# Dynamic Surge Protection: An Approach to Handling Unexpected Workload Surges With Resource Actions That Have Lead Times

E. Lassettre, D. W. Coleman, Y. Diao, S. Froehlich, J. L. Hellerstein, L. Hsiung
T. Mummert, M. Raghavachari, G. Parker, L. Russell, M. Surendra, V. Tseng, N. Wadia, P. Ye
IBM Corporation

## ABSTRACT

Today's information technology departments have widely varying demands for resources due to unexpected surges in subscriber demands (e.g., a large response to a product promotion). Further complicating matters is that many resource actions done in response to surges (e.g., provisioning or de-provisioning an application server) have substantial delays (lead times) between initiating the resource action and its taking effect. This paper describes dynamic surge protection, an approach to handling unexpected workload surges in systems that have lead times for resource actions. Dynamic surge protection incorporates three technologies: adaptive short-term forecasting, on-line capacity planning, and configuration management. The paper includes empirical results from evaluations done on a research testbed, including favorable comparisons with a threshold-based heuristic. The results from an extended test also show that service objectives can be maintained cost-effectively.

## General Terms

Adaptive forecasting, time series models, lead times, subscriber surges

## 1. INTRODUCTION

Today's information technology (IT) departments are besieged with uncertainty. New applications are deployed, but their resource demands are unknown. Traditionally, these situations have been addressed by over-provisioning IT resources and/or manual resource re-allocations. Unfortunately, these approaches are costly – the former in terms of equipment, license, etc and the latter in terms of expert operators. Furthermore, many resource actions involve **lead times**, such as server provisioning, which introduces delays between action initiation and effect. We present an approach to moderate the effect of unexpected workload surges so as to preserve a service level objective (SLO) in a cost effective way while taking resource actions with lead times.

Examples of subscriber overloads abound. On September 11, 2001, the CNN web site was overwhelmed by traffic that doubled every 7 min to a peak of 20 $\times$ normal volume [7]. The Victoria's Secret web site had a similar experience as a result of an advertising campaign during the 1999 Super Bowl [9]. Others have noted that "sites such as Encyclopaedia Britannica, egg.com, and H&R Block have suffered massive overload from subscribers" [16].

Many systems (e.g., [14], [10], [18], [12]) have been developed to adapt to changes in workload. Sun's N1 [14] and HP's Utility Data Center (UDC) [10] initiatives provide convenient ways for operators to move resources between applications. Underlying this is the ability to describe logical application topologies and the physical configuration. However, no automation is provided to determine when to move resources between applications. The MVS workload manager incorporates algorithms for adjusting resource allocations to achieve SLOs [1]. These algorithms assume that actions take effect immediately (e.g., changing CPU priorities) and so do not address actions with substantial lead times. Another relevant technology uses load forecasting together with performance estimation to balance file allocation across a network attached storage system to meet a response time objective under varying load [8]. The ThinkProvision technology of Think Dynamics provides automation for model-based optimization of dynamic resource provisioning [15], and the DynamicIT technology of ProvisionSoft uses long-term forecasts to anticipate time-of-day effects [13]. An important difference between the the current work and approaches that use long-term forecasting is that the latter is appropriate for workloads with periodic variation. Conversely, it is not suited for unexpected workload surges, which have no regular pattern.

As reported in [3], there is considerable benefit in rapidly adjusting resource allocations to handle variable workload. This is a challenge if reallocating resources has substantial lead times. One approach is to reduce the lead times, but this may be of limited utility. A second incorporates long-term forecasting to predict workloads and initiate resource actions sufficiently early to accomodate lead times. However, long-term forecasters can require substantial data to learn patterns and thus work poorly for unexpected surges.

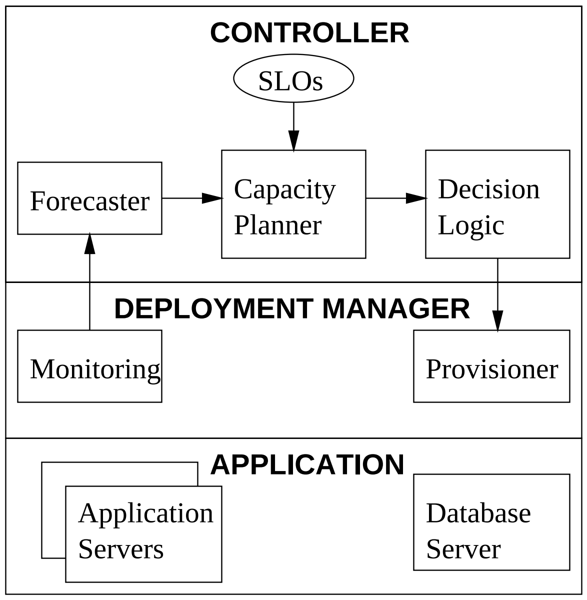

**Figure 1: Architectural layers for dynamic surge protection. The arrows show the control and data flow.**

In line with IBM's autonomic computing initiative for self-managing systems [6], we describe a system that has self-configuring characteristics. Our approach, which we refer to as **dynamic surge protection**, employs three technologies: adaptive short-term forecasting, on-line capacity planning, and configuration management. The forecasting approach we use is designed to be responsive to rapid changes, yet robust. On-line capacity planning determines resources needed to preserve service levels in a cost effective way (e.g., releasing resources when not needed). Configuration management provides the means for adjusting resources, such as by tuning, provisioning, and/or workload throttling to adapt to the rise and decay of unexpected surges.

## 2. ARCHITECTURE AND ALGORITHMS

The system is structured into three layers as shown in Figure 1. The Application layer provides the business function. We use a two-tier web application with one or more application servers and a database server. In general, we require that the application tier scale horizontally (i.e. resources can be added/removed without system shutdown).

The Deployment Manager provides a generic interface for monitoring and configuring the application layer. In this work, transaction rates and response times were monitored. Configuration management was focused on provisioning (addition/removal of application servers). The architecture assumes that for each resource type there is a provisioning function that manages a resource pool that can be shared among applications. Longer-term configuration management will also address configuration parameter (e.g., buffer pool sizes) and admission control adjustment.

The Controller monitors the application layer state and initiats appropriate actions if a SLO violation is anticipated or if the SLO can be satisfied in a more cost-effective way. Figure 1 also depicts the control and data flow for dynamic surge protection. Workload data from monitoring are input to the forecaster, which predicts future workload. The capacity planner takes as input the prediction and SLO to determine the resource requirements (e.g. number of servers). The decision logic manages the information flow and determines the resource adjustments (by comparing to the deployment state data). These adjustments are effected by the provisioner. Note that this flow is straight through (not iterative) since the required inputs are known at each step. The system operates based on six time intervals.

1. A measurement interval $M$ chosen based on overhead and noise vs responsiveness concerns.
2. The control interval $P$ ($P \geq M$), between executions of the control flow in Figure 1.
3. The lead time $S$, between initiating a resource action (e.g. add/remove a server) and its completion.
4. The prediction horizon $H$, should be $\geq S$. The tradeoff is that a longer $H$ increases variability ($\sim \sqrt{H}$).
5. The overflow interval $O$, between situations when resources need to be adjusted to avoid SLO violations. The control engine can respond if $O \geq P + S$.
6. The underflow interval ($U$) between when a SLO can be met with fewer resources and when the extra are removed. The control engine can handle $U \geq P + S$.

We conclude that $H \sim P + S$ is reasonable. In our prototype (more details below) $S$ is $\sim$ 30 sec, and we found that $M = P = 10$ sec worked well. Incorporating some "safety margin", we use $H = 60$ sec.

The capacity planner provides both performance estimates and the ability to determine resource requirements given the workload and the SLO. We used an IBM internal tool [5] that has been widely used in service engagements over the last two years. This tool uses analytic queueing approaches to estimate performance and capacity of a web deployment based on workload patterns (predefined or modeled by the user), performance objectives, together with hardware and software specifications. Other approaches to online capacity estimation are described in the literature [11].

We use a short term forecaster (or predictor) since our emphasis is on managing unexpected surges. The short term forecaster needs to provide useful information about the leading edge of the surge. Key to this is the ability to learn quickly, which is achieved by using short history data, and not relying on extensive training. In this work, the short term forecaster typically used 2 min (12 points) of history to predict 1 min into the future. The short term predictor is almost "memoryless" – it does not retain knowledge of past surges. This is important since the occurence of one unexpected surge (or busy period) is not assumed to provide information about when to expect the next surge. Analysis of Web traffic [4] suggests that busy periods are not strongly auto-correlated with idle periods (time between busy periods). Within a busy period there can be reasonable auto-correlation [17], hence non-seasonal ARIMA (autoregressive, integrated, moving average) models [2] are effective.

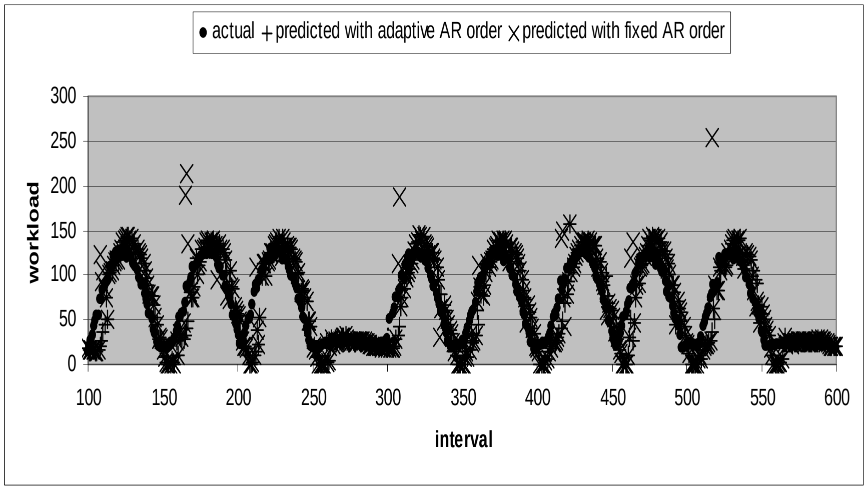

**Figure 2: Short term forecaster showing the comparison between an actual time series and 1 min (6 interval) ahead predicted values. Note that spurious predictions (e.g. high flyers) occasionally resulting from the fixed order model are greatly minimized with the adaptive order model.**

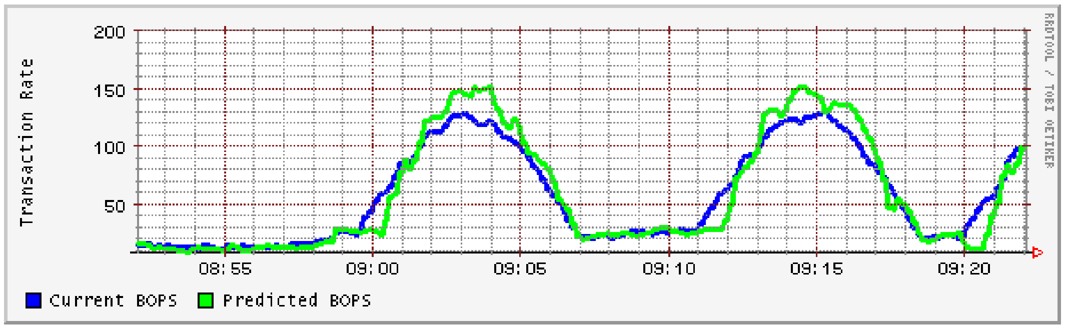

**Figure 3: Actual and 1 min ahead predicted values of the workload.**

An undesirable effect of using a short history for prediction is increased inaccuracy. Thus, we dynamically adjust the model order (the number of, AR, or autoregressive terms) based on the stability of the estimates. The latter is determined by checking if the poles of the transfer function lie within the unit circle (a requirement for stability in discrete time systems). The value of the adaptive model order is shown in Figure 2. While most of the predictions from a fixed AR order effectively track the actual time series, there are occasions when spurious predictions occur. These spurious predictions, which drive unnecessary control actions, are significantly minimized by adapting the model order.

We note that the short term forecasting we discuss above has also been used in conjunction with a long term forecaster [8] which is effective at capturing cyclic variation. Currently, the training data of the long term forecaster would include the unexpected surges since doing so simplifies data management. However, this approach can increase forecast variability and result in predicting phantom surges.

## 3. RESULTS

Figure 3-Figure 5 show the results of experiments conducted on a research testbed. The testbed consists of a workload driver, multiple application servers running IBM's Websphere Application Server (WAS) v5.0, a single database server running IBM's DB2 v8.1, and a manager machine that incorporates code for the Controller and Deployment Manager. The provisioner leverages WAS 5.0's cellular cluster capability and uses its startServer/stopServer commands to add/remove servers.

The application that we deployed on our two tier system simulates the supply chain management of a manufacturing company. Briefly, the application processes injected order transactions and kicks off manufacturing transactions internally. The total transaction (business operations) rate is the sum of the actual order and manufacturing rates, and is normally (no transaction rollbacks) $\sim 1.75\times$ order injection rate. The workload driver can vary the order injection rate, where the inter-arrival time is exponentially distributed. Large workload surges (e.g. to mimic an influx of new users) are randomly triggered, and the Controller's SLO is to keep response time below 2 sec.

Figure 3 plots the actual (blue or darker line) and predicted 1 min into the future (lighter or green line) business operations per second (BOPS), the metric used to character-

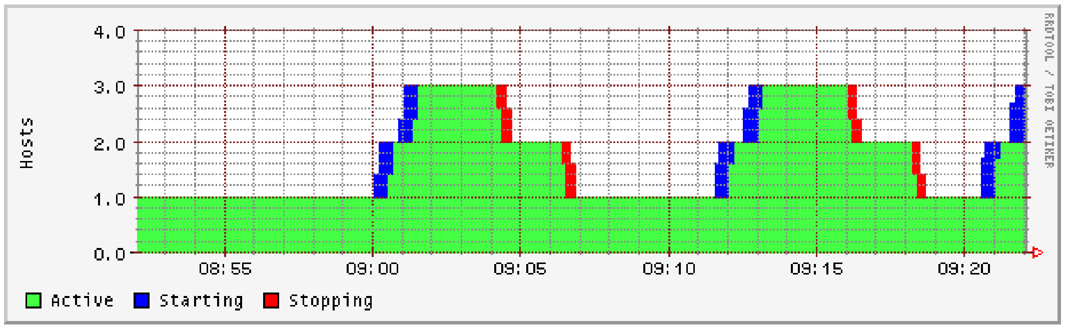

**Figure 4: State and number of application servers.**

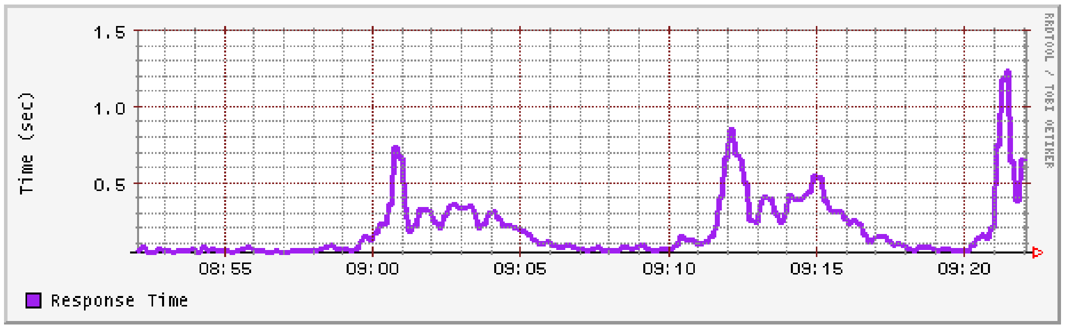

**Figure 5: Response time of application.**

ize workload. During the non-surge or normal periods (e.g. 8:52–8:59), BOPS has little variation. When a surge begins (e.g., 8:59, 9:11), BOPS increase rapidly to a peak of 120, ($\sim 6 \times$ normal). Not surprisingly, predicted BOPS for the first 60 sec are significantly below actual BOPS since the prediction is based on non-surge data. Within a few control intervals into the surge, the prediction accuracy improves considerably. Note that prediction accuracy during the surge (especially in regions of highest curvature) is not nearly as good as during the non-surge periods. It is important to emphasize that while the forecaster cannot predict the occurence of the surge, it can quickly recognize the workload trend change, and thus provide information on the anticipated evolution of the surge. Also, we reiterate that the forecaster essentially does not retain any knowledge about past surges since the history used for prediction ($\sim$ 2 min) is less than the surge duration.

Figure 4 shows the changes of the state and the number of application servers in response to the actual and predicted workload. When a rapid increase in load is detected just after 9:00, a server is added (the leading dark or blue section). This is about 40 sec before a server would have been added had the decision been based on the actual BBOPS vs predicted BOPS. A second server is added at 9:01 as the short-term forecaster anticipates the progression of the surge. As the surge subsides around 9:04, servers are released (the trailing red or less dark section). Note that while the decisions about adding resources is made quite aggressively, the removal of servers is more gradual, due to damping introduced by the Controller decision logic to minimize repeated add/remove server operations.

Figure 5 depicts the effect of these actions on response times. We see an initial bump in response time around 9:00. In part, this is due to the increased load that cannot be handled until the server has completed its startup phase. But there is also some delay introduced by the action of adding a server. Nonetheless, the SLO is not violated We note that because of the stochastics of the system, the response times and control actions responding the different surges are not identical.

Although in Figure 3 we show one type of surge, we have tested the system with different types of surges (e.g. multi-peaked, different shapes, etc). For instance the surge shown in Figure 6 has exponential rise/decay with a plateau in between vs the half sine wave surge in Figure 3). In an effort to test the responsiveness of the system, we experimented with surges (peak = $8 \times$ base) that have different exponential rise rates (characterized by doubling time). While the observations are dependent on the specifics of the deployment, we find that the system can maintain the SLO when the doubling time is 60 sec, and even at 45 sec, but has trouble keeping up when it is 30 sec. We have also successfully tested it with larger surges ($20 \times$ base @ 60 sec doubling time, which is similar in relative size to the CNN surge, but at higher ramp rate [7]).

In addition, to estimate the efficiency of our approach, we ran a 10 hour marathon with surges (about 60) initiated at random times (Figure 2 is a part of this run). We then calculate the optimal deployment, which we define as minimum number of servers required to handle the actual workload at any time (see [3]), based on a server capacity of 60 BOPS. The average optimal deployment over the run is 1.61 servers. Our approach used an average of 1.92 servers, which is only

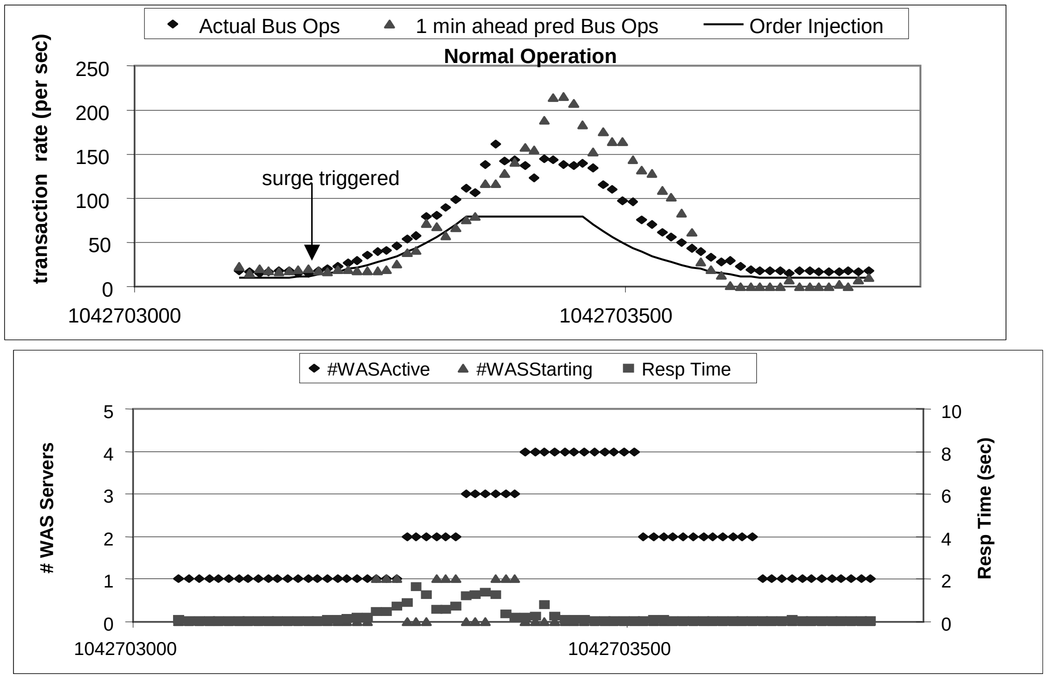

**Figure 6: Operation of system using dynamic surge protection.**

20% more than optimal. Conversely, a static deployment that can handle all the surges would require 3 servers.

We also note that this approach has been successfully tested on several different deployments. The results in Figure 6 and Figure 7 are from different hardware deployments.

For comparison, we explore a threshold-based heuristic (i.e. no forecasting, no online capacity planning) as an alternative to dynamic surge protection. We use the following threshold-based heuristic:

> Increase number of servers by 1 if response time exceeds SLO and wait 60 sec (sufficient time for server to be taking load normally) before next control action is considered.

Figure 7 displays results from this threshold-based approach, which can be compared to the performance of the dynamic surge protection approach in Figure 6. It appears that the heuristic is set too high to accomodate the provisioning lead time. By the time the heuristic reaction takes effect, response times have already grown very large, and BOPS cannot keep up with the order injection due to transaction rollbacks. (normally BOPS $\sim 1.75 \times$ order injection). Also, there needs to be a rule for releasing servers when they are no longer needed. Choosing a lower thresholds for when to add (high water mark) and remove (low water mark) servers could address these concerns. However, response time is often quite noisy. On another hardware deployment where we did some characterization of response time variability, we found that with 3 WAS servers and order injection rate = 70 (which was about the capacity of 2 servers) the response time ranged from 0.1–0.3 sec. However, on increasing the rate to 90 (still below capacity for 3 servers – average CPU idle $\sim$ 30%) the range increased significantly (0.2–0.9 sec). Choosing high and low (especially) water marks while trying to avoid the possibility of cycling servers in and out in this situation can be challenging. More robust high/low watermarks can be based on transaction rate, but that would involve some form of capacity estimation.

## 4. CONCLUSIONS

This paper describes dynamic surge protection, a technique for handling unexpected subscriber surges in systems that have resource actions with lead times (e.g. provisioning an application server). Dynamic surge protection incorporates three technologies. Short-term forecasting provides a way to anticipate the trajectory of workload demands of a surge. On-line capacity planning determines the resources required to maintain a SLO based on the anticipated workload. Configuration management via provisioning (adding/removing application servers) is how the the onset and subsiding of unexpected surges is managed.

We have conducted a number of experiments on testbed systems to gain insight into the characteristics of dynamic surge protection. Overall, we have found it to be well behaved, and the performance compares favorably to a threshold-based heuristic. In addition, the approach is cost effective – in one extended test, we found that it uses only 20% more resources than a theoretical optimal deployment and 35% less resources than a static deployment.

Our future work will address handling of multiple workloads, and resource actions that include tuning and admission control. We also plan to leverage Grid services.

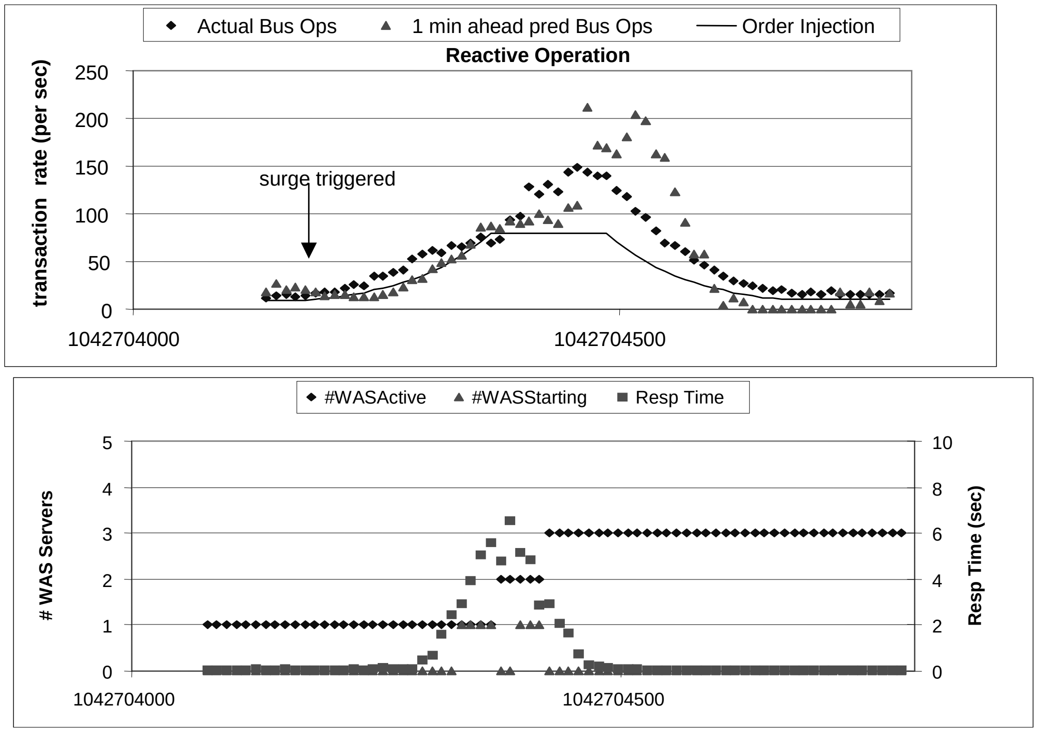

**Figure 7: Operation of system using a threshold-based heuristic.**

## 5. REFERENCES

[1] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger. Adaptive algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2), 1997.

[2] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.

[3] A. Chandra, P. Goyal, and P. Shenoy. Quantifying the benefits of resource multiplexing in on-demand data centers. *Proceedings of the First ACM Workshop on Algorithms and Architectures for Self-Managing Systems - to appear*, 2003.

[4] M. E. Crovella. Performance characteristics of the world wide web. *Performance Evaluation, LNCS*, 1769, 2000.

[5] IBM. High volume web site performance simulator. *http://www7b.boulder.ibm.com/wsdd/library/techarticles/hvws/perfsimulator.html*, 2002.

[6] IBM. Autonomic computing. *http://www.ibm.com/autonomic*, 2003.

[7] Bill Lefebvre. Facing a world crisis. *USENIX LISA*, 2001.

[8] L. W. Russell S. P. Morgan and E. G. Chron. Clockwork: A new movement in autonomic systems. *IBM Systems Journal*, 42(1), 2003.

[9] Kathleen Ohlson. Victoria's secret knows ads, not the web. *Computer World*, February 1999.

[10] Hewlett Packard. HP utility data center. *http://www.hp.com/go/hpudc*, 2003.

[11] M. Goldszmidt D. Palma and B. Sabata. On the quantification of e-business capacity. *Proceedings of the 3rd ACM conference on Electronic Commerce*, 2001.

[12] K Appleby S. Fakhouri L. Fong M. K. G. Goldszmidt S. Krishnakumar D. Pazel J. Pershing and B. Rochwerger. Oceano–SLA-based management of a computing utility. *Proceedings of the IFIP/IEEE Symposium on Integrated Network Management*, 2001.

[13] ProvisionSoft. ProvisionSoft Home Page. *http://www.provisionsoft.com*, 2003.

[14] Sun Micro Systems. Sun N1. *http://wwws.sun.com/software/solutions/n1*, 2003.

[15] ThinkDynamics. Thinkdynamics Home Page. *http://www.thinkdynamics.com*, 2003.

[16] Zeus. Why web technology is vital to your business. *Computer World*, 2003.

[17] M. S. Squillante L. Zhang and D. Y. Yao. Web traffic modeling and web server performance analysis. *Proceedings of the 38th IEEE Conference on Decision and Control*, 5, 1999.

[18] J. Rolia X. Zhu and M. Arlitt. Resource access management for a utility hosting enterprise applications. *Integrated Network Management VIII*, 2003.