services, and the lower levels of resources can be invisible to the users. Even the users may not have the liberty to deploy their own monitoring infrastructure in Cloud computing, and many monitoring approaches developed for Cluster and Grid computing infrastructure cannot work efficiently and effectively under Cloud computing. The same problems may also exist for the developers and administrators, as monitoring different levels of virtualized resources are different. Resource monitoring in Cloud computing requires a fine balance of business application monitoring, enterprise server management, virtual machine monitoring and hardware maintenance, and will be a significant challenge for Cloud computing. For many applications on Cloud computing, it is critical to expose the monitoring and evaluation of underlying resources to the users for appropriate use [1].

In Cluster and Grid computing, resource monitoring infrastructure consists of Producers, Consumers and Directory Services (or Registers). Producers generate status information of monitored resources. Consumers make use of status information. The Directory Service is responsible for locating Producers and Consumers, and enabling bootstrap communication between the two sides. There are two basic methods for communications between Consumers and Producers: the Pull model and the Push model [4]. In the Pull model, Consumers are responsible for "Pulling" information from Producers to inquire status. However, in the Push model, when updates occur at a Producer, under some trigger conditions, the Producer "Pushes" the new resources' status to Consumers. The Push model is more accurate when threshold, *i.e.,* condition determines whether to trigger Push operation, is appropriate; while the Pull model requires less transmission costs when inquiring interval is proper. Cloud monitoring entities can also be modeled as Producers, Consumers, and Directory Services. However, in Cloud computing, especially at the Platform-as-a-Service (PaaS) and the Software-as-a-Service (SaaS) paradigms, the users just see one level of resources such as a predefine API instead of the underlying resources. A pure Push or Pull model is not suited for many different kinds of virtualized resources.

Motivated by the complementary properties of the Push and Pull models in Cluster and Grid computing, we propose a hybrid resource monitoring model called *P&P model* for Cloud computing. Our P&P model consists of Push algorithm and Pull algorithm. The two algorithms are deployed in Consumers and Producers respectively and run simultaneously. The main features include: (1) it has better performance because the P&P model can intelligently switch between Push and Pull styles according to users' requirements and resources' status; (2) It is more appropriate to Cloud computing than the pure Push and the Pull models, which are widely used in Cluster and Grid computing, because virtualized resources may have different privileges and access styles. For example, the applications like alert system are appropriate for the Push manner, applications like database are appropriate for the Pull manner, and the applications like scientific computation may be the combination of the two manners. The experimental results demonstrate this potential and show that the P&P model can provide different consistency between monitoring component and monitored resources and reduce the updating cost considerably compared to the pure Push model and pure Pull model. Note that our P&P model is relatively general and can be easily adopted to other distributed or service-oriented systems, such Cluster and Grid computing environment.

The rest of the paper is organized as follows. Section 2 introduces relevant work concerning Cloud and Grid monitoring. Section 3 discusses the motivation for combination of the two traditional methods. In Section 4, we present the principle and details of the P&P model. Section 5 shows the experimental results of the P&P model. Finally, we summarize conclusions and discuss future work.

2. Related work

The Grid Monitoring Architecture (GMA) [5] proposed by Global Grid Forum, is the prevailing principle adopted by many types of Grid monitoring systems. GMA contains three principal roles: Producers, Consumers, and Directory Services (or termed Registers). Once the contact between Producers and Consumers is established, Consumers can collect information directly from Producers. One main purpose of Directory Services is to facilitate Consumers and Producers to find each other. Another purpose is that Producers or Consumers may be notified if changes happen at the related Producers and Consumers.

Many existing Grid monitoring system is based on GMA. Relational Grid Monitoring Architecture (RGMA) [6][7] offers a global view of the information as if each Virtual Organization is a large relational database. Network Weather Service (NWS) [8], is a distributed system that provides portable and non-intrusive performance monitoring and forecasting, mainly intending to support scheduling and dynamic resource allocation. Globus: Monitoring and Discovery System (MDS) [9], is the information services component of the Globus Toolkit and provides information about the available resources on the Grid and their status.

Foster *et al.* compare the difference for the resource management and monitoring between Grid and Cloud [2]. Because of the major differences on compute model, data model, resource virtualization and sharing, the approaches of resource monitoring for Grid computing has to be revised for Cloud environment. This motivates us to design a hybrid monitoring models to deal with different Cloud computing types such as Infrastructure-as-a-Service (IaaS), PaaS, and SaaS.

Brandt *et al.* propose a tool called OVIS for monitoring resources to enable high-performance computing (HPC) in Cloud computing environment [1]. Since intelligent resource utilization is critical to enable efficient HPC applications, OVIS can dynamically characterize the resource and application state, and optimally assign and manage resources based on the monitored information. OVIS mainly uses statistical analysis to scale data collection and resource allocation.

One common concern for Grid and Cloud resource monitoring is the relationship between consistency and efficiency. On one hand, a monitoring system must collect resources' status as frequently as possible in order to keep Consumers' information consistent with Producers. On the other hand, the frequency of communications is directly related to network consumption. Therefore, a trade-off between consistency and efficiency is required. For example, GHS [10] uses an adaptive measurement methodology to monitor resource usage patterns, where the measurement frequency is dynamically updated according to the previous measurement history. This method obtains relatively accurate patterns and reduces monitoring overhead considerably.

3. Motivation for combining Push model and Pull model

As we mentioned in Section 1, there are two basic methods for communications between Consumers and Producers: the Pull model and the Push model [4].

In the Push model, the initiator is the Producer. The Producer sends status information when it detects that the status changes are greater than the threshold. It is ideal for keeping maximum consistency between the Producer and the Consumer if threshold is proper. However, if threshold is small, minor changes result in too much information transmission, which may place strain on the network. If threshold is large, important updating may be lost. Based on Push model, Wu-Chun Chung and Ruay-Shiung Chang

[13], attempt to minimize the useless updating, and maximize information consistency between Consumers and Producers. They propose three approaches, which are the Offset-sensitive mechanism (OSM), the Time-sensitive mechanism (TSM), and the Hybrid mechanism incorporating OSM and TSM. These mechanisms partially improve performance of the Push model.

In the Pull model, the Consumer is the initiator. As such, the low Pulling rate consumes little network bandwidth, but may imply missing important updating during the Pulling interval, which is undesirable for Consumers. However, information at high Pulling rate is "fresher" but heavily intrusive to the original system. Based on the Pull model, R. Sundaresan *et al.* [14][15] propose an adaptive polling using the time series information obtained from the sensors to estimate the time of the next significant update. This model, to some extent, improves accuracy. However, it is based on Pull model in nature, and it still cannot avoid Pull model's drawbacks completely.

M. Bhide *et al.* [16] study adaptive Push-Pull strategy to disseminate dynamic web data. They combine the two methods in the Web research area and present "Push and Pull" (PaP) method as well as "Push or Pull" (PoP) method. The PaP method simultaneously employs both Push and Pull methods to exchange data, but has tunable parameters which determine the degree to which Push and Pull are used. The PoP method allows servers to adaptively choose between Push and Pull methods for each connection. Disseminating web data has many similarities with Cloud resource monitoring. Hence the idea of combining Push and Pull methods can also be extended to research in Cloud resource monitoring.

Pay-as-you-go is one of important features for Cloud computing, and this feature relies on virtualization of the computing facilities. Virtualization middleware requires real-time system status monitoring for decision making and optimization. But large-scale virtualization usually involves thousands of nodes, and frequent status information transmission will greatly degrade network efficiency. The Push model is more accurate when threshold is appropriate, while the Pull model performs less transmission costs when inquiring interval is proper. Since the two models have complementary properties under specific condition, a reasonable trade-off between the two models becomes possible. Thereby, we propose a combined Push and Pull model for Cloud computing.

4. The P&P resource monitoring model

4.1. Overview

The P&P model is responsible for interaction between the Producer and Consumer in each pair. The Push and Pull operations are mutual exclusive. By comparing status of the monitored resources and user's requirement, the Push and Pull models are alternated in our P&P model.

The change degree, defined in (1), describes the extent of change between the current status of a Producer and the status preserved in the corresponding Consumer. Every status' information contains a time stamp which records the time at which a Producer collects status information. The t_p represents a Producer's closest time stamp prior to time t, and similarly, t_c represents a Consumer's closest time stamp prior to time t. We have $t_p \geq t_c$ because a Producer's update is always prior to a Consumer's update. Therefore, $P(t_p)$ denotes the "real" status of the Producer at time t and $C(t_c)$ denotes the status information that the Consumer holds at time t. In fact, $C(t_c)$ represents the last update that the Consumer receives. MAX and MIN are the maximal and minimal possible value of status.

$$\text{change\_degree} = \frac{|P(t_p)\text{-}C(t_c)|}{\text{MAX-MIN}} \leq \text{UTD} \ \ (t_p \geq t_c) \quad (1)$$

The requirements of users are expressed by the concept of User Tolerant Degree (UTD), which describes how tolerant a user is to the status inaccuracy. A small UTD indicates that the user has strict accuracy requirements. On the opposite, a large UTD indicates that the user is prepared to tolerate a significant level of inaccuracy. The P&P model aims to keep the change degree not greater than the UTD. Note that the UTD is a value depending on specific application environment and requiring users' prior experiences. The UTD can be set on-the-fly in the similar approach as the setting of threshold in [13], which is given by users or administrator's according to their preference.

For every Consumer-Producer pair, the most appropriate monitoring model depends on the value of UTD. The strategy for three possible cases is shown in (2).

$$\begin{matrix}\text{monitoring}\\ \text{strategy}\end{matrix} = \begin{cases}\text{push-based dominates (UTD is relatively small)}\\ \text{pull-based dominates (UTD is relatively large)}\\ \text{none dominates (UTD is relatively moderate)}\end{cases} \quad (2)$$

In the first case, the user has strict requirements for the monitored resources. For example, a user assigns computational tasks on some resources, and it requires up-to-date knowledge of these resources' status. The user is very sensitive to small changes which may occur at the resources. Once the status' change of a Producer is larger than the threshold predefined according to UTD, the Producer "pushes" the status information to the Consumer. In some rare situations, the status' changes are too small to trigger the Push operation, but the Pull method will be activated periodically to avoid the Producer's unavailability for a long period. Consequently, from the global view, the Push method dominates the monitoring strategy in this case.

In the second case, the user desires only coarse information regarding the resources. For instance, a user may wish to know the approximate load information about some resources on particular days to decide whether to choose these resources for specific tasks. The user only needs to Pull the resources periodically such as on hourly basis. Also, in very few case, the status of monitored resources changes noticeably, and Push operations are needed during the two Pull operations. But in all, the number of Pull operations greatly outweighs that of Push operations when UTD is relatively large.

In the last case, the Consumer's requirement is relatively moderate. The "correct" decision is ambiguous, because the strictness of the user's requirement is between the two above cases. Depending on specific circumstance, either Push or Pull operations are triggered irregularly. Neither Push nor Pull operation overwhelms the other one.

4.2. Algorithm description and analysis

The P&P model is composed of P&P-Push algorithm and P&P-Pull algorithm. Figure 1 and Figure 2 show details of the P&P-Push and P&P-Pull algorithms.

```
1   WHILE TRUE
2     set Pull operation identifier isPulled ←FALSE
3     waiting for Push_interval
4     IF isPulled equals to TRUE during Push_interval
5       update status information (c_now) that Consumer currently holds
6     ELSE //examine whether need to Push
7       get sensor's current value (sensor_now) at Producer
8       IF |sensor_now-c_now|/(MAX-MIN)≥UTD [1]
9         isPushed←TRUE, c_now←sensor_now,
10        Push c_now to the Consumer
11      ENDIF
12    ENDIF
13  ENDWHILE
```

Figure 1. P&P-Push algorithm

The P&P-Push algorithm runs at the Producer and the P&P-Pull algorithm runs at the Consumer simultaneously. The two algorithms try to make the resource monitoring system intelligently switch between Push and Pull operations according to user's requirements and status' changes of monitored resources. Now we analyze the algorithms in four aspects.

(a) The Pull operation identifier "isPulled" and Push identifier "isPushed" are set to be mutual exclusive to avoid Push and Pull operations concurrently happen in the same period, which may further reduce updating times, *i.e.* if Pull happens, Push operation in the corresponding interval is abandoned, and vice versa. Therefore, when UTD equals to 0, all Pull operations are forbidden, and the P&P model degrades to the pure Push model. Similarly, when UTD equals to 1, all Push operations are forbidden, and P&P model degrades to pure Pull model. Also, "isPulled" and "isPushed" should be controlled by synchronization model to avoid inconsistency when concurrently reading or writing.

```
1   Initialize Pull operation's initial query interval:
    PULL_INIT_INTERVAL, minimal possible inquiry interval: PULL_INTERVAL_MIN and maximal possible
    inquiry interval: PULL_INTERVAL_MAX
2   Pull_interval←PULL_INIT_INTERVAL
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 (c_now) that Consumer currently holds
8     ELSE
9       isPulled←TRUE, Pull the Producer
10      update c_now
11    ENDIF
12    change_degree=|c_now-c_last|/(MAX-MIN) [2]
13    IF (change_degree≤UTD)
14      IF increased_Pull_interval ≤
           PULL_INTERVAL_MAX
15         Pull_interval=increased_Pull_interval
16      ELSE
           keep current Pull_interval
17      ENDIF
18    ELSE IF (change_degree>UTD)
19      IF decreased_Pull_interval≥PULL_INTERVAL_MIN
20         Pull_interval=decreased_Pull_interval
21      ELSE
           keep current Pull_interval
22      ENDIF
23    ENDIF
24    c_last←c_now
25  ENDWHILE
```

Figure 2. P&P-Pull algorithm

(b) When the value of UTD is relatively small, the Push method dominates. As shown in Figure 1, because the condition at line 8 in the P&P-Push algorithm is easily to be met, Push operations are frequently triggered. On the other side, although the P&P-Pull algorithm is trying to minimize Pull interval's value (line 20 in Figure 2), the PULL_INTERVAL_MIN blocks this trend when Pull interval becomes very small (lines 19 and 21 in Figure 2). In most situations, Push operation runs before Pull operation. Hence, the Push-based method becomes dominant. An extreme case is that there are very few status changes happening at the Producer, so Push operation is not triggered for a long time, but the P&P-Pull algorithm still Pulls the Producer at most PULL_INTERVAL_MAX periods to inform availability of the Producer to the Consumer.

[1] sensor_now corresponds to $P(t_p)$, c_now corresponds to $C(t_c)$ in (1).

[2] c_now corresponds to $P(t_p)$ in (1), because Consumer cannot obtain real-time status of Producer. c_last is the status information that Consumer holds after last update, and it corresponds to $C(t_c)$ in (1).

(c) When the value of UTD is relatively large, the Pull-based method dominates. Because the condition of line 8 in Figure 1 is hard to meet, Push operation is seldom triggered. However, the P&P-Pull algorithm adjusts its Pull interval (lines 8-23 in Figure 2) according to status changes. In this case, the Pull-based method becomes dominant. The extreme situation is that when the Pull interval is very large, a dramatic change violating UTD happens during the very large Pull interval. The Push operation is triggered at this moment and Pushes the unusual status to the Consumer. Meanwhile, the P&P-Pull algorithm tries to decrease Pull interval because the Push operation commonly indicates Figure 2: line 13's condition is met.

(d) When the value of 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 both act frequently.

4.3. An example

Table 1 gives an example using the P&P model with a moderate UTD that equals to 0.25. Initially, the Pull interval is set to 50s and Push interval is set to 10s. During the first Pull interval, at 10:46:30, the Producer Pushes the status since the change degree of Push is 0.31>UTD. At 10:46:50, when the first Pull interval expires, Pull operation in this interval is forbidden since Push operations happen. When the change degree of Pull is 0.31>UTD, the Consumer decrease its next Pull interval to 40s. During the second Pull interval, there was no Push operation. Thus, at the end of the second Pull interval (time stamp: 10:47:30), the Consumer executes Pull operation and increases its third Pull interval to 50s. The next two Pull intervals also follow the rules explained above. The total updating cost is 5 including both Push and Pull operations compared to the cost 19 in the pure Push method.

Table 1. An example employs P&P model[3]

Time stamp	sensor CPU Load (%)	Push change degree	Pull change degree	Push/Pull operation	Push/Pull value	remaining Pull interval
10:46:00	53	0.53	\	***Push***	53	50
10:46:10	48	0.05	\	\	53	40
10:46:20	39	0.14	\	\	53	30
10:46:30	22	0.31	\	***Push***	22	20
10:46:40	27	0.05	\	\	22	10
10:46:50	18	0.04	0.31	\	22	***40***
10:47:00	30	0.08	\	\	22	30
10:47:10	15	0.07	\	\	22	20
10:47:20	8	0.14	\	\	22	10
10:47:30	14	0.08	0.08	***Pull***	14	***50***
10:47:40	44	0.30	\	***Push***	44	40
10:47:50	38	0.06	\	\	44	30
10:48:00	24	0.20	\	\	44	20
10:48:10	38	0.06	\	\	44	10
10:48:20	20	0.24	0.3	\	44	***40***
10:48:30	36	0.08	\	\	44	30
10:48:40	49	0.05	\	\	44	20
10:48:50	22	0.22	\	\	44	10
10:49:00	41	0.03	0.03	***Pull***	41	***50***

5. Experiment

5.1. Experimental environment

Every transmission pair works independently. In every pair, the Consumer and Producer only communicate with its partner within the pair. So we choose two PCs as a transmission pair to evaluate the performance of the P&P model. One PC plays as a Producer and the other plays as a Consumer. Each PC is equipped with Intel Celeron CPU 420@1.60GHz, 2 GB memory and Windows XP operating system. We adopt *Microsoft Management Console 2.0: Performance Logs and Alerts* as sensors in Producers. To simplify the experiment, we use only one of the resource parameters, *i.e.* the CPU load percentage, instead of other parameters to test performance of the P&P model. The Producer and Consumer are implemented by Java programming language with Java's synchronization and multithread mechanisms.

According to [1], high accuracy and low intrusiveness are two important metrics for distributed monitoring system. So our experiments will analyze and evaluate the P&P model in the above two aspects.

5.2. Experimental results

We conducted two groups of experiments in the transmission pair. The first group of experiments aims to reveal the relation between updating number and UTD, and compare updating number of the P&P model with the pure Push and pure Pull models. We set sensor's updating period to 1s. It began at 03/11/2010, 18:02:52.093 and ended at 03/11/2010, 19:33:01.421 with 5,315 times of updating totally. Producer's Push interval was set to 1s, and Consumer's PULL_INIT_INTERVAL, PULL_INTERVAL_MIN, and PULL_INTERVAL_MAX were set to 5s, 3s and 12s, respectively. The Pull interval increased or decreased 1s each time.

The total updating number includes both the numbers of Push and Pull operations. As shown in Figure 3, the total updating number, Push operations' number, and Pull operations' number are closely relevant to UTD. As UTD rises,

[3] UTD=0.25, Push interval =10s, PULL_INIT_INTERVAL=50s, PULL_INTERVAL_MIN=30s, PULL_INTERVAL_MAX=120s, Pull interval=Pull interval+/-10s

the total updating number decreases, and the number of Push operation drops dramatically, but the number of Pull operation grows slightly. The reason for this phenomenon is that the speed at which the number of Pull operations increases is much less than the speed at which the number of Push operations decreases. Another phenomenon is the proportion of Push operation decreases, while Pull operation accounts for more of the total as UTD grows. When UTD is 0, the P&P model degenerates to the pure Push model. When UTD is relatively low, take 0.15 or 0.2 for example, most of the operations are Push, whereas Pull operations only occupy a little percentage. So Push is dominant. If UTD (0.45 or 0.5) is relatively moderate, neither Push nor Pull operations exceed the other too much. When UTD (0.7 or 0.75) is relatively high, the number of Pull operations is dominant. Finally, the P&P model becomes the pure Pull model when UTD is 1.

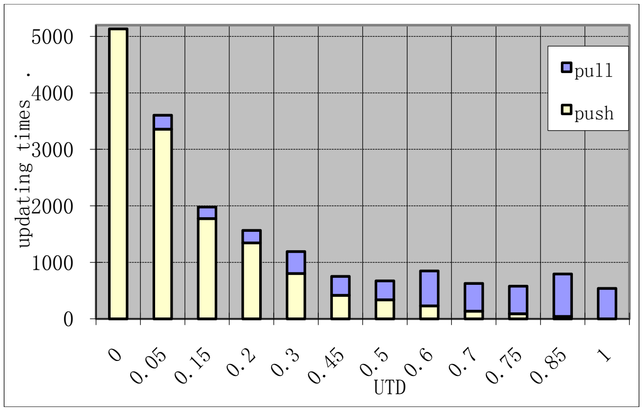

Figure 3. Updating number of P&P model at different UTDs (1st group)

Also we observe that the updating number in the P&P model is much smaller than the updating number in the pure Push method (UTD=0), or the value of UTD is relatively small. This is because the threshold of the P&P-Push algorithm depends on the users' requirements and reduces unnecessary Push operations. On the other side, the updating number of the P&P model is slightly greater than that of pure Pull method (UTD=1) when UTD is relatively large since it involves a small number of Push operations.

In the second group of experiments, the sensor's updating period was 10s. It began at 03/11/2010, 20:08:47.734 and ended at 03/1/2010, 20:41:57.765 with 200 times of updating in total. The Producer's Push interval was set to 10s, and the Consumer's PULL_INIT_INTERVAL, PULL_INTERVAL_MIN, and PULL_INTERVAL_MAX were set to 50s, 30s and 120s, respectively. The Pull interval increased or decreased 10s each time. The goal of this group experiments is to test the coherency between the P&P model and the pure Push or Pull methods at different UTDs.

Figure 4 shows the updating number of the second group of experiments with the P&P model at different UTDs. Figure 5 shows the comparison between the result for the P&P model with relatively small UTD and the pure Push model. The P&P model performs strong coherence with the pure Push method when UTD is relatively low (0.1), but only has half number of updating. Figure 6 (UTD=0.25, moderate) gives detailed operations of the P&P model. The diamond points are Pull operations, and the square points are Push operations. Figure 6 loses more minor details, but is still much similar to Figure 5 with less number of updating. From the graphs we can see that the Pull operations are comparably more regular, while Push operations happen somewhat randomly. This is because Push operations are usually triggered at points that have remarkable status changes.

Figure 7 (UTD=0.4, large) is an outline of the pure Push model in Figure 5, but has more details than the pure Pull model (Figure 8) with increasing a small number of updating. Most of the additional updatings are Push operations caused by dramatic status changes. So UTD=0.4 is acceptable too for users who only desire coarse status trend about resources.

6. Conclusions and future work

In order to make resource monitoring in Cloud environment more flexible and efficient, we propose the P&P model which 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. The experimental results show that, compared with the pure Push model and pure Pull mode, the P&P model can effectively reduce updating number and maintain various levels of coherence in accordance with the users' requirements.

In the future, we will study the method to measure the value of UTD based on users' experiences. Also, we plan to study how to increase or decrease the Pull interval value effectively to adapt to status changes. Then, we would like to synthesize more types of parameters of monitored resources, such as memory, disk capacity, network bandwidth, etc., to reflect more authentic status of resources. Finally, we plan to implement the new model, and test its performance in large scale Cloud computing or other distributed environment.

Acknowledgment

This work was supported in part by the Graduate Assistantship of the School of Energy Resources at the University of Wyoming.

References

[1] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong. “Resource monitoring and management with OVIS to enable HPC in Cloud computing”, Proc. of the 23rd IEEE International Parallel & Distributed Processing Symposium (5th Workshop on System Management Techniques, Processes, and Services), Rome, Italy.2009.

[2] I. Foster, Y. Zhao, I. Raicu, S. Lu. “Cloud computing and Grid computing 360-degree compared”, Grid Computing Environments Workshop, 2008.

[3] Ganglia, http://ganglia.sourceforge.net/, 2010.

[4] S. Zanikolas and R. Sakellariou, “A taxonomy of Grid monitoring systems,” Future Generation Computer Systems 21 (1): pp. 163–188, 2005.

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

[6] A. Cooke, A.J.G. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, “R-GMA: an information integration system for Grid monitoring,” in Proc. 10th International Conference on Cooperative Information Systems, 2003.

[7] A. Cooke, A.J.G. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, “The relational Grid monitoring architecture: Mediating information about the Grid,” Journal of Grid Computing, 2(4): pp. 323-339, 2004.

[8] R.Wolski, N. Spring, J. Hayes, “The network weather service: a distributed resource performance forecasting service for metacomputing,” Future Generation Computer Systems 15 (5/6): pp. 757–768, 1999.

[9] Globus: Monitoring and Discovery System (MDS). Available: http://www.globus.org/toolkit/mds/

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

[11] H. Eichenhardt, R. Muller-Pfefferkorn, R. Neumann, and T. William. “User-and job-centric monitoring: analysing and presenting large amounts of monitoring data,” In the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008), Tsukuba, Japan, pp. 225-232, 2008.

[12] D. Cesini, D. Dongiovanni, E. Fattibene, and T. Ferrari. “WMSMonitor: A monitoring tool for workload and job lifecycle in Grids,” In the 9th IEEE/ACM International Conference on Grid Computing (Grid 2008), Tsukuba, Japan, pp. 209-216, 2008.

[13] Wu-Chun Chung and Ruay-Shiung Chang, “A new model for resource monitoring in Grid computing,” Future Generation Computer Systems, 25(1): pp. 1–7, 2009.

[14] R. Sundaresan, Tahsin Kurcy, Mario Lauriaz, Srinivasan Parthasarathyz and Joel Saltz, “Adaptive polling of Grid resource monitors using a slacker coherence model,” In Proc. 12th IEEE International Symposium on High Performance Distributed Computing, pp. 260–269, June 2003.

[15] R. Sundaresan, T. Kurc, M. Lauria, S. Parthasarathy, and J. Saltz, “A slacker coherence protocol for Pullbased monitoring of on-line data sources,” In Proc. CCGrid 2003 Conference, May 2003.

[16] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. J. Shenoy. “Adaptive Push-Pull: disseminating dynamic web data,” In World Wide Web, pp. 265–274, 2001.

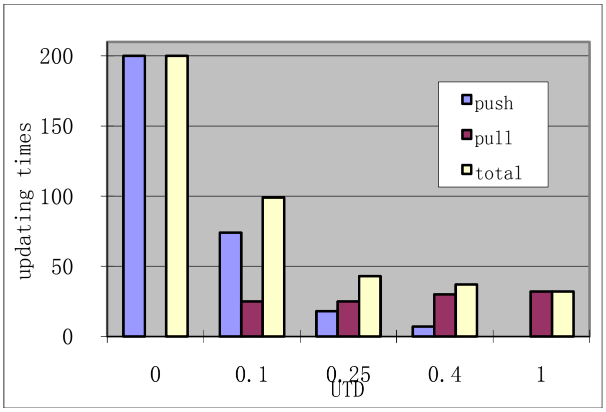

Figure 4. Updating times of P&P model at different UTDs (2nd group)

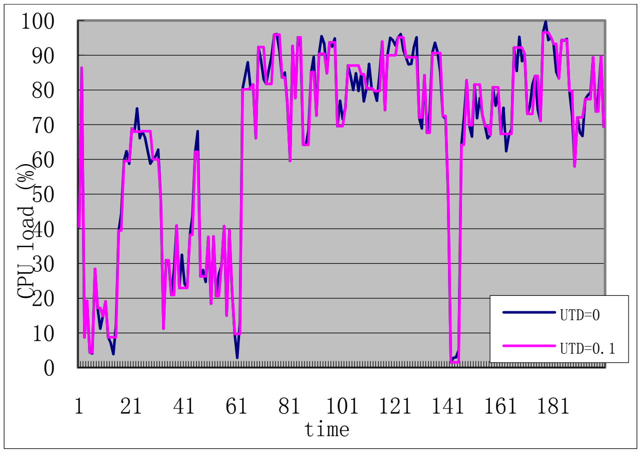

Figure 5. Result of P&P model (small)

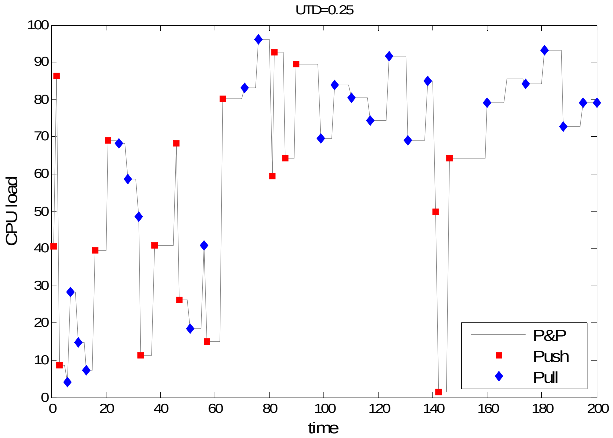

Figure 6. Detail operations of P&P model (moderate, UTD=0.25)

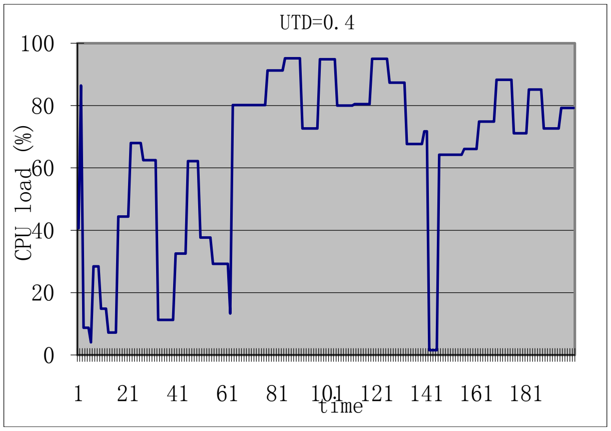

Figure 7. Result of P&P model (large)

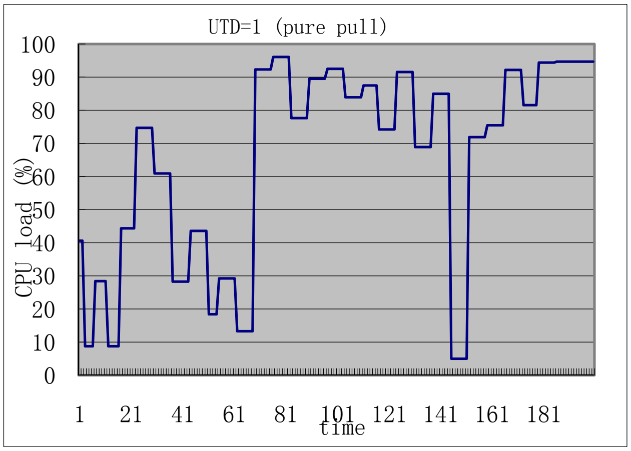

Figure 8. Result of P&P model (pure Pull)