

# Adaptive Polling of Grid Resource Monitors using a Slacker Coherence Model \*

R. Sundaresan<sup>‡</sup>, M. Lauria<sup>‡</sup>, T. Kurc<sup>†</sup>, S. Parthasarathy<sup>‡</sup>, and Joel Saltz<sup>†</sup>

<sup>‡</sup> Dept. of Computer and Information  
Science

The Ohio State University  
Columbus, OH, 43210

{sundarer, lauria, srini}@cis.ohio-state.edu

{kurc-1, saltz-1}@medctr.osu.edu

<sup>†</sup> Dept. of Biomedical Informatics

The Ohio State University  
Columbus, OH, 43210

## Abstract

*As Data and Computational Grids grow in size and complexity, the crucial task of identifying, monitoring and utilizing available resources in an efficient manner is becoming increasingly difficult. The design of monitoring systems that are scalable both in the number of sources being monitored and in the number of clients served is a challenging issue. In this paper we investigate the trade-offs of different polling strategies that can be used to monitor resource availability on machines in a distributed environment. We show how adaptive polling protocols can substantially increase scalability with a less than proportional loss of precision, and how these protocols can be personalized for different types of resource usage patterns.*

## 1 Introduction

Computational and Data Grids, the new wave of distributed computing, provide unprecedented opportunities to problems of large scale. In addition to being diverse in their capacity, resources in a Grid environment can be shared by multiple applications, and their availability changes dynamically. Given the nature of the Grid, resource availability needs to be dynamically monitored and the relevant information needs to

be made available to clients in a timely and efficient way [11, 3]. Designing a monitoring system for Grid resources is a challenging problem. First, resources have heterogeneous interfaces. Some resources behave as event generators, some need to be queried to obtain availability information. Second, the number of resources to be monitored is potentially quite large. With more and more institutions that are making their resources available through Grid infrastructures, the monitoring system should be able to monitor thousands of resources. Third, monitored data should be made available to a large number of clients. Hence, the scalability of a monitoring system, both in terms of the number of resources and the number of clients, is important to achieve effective utilization of the Grid environment.

In this paper we study the problem of resource monitoring in a dynamic wide-area context using pull- and push-based models of data acquisition. Since most information servers adopt a pull-based model, the main focus of the paper is on polling techniques. We use a simulated sensor of available memory as a representative example of an information server, and we analyze the performance of both active (i.e., push-based) and passive (i.e., polled) sensors. In the case of passive sensors, we propose different polling strategies that attempt to maximize the freshness of data while minimizing the number of polls, in order to improve scalability. Some of the proposed polling strategies are based on estimation techniques that try to predict the time of the next update to the information resource

---

\*This work was supported in part by an Ameritech Faculty Fellowship grant, the National Science Foundation under Grants #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #ACI-0203846, #ACI-0130437, #ACI-9982087, Lawrence Livermore National Laboratory under Grant #B517095 (UC Subcontract #10184497), and Ohio Board of Regents BRTTC #BRTT02-0003.

(memory usage in this case), using historical information. In this paper we address the problem of predicting the time at which an update will happen. In this respect, our aim is orthogonal and complementary to approaches for forecasting resource utilization, as implemented in tools like the Network Weather Service [11]. An in-depth performance analysis of both push- and pull-based models is carried out. In order to measure performance we suggest relevant metrics consistent with the objectives of a typical information monitoring system. For each of the the new models we propose, we study the influence of several design parameters.

## 2 Related Work

The Network Weather Service (NWS) [11, 10], is a distributed tool that measures and forecasts the available resources in a given set of nodes. It consists of sensors running on target nodes, which periodically measure network and host based resource availability. Ganglia (<http://ganglia.sourceforge.net>) is a system that closely resembles the active sensor architecture described in this paper. It consists of sensors that monitor the local node for changes and multicast any change to registered listeners. Ganglia does not deal with forecasting future availability. The Global Grid Forum (GGF) is also working on a Grid Monitoring Architecture (GMA) as a part of its Performance Working Group [8]. Several related approaches [9, 1] have looked at agent-based and sensor-based approaches for monitoring resources and integration with existing Grid infrastructure.

The InterAct framework [5], supports efficient data structure sharing with client-controlled coherence for interactive and distributed client-server applications. The framework allows the user to control the degree of coherency of the cached copy. It also supports relaxed coherence models including temporal, diff-based, and delta coherence. Shah et al. [6] propose techniques for delivery of time varying data from data sources to a set of cooperating repositories. They focus on strategies for cooperation among repositories for pushing the data updates to reduce communication and computation overheads. Ninan et al. [4] present an approach for maintaining cache consistency in multiple proxy environments. They propose a *cooperative consistency* model and *cooperative leases* mechanism to

support it. Their work targets content distribution networks where servers push data to proxies, which sit between servers and clients. Deolasee et al. [2] examine methods that combine pull- and push-based models. They present approaches that adaptively choose between pull- and push-based models or simultaneously apply both models. In the latter case, the proxy mainly is responsible to pull updated data from the server, while the server may push additional updates. Zhuge et al. [12] present algorithms for incremental maintenance of views at a data warehouse when the view spans multiple data sources while minimizing costs on query execution. Their approach involves the use of *monitors* at the data sources. A monitor identifies updates of interest and notifies the corresponding data warehouse.

## 3 System Design

The monitoring system consists of three main components. We refer to the machines or systems to be monitored as *Sources*. *Clients* are the ultimate consumers of monitored data. A client may want to monitor changes in resources at multiple sources, thus creating a view of the environment. A view corresponds to a set of sources and the type of information to be monitored at each source. A *Monitor Proxy* is placed between clients and sources. As an example based on familiar Grid tools, a NWS information server could be a source, a global metadata directory service such as the Monitoring and Discovery Service (MDS) in the Globus toolkit<sup>1</sup> would be a monitor proxy, and applications and middleware querying the MDS would represent the clients.

In this section we describe the design of the *monitor proxy* (or proxy) and present different algorithms employed by the monitor proxy to poll sources. We also describe the different types of sensors and the mechanism they use to communicate with the monitor proxy.

**Monitor Proxy:** A *Monitor Proxy* behaves as an intermediate server between clients and sources. Thus, we use proxy and server interchangeably in the rest of the paper. Clients monitor sources through the monitor proxy. Having a proxy has several advantages. First, without a proxy, the load of the source will increase as the number of clients increases. Multiple clients may want to monitor overlapping sets of sources and

<sup>1</sup><http://www.globus.org/mds>

create similar views. By consolidating views at the proxy, data duplication is minimized and the load seen by sources is reduced. Second, a proxy allows clients to create complex views that can span multiple sources and request different types of information be monitored. In order to serve clients based on these views, the proxy should monitor the sources on behalf of the client, should retrieve the monitored information from sources and store them locally. The main aim of the proxy therefore is to ensure that it maintains a coherent copy of the data at the sensor to server the clients.

There are two basic ways by which a proxy can synchronize with a source. In the push model, when the availability of a resource changes at a source, it is *pushed* by the source to the proxy. In the pull model, it is the responsibility of the proxy to poll the sources for resource status updates. We next describe the three types of sensors that we studied.

**Types of Sensors:** Here we propose three different mechanisms to monitor host based resources on nodes. First, in the *active sensor* model, the sensors run on all the nodes in the system. The sensors periodically measure the resource availability at the node and multicast the information to registered servers. These active sensors need not multicast the information periodically. Rather they multicast the information when there is a significant change to the value that was last broadcast, wherein the definition of significance is something that each server that registers with the active sensor will have to specify. The active sensors are unmatched in terms of data freshness as we shall show in Section 4. However in the Grid scenario the use of active sensors is complicated by a number of issues. A main point of concern is the possibility of security holes. Other practical issues include automatic startup, system administration overhead, portability to different architectures. Second, the *passive sensor* model consists of sensors that are run on all nodes and periodically probe the system to collect resource availability details. The server polls the sensors when it requires information. Third, the *ssh sensor* model opens a secure shell connection to the target machine, execute the necessary system commands to measure the resource availability, and close the connection. This method has the advantage that it does not require sensor be installed on the target machine and allows for target machines to be monitored irrespective of their architecture. The

disadvantage in the ssh scheme is the amount of overhead that is involved in opening a ssh connection every time the target machine is polled.

The polling model we study is called slacker coherency. An analysis of this scheme was reported in a previous study [7]. Here we describe the application of the slacker coherency scheme to the monitoring of memory availability. In this scheme, the server maintains the history of recent measurements. The server, using this history, estimates the best time to poll the sensor when there is a high likelihood of a significant change in resource availability.

**Objective Functions:** The main aim of the proxy is to maintain a local view of the availability of resources on remote machines. The view should be coherent with the actual availability of remote resources. In order to achieve this, the proxy needs to poll remote sensors frequently enough to maintain a coherent view most of the time. However, polling too frequently results in increased resource usage of the proxy node, heavier intrusion at the source, and increased network utilization. Based on these observations, we consider the following objective functions:

1. The number of times a sensor is polled. The ideal value for the number of polls is equal to the number of updates at the sensor during that period. Obviously we need to minimize the number of polls that were useless because no change occurred since the last poll.
2. The Out-of-Sync period. This is defined as the percentage of time when the proxy is not coherent with the value at a sensor. Minimizing the number of useless polls achieves almost the same result as minimizing the out-of-sync period. However, minimizing useless polls ideally will produce a poll once for each update at the server in the ideal case. Assume the update happens at the rate of once every 60 seconds. The server will still minimize the number of useless polls if it polls 59 seconds after the update is done. But this does not minimize the out-of-sync period which would be close to 100% in this case.
3. The Average Age of a View. This is the expected value of how much time has elapsed since the value that is currently maintained by the proxy has changed at the sensor. This metric gives a

good measure of how "old" the view is expected to be any point of time.

4. **Weighted Average of the Error.** When monitoring resources, one important issue is how much difference there is between the value as reported by the proxy and the actual value at the target node at that time. This gives a measure of the confidence level of the value reported by the proxy. Hence an important objective function is the minimum, maximum and average error in the value reported by the proxy. However these metrics do not account for how long a particular value was reported wrong. It may be acceptable for some clients if a high error value is reported for a very short amount of time. Hence we also calculate the average of the error in the reported value weighted by the fraction of time the reported value was wrong.

**Polling Algorithms:** The *Monitor Proxy* makes use of the time series information obtained from the sensors to estimate the time of the next significant update. For the proxy to make an accurate estimate, it will require the information about all significant updates that happened at the remote node. However, it is quite possible for the server to miss some updates due to the slacker polling technique which leaves the time series incomplete and results in inaccurate estimates.

It is desirable that the server maintains a list of all the recent updates. Hence the sensor in addition to sending the latest measurement also sends a short history of the previous updates. This allows the server to maintain a more accurate time series for estimation. This scheme works only in the case of passive sensors. The ssh-based scheme cannot take advantage of this method because there is no state that is maintained in the target nodes as ssh connections are opened on a per-poll basis. Hence the ssh-based scheme will give only the last update value to the proxy.

In addition to the time-series based estimation for slacker polling, we explore another polling strategy called *Adaptive Polling*. Here the proxy polls the sensor after a time interval  $t$ . If the poll results in a value that is significantly different from the value that is currently maintained by the server, the server decreases the time to next poll to  $t * (1 - d)$ , where  $d$  is the damping factor and hence increase the polling rate. How-

ever if the poll did not result in any significant change in value, the proxy increases the time to next poll to  $t * (1 + d)$  and hence decrease the polling rate. To avoid runaway events in extreme situations, we define upper and lower limits to the polling rate.

The memory usage characteristics of applications typically have a range of values that is required for its normal running. Any node that has less than the minimum requirement cannot be used to run the application. Also, if any node has more memory than the maximum of the requirement range, the application only needs to know that the node has more memory than its requirement and does not require the actual value of available resource. We can devise a polling strategy that will estimate the time when the value is likely to shift from one of the three states. We plan to look at various estimation technique for this strategy in a future work.

### 3.1 Estimation Mechanisms

The server employs different types of statistical estimators to analyze the time series of memory usage updates and estimate the best time to poll the sensor next.

**Moving Average.** The Moving Average estimator is a very effective and simple time series estimation tool. The server maintains a window over the timestamps of significant measurements from the sensor. The Moving Average Estimator does an average of the time interval between two consecutive updates. At any time instance  $t$ , the estimation of the time to next poll  $E(t)$  is given by

$$E(t) = \frac{1}{N} \sum_{i=1}^N (ts_i - ts_{i-1})$$

where  $ts_i$  is the  $i^{th}$  update time-stamp. The effectiveness of this scheme depends on the window size that is used. Larger window sizes are more suitable for usage patterns that have a gradual or long-term fluctuation patterns. However, usage patterns that exhibit sharp non-periodic changes will require a smaller window time for faster response times.

**Exponential Weighted Moving Average.** The Exponential Weighted Moving Average (EWMA) operates on the principle of assigning exponentially lower weights to older values of the time series. There are different types of EWMA estimators. The Single EWMA estimator is effective in estimating traces which do not exhibit well defined trends or patterns. At any time instance  $t$ , the estimation of the time to next poll  $E(t)$  is given by

$$E(t) = \alpha(ts_t - ts_{t-1}) + (1 - \alpha)E(t - 1)$$

where  $(ts_t - ts_{t-1})$  represents the last observed time between two consecutive significant updates. The value of  $\alpha$  determines how much weight is given to the last observed value and how much to the previous estimates. Smaller the value of  $\alpha$  is, the more the weight is given to the older values in the series. Larger values of  $\alpha$  result in more weight being given to more recent observations.

Choosing the right value for  $\alpha$  requires some amount of knowledge of the usage patterns on the series. This implies that the value of  $\alpha$  should be chosen dynamically using the observed time series characteristics. Every time the server estimates the next time to poll, it uses the current time series of update timestamps and simulates the estimation on that time series for different values of  $\alpha$ ; it then chooses that value of  $\alpha$  that results in the least error. This mechanism ensures that the value of  $\alpha$  adjusts to the usage pattern of the specific sensor being monitored.

Single EWMA is a powerful mechanism for estimation on time series that do not exhibit any type of regular trend. There are higher order EWMA estimators that can effectively capture such behavior. We have investigated the use of Double EWMA which is a second order estimator. At any time instance  $t$ , the estimation of the time to next poll  $E(t)$  is given by

$$\begin{aligned} F(t) &= \alpha(ts_t - ts_{t-1}) + \\ &\quad (1 - \alpha)(F(t - 1) + b(t - 1)) \\ b(t) &= \gamma(F(t) - F(t - 1)) + (1 - \gamma)b(t - 1) \\ E(t) &= F(t) + b(t) \end{aligned}$$

Here,  $0 \leq \alpha \leq 1$  and  $0 \leq \gamma \leq 1$ . Double EWMA is characterized by two parameters  $\alpha$  and  $\gamma$ . The  $E(t)$

adjusts to the trend of the previous period  $b(t - 1)$ , by adding it to  $F(t - 1)$ . This is effective in capturing trends. We use a dynamic version of Double EWMA wherein the values of  $\alpha$  and  $\gamma$  is calculated dynamically based on the observed time series, using the scheme described for Single EWMA estimation.

**Lagrange Interpolation.** The Lagrange Interpolation is an estimation technique that fits a n-degree polynomial using n+1 points. The polynomial is then used to extrapolate for different future time periods. The time period that extrapolates to a value that is a significant change compared to the last available value is taken as the time interval to the next poll. The time series here consists of the actual values of amount of available memory, whereas in the other estimation mechanisms, it consists of time intervals between consecutive updates.

## 4 Experimental Results

For the experiments, we obtained memory traces by collecting memory availability data on about 50 nodes of two clusters at the Ohio State University and the Ohio Supercomputing Center. The data was collected over a two week period with a periodicity of 60 seconds. We identified three classes of traces and ran experiments on one representative trace from each class. The main classes of traces that we identified based on the memory usage patterns were *i*) NFS servers, *ii*) experimental workstations, and *iii*) user desktop machines. Figure 1 shows the trace of available memory on the nodes drawn from the three classes of usage patterns. The NFS server exhibits a good amount of fluctuation in memory availability but the magnitude of changes is typically low. The experimental workstations are typically used to run experiments on various data intensive applications, which explains the spiked behavior of the trace. The desktop nodes have a rather flat behavior with only occasional changes. The experiments discussed in this section had the sensors configured to report data from these traces rather than from actual measurements. This ensures repeatability and simplified analysis of the results. Experiments that required a larger number of sensors had more than one of these simulated sensors running on the same node. The experimental testbed consisted of Pentium III single-

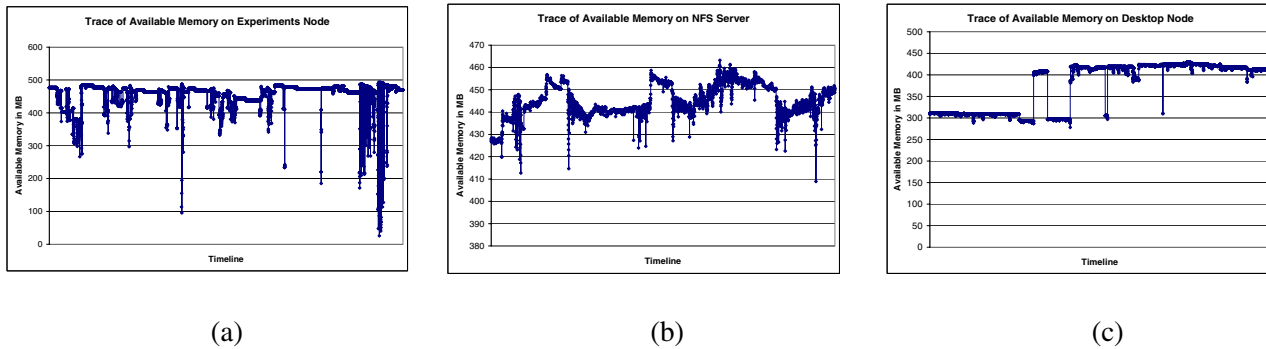


Figure 1. Traces of available memory: (a) Experimental Node, (b) NFS Server, (c) Desktop Node.

processor nodes running Linux 2.4 with 1GB of memory and interconnected using Fast Ethernet. The experiments were conducted on traces collected from these machines on the three different types of monitoring systems, namely active, passive and ssh based sensors.

#### 4.1 Basic push- and pull-based models

**Active Sensors.** The most popular approach to resource monitoring is based on active sensors. The sensors autonomously send information on significant changes to the listening proxies/clients, following a push model. This scheme is very scalable with respect to increasing number of proxies and sensors. An experiment with 10 proxies and 400 active sensors running on multiple clusters resulted in an out-of-sync value of 0.01%. We also conducted an experiment with 400 proxies and 10 sensors and this results in an average out-of-sync value of 0.175% across all the proxies. While this approach is clearly highly scalable and performs quite well in terms of data freshness, it is also the one with the largest administrative overhead. In particular, it requires that the sensors maintain a list of clients, and a protocol to update such lists. Full access to the nodes is also required in order to install the sensors.

**Ssh-based scheme.** The ssh based scheme is the opposite end of the spectrum in that it makes the least amount of assumptions on what is installed on the nodes. Therefore is best suited to those situations where there are a large number of nodes to be monitored and/or there are administrative restrictions that make installing active sensors on those machines infeasible. However ssh based sensors incur a connection management overhead at every poll. Typical

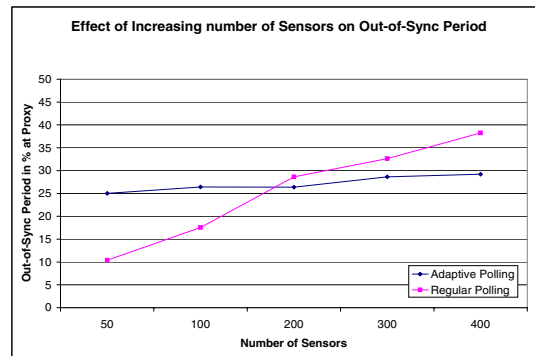


Figure 2. Comparison of the scalability of the Regular Polling and Adaptive Polling.

monitoring systems poll the sensors at regular intervals. To quantify the advantage of adaptive polling we performed two set of measurements, one with a constant polling interval of 60 seconds, the other using the adaptive polling with a damping factor of 25%. Figure 2 shows the average fraction of time when the proxy is out of sync with the sensors, for increasing number of sensors. In this experiment, we used a mixture of the three types of memory traces. For the case with  $N$  sensors,  $N/3$  sensors used traces from experimental nodes,  $N/3$  sensors used traces from NFS server, and the remaining sensors used traces from desktop nodes. The regular polling case has a lower scalability compared to the adaptive polling case for a sufficiently large number of sensors ( $> 200$ ), beyond which the advantage of the adaptive polling increases with the number of sensors polled.

#### 4.2 Advanced Pull-based Models.

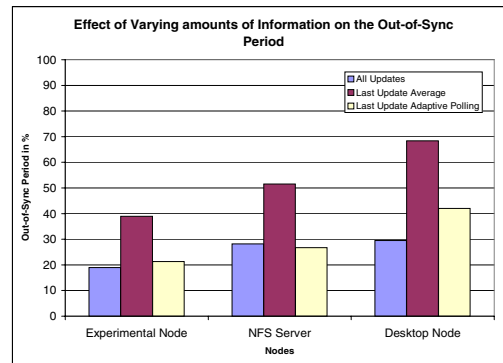
These models assume the use of passive sensors, but rely on estimation of future event occurrences to adap-

tively adjust polling frequency. We investigate the following design parameters: i) the amount of historical information available for estimation, ii) performance of the estimation techniques using different usage patterns, iii) the effect of tuning parameters such as window size and damping factor, iv) varying levels of tolerance.

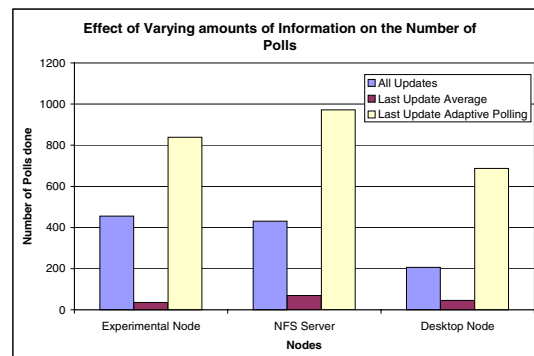
**Effect of Amount of Historical Data.** The proxy relies on the time series information from the sensors to make predictions on the time of the next significant update. The amount of information available at the proxy depends on the kind of sensor. Passive sensors can be designed to send a proxy any amount of historical information on the updates. This allows the proxy to maintain information about all updates, including the updates that were missed. On the other hand, ssh-based sensors do not maintain any remote-side state. Since the amount of information available at the proxy for estimation is limited, the only polling scheme available for the ssh-based case is the adaptive polling technique.

To quantify the importance of historical information availability, we perform a comparison between i) Moving Average Estimator when all updates are available, ii) Moving Average Estimator when only the last update is available, and iii) adaptive polling using last update. We use a damping factor of 25% in the case of adaptive polling using last update. The graphs in Figure 3 compares the various objective functions at the server for the three different traces. As expected, the all updates case gives the lowest out-of-sync value as it has the maximum time series information. The last updates case on the other hand has incomplete time series and hence it underestimates the update rate at the sensors. As a result, it performs fewer polls that results in a higher out-of-sync period. The last update adaptive polling case makes use of the information from the last poll to either increase or decrease the polling rate by an amount regulated by a damping factor. Its performance makes it the second best to the all updates case (except for the number of polls performed), and the best choice when historical data is not available (i.e. ssh-based polling).

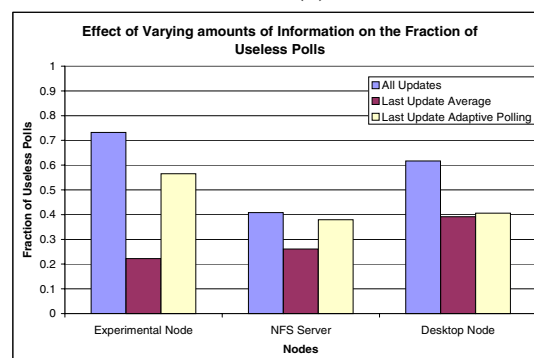
**Effect of Estimation Techniques.** In this experiment we compare the performance of the passive



(a)



(b)



(c)

**Figure 3. Comparison of the three polling schemes. (a) Out-of-Sync Period, (b) Number of Polls, (c) Fraction of Useless Polls.**

polling scheme when using three different statistical estimators - the Moving Average, Lagrange Interpolation and Exponential Weighted Moving Average (EWMA). We analyze the performance of these estimators on different resource usage patterns by using trace data collected from all three types of nodes (NFS server node, workstation node, experimental node). The objective functions used for the comparison are the Out-of-Sync period, the number of polls made to the sensors, the Average Age Value of the sensor and the weighted error in the reported value. Figure 4 shows the effect of different estimators on the objective functions. The baseline is the regular polling case, which gives the lowest average age value but also results in the largest number of polls. We found that the Lagrangian estimator gives the highest average age value, and also the lowest number of polls. Such poor performance is not entirely unexpected, since Lagrangian estimator are known to be most effective in extrapolations over continuous curves, which do not match the kind of patterns exhibited by the memory usage.

Single EWMA is characterized by its parameter  $\alpha$ . A lower value of  $\alpha$  results in larger weight given to the oldest historical information, a higher value of  $\alpha$  implies greater weight for the more recent observations. This trend is evident in Figure 4, where the performance of EWMA( $\alpha = 0.9$ ) is better than that of EWMA( $\alpha = 0.1$ ) for the experimental node. This is because of the rapid shifts in trends and non-periodic changes that occur in the usage patterns and hence for better performance, the adaptivity to sudden changes should be faster. The performance of EWMA( $\alpha = 0.1$ ) being better than that of EWMA( $\alpha = 0.9$ ) for the other two usage patterns can also be explained along similar lines.

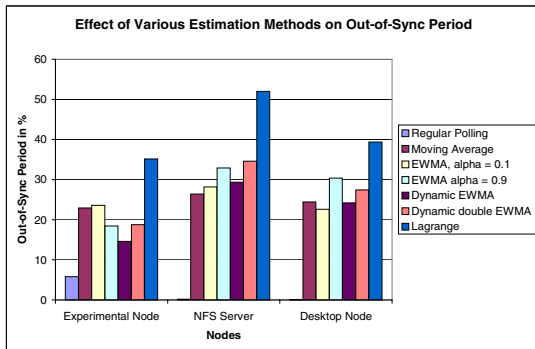
Figure 4 also illustrates the effectiveness of the dynamic EWMA scheme. The value of alpha in this case is dynamically adjusted so that the estimation error is minimized. Double EWMA is a second degree estimator which can characterize seasonality of data change and periodicity patterns. Double EWMA is characterized by the two parameters  $\alpha$  and  $\gamma$ . The Figure 4 shows the results for dynamic Double EWMA( $\alpha, \gamma$ ), where the parameters are determined dynamically to minimize the local error. Though Double EWMA is expected to work well for certain resource usage pat-

terns like bandwidth usage, which have been shown to exhibit temporal periodicity [11], the memory traces do not show any consistent trends. Also, it has been shown that Double EWMA is less tolerant to noise than the single exponential, as the double exponential can be lead into incorrectly identifying outliers in the data as being part of a trend. This explains why Double EWMA does not perform better than Single EWMA on our traces. Finally, Figures 4 also show the weighted average error in the reported value. This metric depends on the age of the data and the actual magnitude of change in value when an update occurs on the sensor. The weighted error value is higher across all estimators for the experimental nodes where the typical change in value is greater than the other two classes of traces which exhibit a more smoother behavior and a smaller magnitude in change.

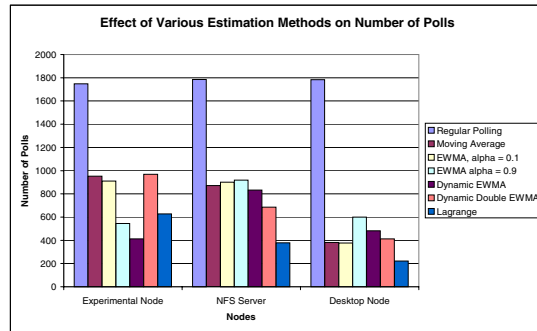
**Sensitivity to Polling Parameter Values.** The performance of the polling techniques often depends on the value of one or more parameters. This set of experiments studies how sensitive these techniques are to the values of polling parameters. A low sensitivity is a desirable and important property because this means the design of the system is robust to tolerances in the determination of the optimal value of the parameters.

For the first test we measured the dependence of the performance of the moving average estimator on the window size. The window size determines the significance attached to older observations in the current estimate. Figure 5(a) illustrates the effect of varying window sizes on the out-of-sync period. Intuitively, one might expect that nodes which exhibit a larger non-periodic rate of change will require a smaller window size for better response times, while nodes which exhibit a smooth fluctuation or infrequent changes will require a larger window size in order to capture this behavior. Figure 5(a) confirms this intuition by showing that the optimal window size is different for each of the three categories of traces analyzed. The trace of experimental nodes requires a window size of 10 and Desktop nodes require a window size of 25 for optimal performance. However the graph also show that within each usage class the curves are reasonably flat, which means the window size value is not critical.

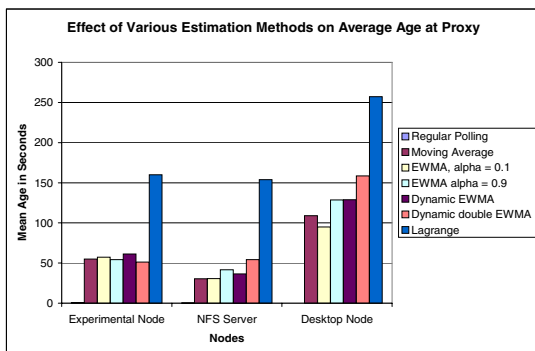
The Adaptive Polling Scheme uses the least amount of information from the sensors to decide the polling



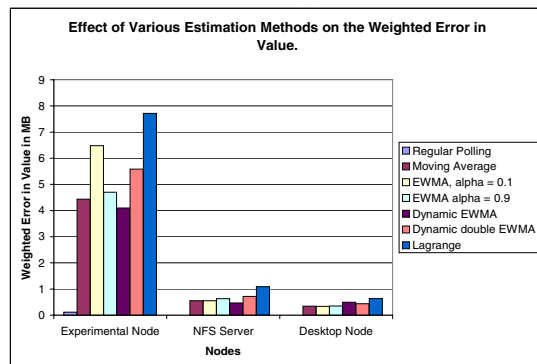
(a)



(b)

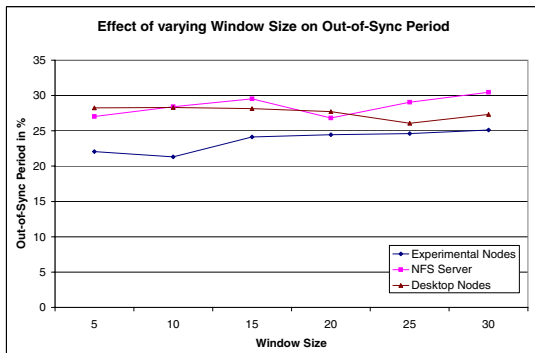


(c)

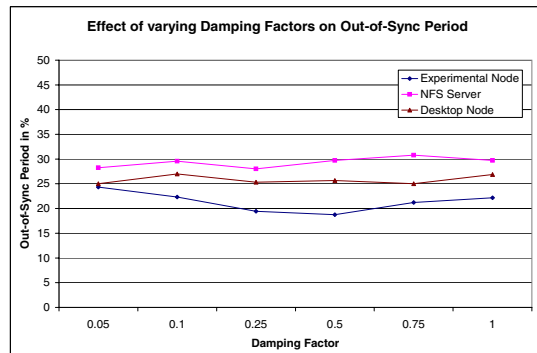


(d)

Figure 4. Comparison of estimation techniques. a) Out-of-Sync Period, (b) Number of Polls, (c) Average Age of Data, (d) Weighted Error in Value



(a)



(b)

Figure 5. Effect of (a) window size and (b) damping factor on the Out-of-Sync period.

rate. The main parameter in this scheme is the damping factor that is used to increase or decrease the polling rate. The higher the damping rate, the faster is the response to change and higher the oscillation. Lower damping rates result in less oscillations and are more suited for smoother trends. Conversely, lower damping rates do not perform well when there are sudden shifts in trends in the usage pattern. Figure 5(b) illustrates the effect of varying damping factors on the Out-of-Sync period. The heavily loaded experimental workstation node show high fluctuation in memory availability and hence has a higher optimal damping rate (50%). However, the NFS server and desktop machines, which exhibit more slow fluctuations, have an optimal damping rate of 25%. We observe that there is a modest dependence of performance on the value of the damping factor.

## 5 Conclusions

In this paper we propose a number of novel pull-based models of information monitoring, and we present a performance analysis of these models. We identify the main components of their design, and we measure the effect of each of these component on the overall performance of the models. Based on the slacker coherence concept, our polling models use historical data to estimate the time of the next information update. We quantify the relevance of the amount of available historical data using performance metrics such as freshness and number of polls. One of the crucial components of these models is the estimator; we show that different estimators performs best in different situations depending on the specific pattern of resource usage updates. Finally, we observed that the performance of a polling scheme is not particularly sensitive to the values of estimator parameters and other model parameters. We also show how these parameters can be adaptively tuned for maximum freshness.

## References

[1] DBLP, <http://dblp.uni-trier.de>. *An Infrastructure for Monitoring and Management in Computational Grids*, volume 1915 of *Lecture Notes in Computer Science*. Springer, may 2000.

- [2] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. J. Shenoy. Adaptive push-pull: disseminating dynamic web data. In *World Wide Web*, pages 265–274, 2001.
- [3] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, 5-8 Aug. 1997.
- [4] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamritham, and R. Tewari. Scalable consistency maintenance for content distribution networks. Technical report, University of Massachusetts, Amherst, 2001.
- [5] S. Parthasarathy and S. Dwarkadas. Shared state for distributed interactive data mining applications. *The International Journal on Distributed and Parallel Databases*, March 2002. An earlier version appeared in the SIAM international conference on Data Mining 2001.
- [6] S. Shah, K. Ramamritham, and P. Shenoy. Maintaining coherency of dynamic data in cooperating repositories. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB 2002)*, Aug. 2002.
- [7] R. Sundaresan, T. Kurc, M. Lauria, S. Parthasarathy, and J. Saltz. A slacker coherence protocol for pull-based monitoring of on-line data sources. In *Proceedings of the CCGrid 2003 Conference*, May 2003.
- [8] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany. A grid monitoring architecture, 2002.
- [9] B. Tierney, B. Crowley, D. Gunter, M. Holding, J. Lee, and M. Thompson. A monitoring sensor management system for grid environments. In *HPDC*, pages 97–104, 2000.
- [10] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.
- [11] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.
- [12] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. Consistency algorithms for multi-source warehouse view maintenance. *Journal of Distributed and Parallel Databases*, 6(1):7–40, 1998.