Dynamic Adaptation of Rules for Temporal Event Correlation in Distributed Systems

Rean Griffith*, Joseph L. Hellerstein**, Yixin Diao**, and Gail Kaiser*

*Computer Science Department
Columbia University, New York, New York
{kaiser, rg2023}@cs.columbia.edu

**IBM Thomas J. Watson Research Center
Hawthorne, New York
{hellers, dias}@us.ibm.com

January 22, 2005

Abstract

Event correlation is essential to realizing self-managing distributed systems. For example, distributed systems often require that events be correlated from multiple systems using temporal patterns to detect denial of service attacks and to warn of problems with business critical applications that run on multiple servers. This paper addresses how to specify timer values for temporal patterns so as to manage the trade-off between false alarms and undetected alarms. A central concern is addressing the variability of event propagation delays due to factors such as contention for network and server resources. To this end, we develop an architecture and an adaptive control algorithm that dynamically compensate for variations in propagation delays. Our approach makes Management Stations more autonomic by avoiding the need for manual adjustments of timer values in temporal rules. Further, studies we conducted of a testbed system suggest that our approach produces results that are at least as good as an optimal fixed setting of timer values.

1. Introduction

Event correlation is a key component of today's Information Technology management infrastructure, and we expect it to be essential to the future of self-managing systems. Increasingly, it is important to correlate events from different systems using temporal patterns. For example, a denial of service attack may be detected by correlating failed logins on multiple machines in a short period of time (e.g., under one second), and problems with multi-server applications can be detected by the transition times between processing stages that occur on different severs. This paper addresses how to determine timer values for temporal patterns so as to properly balance false alarms and undetected alarms. A central concern is addressing the variability in event propagation delays due to contention for network and server resources and other factors. To this end, we develop an architecture and an adaptive control algorithm that dynamically compensate for variations in propagation delays. Not only does our approach make Management Stations more autonomic by avoiding the need for manual adjustments in timer values, our studies suggest that our approach produces results that are at least as good as an optimal fixed adjustment in timer values.

Traditionally, event correlation is done using if-then rules (also called event-condition-action) that are interpreted by an engine in a **Management Station**. The if-part of these rules consists of an event pattern and the then-part specifies an action to be taken (although other approaches can be employed as well as in [11]). Herein, our focus is on the if-part and so we assume that the then-part is an alarm (which is the most common case in practice) such as sending an email, paging an administrator, or creating a trouble ticket.

The challenges associated with managing distributed systems have made it common for correlation rules to relate events from multiple systems. Consider the illustrative

examples below in which the question marks indicate variables that are bound to values based on the content of events received.

- Rule 1: If there is a `SlowResponse` event from system ?S1 at location ?L1 within 1 minute of another `SlowResponse` event from system ?S2 $\neq$?S1 at location ?L1 and there is no `SlowResponse` event from system S3 at location ?L2 $\neq$?L1, then alert the Network Manager for location ?L1.

- Rule 2: If there is a `CompletedPhase1` event from application ?A1 and there is no `CompletedPhase2` event from application ?A1 within 5 seconds of the first event, then alert the Application Manager for application ?A1.

- Rule 3: If there is a `FailedLogin` event from system ?S1 in cluster ?C1 and there is a `FailedLogin` event from system ?S2 in cluster ?C1 within 1 second, then alert the Security Manager for cluster ?C1.

Rule 1 provides a way to distinguish network problems from application problems based on a pattern consisting of two events from different machines at the same location. Rule 2 checks on the health of a critical business application that has processing steps that may be executed on different systems. Rule 3 checks for certain kinds of security intrusions by looking for patterns of failed logins. In all of these rules, the if-part of the rule contains a pattern that is to be matched by events from multiple nodes. Also, in all cases there is a **timer value** that constrains the maximum elapsed time between receiving the first and last events in the pattern (although in general more complex temporal patterns may be used [5]). For Rule 1, the timer value is determined by the experience of system administrators with the timing of related events. For Rule 2, the timer value relates to the time between processing steps. For Rule 3, the timer value is chosen to distinguish human interactions from robots.

Performing temporal event correlation such as is done in Rules R1–R3 requires that events be timestamped. Unfortunately, for many situations in which event correlation is needed, the clock at the event source is unreliable. For example, the event source may be a personal workstation or a customer-managed server in outsourcing situations that is not running appropriate clock synchronization software. Even if the event source is running the right software, the system may have a partial failure that affects clock synchronization, which may in fact be why it is sending an event. Last, security protocols such as Kerberos rely on clock synchronization, and so clock synchronization may be compromised early in an attack.

If the timestamp at the event source is unreliable, the management station uses the arrival time of the event at the Management Station. These arrival times include the delay to propagate the event from its source. If propagation delays are the same for all events in a pattern, this is not a problem since the elapsed time of event patterns is unchanged. Unfortunately, there may be substantial **propagation skew**, a term we use to refer to the difference between propagation delays in an event pattern. Experiments we conducted reveal propagation skews that are often within 50% of the pattern generation time. Among the reasons for propagation skews are transients in resource usage and contention with administrative tasks (e.g., Java garbage collection).

We can compensate for propagation skew by adjusting timer values based on measurements taken from appropriate probes. However, since the propagation skew varies from one instance of a pattern to another, we can only estimate the skew for any particular pattern. This creates a conundrum. If we over-compensate for propagation skew by using too large of a timer value, true problems may go undetected. We refer to these as **undetected alarms**. Clearly, undetected alarms diminish the ability of the system to respond quickly to problems. On the other hand, if we use too small of a timer value, there will be many alarms that are generated for situations that do not merit action, such as a change in workload or re-allocation of resources to the application. We refer to these as **false alarms**. False alarms are undesirable since they divert the operations staff from true problems.

In terms of related work, event correlation has been widely used to monitor and analyze networks, systems, and applications for the last twenty years (e.g., [8]). Commonly addressed issues include correlation speed and accuracy [4, 11, 9] and the expressiveness of correlation patterns. For the latter, there has been particular interest in non-rule based approaches [11], probabilistic correlation [6], and temporal patterns [7, 1, 5]. Our work relates to temporal patterns in distributed systems. In particular, none of the systems in [7, 1, 5] mention propagation skew. Hence, none of these systems provide the architectural or algorithmic support needed to compensate for propagation skew.

This paper makes the following contributions:

1. description of the problem of propagation skew for temporal event correlation in distributed systems, including measurements of propagation skew for a testbed system;

2. an architecture that includes Calibration Probes, Probe Monitors, and a Controller that collaborate to adjust timer values in order to compensate for propagation skew; and

3. an adaptive control algorithm for dynamically adjusting timer values to compensate for propagation skew

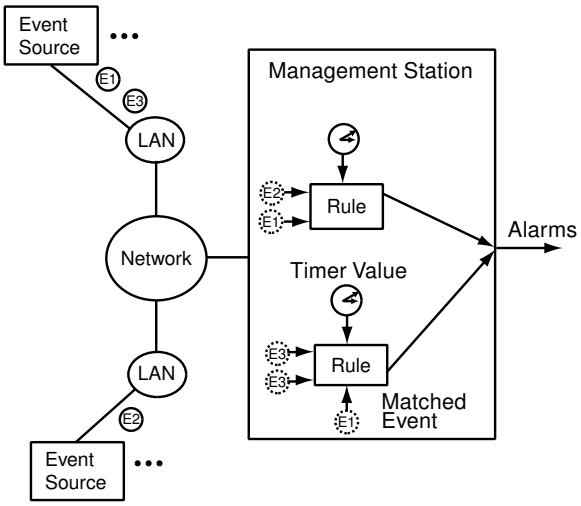

Figure 1. *Architecture of a Management Station that supports temporal event correlation.*

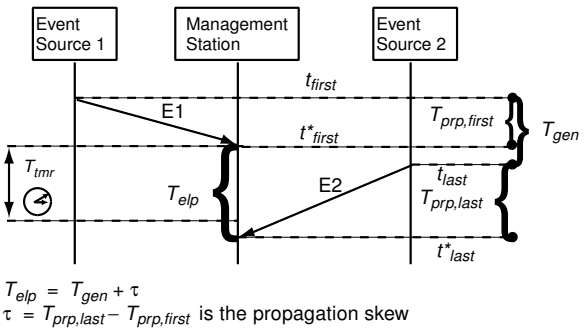

Figure 2. *Interaction diagram for temporal event correlation. Timer values are specified based on the time to generate a pattern T_{gen}, but the estimate of this at the Management Station is T_{elp}.*

and an assessment of the algorithm in terms of the probability of a correct result.

Our approach makes Management Stations more autonomic by avoiding the need for manual adjustments in timer values. Further, studies of a testbed system suggest that our approach produces results that are at least as good as an optimal fixed setting of timer values.

The remainder of the paper is organized as follows. Section 2 describes the architecture we propose. Section 3 details our adaptive control algorithm that compensates for propagation skews. Section 4 assesses our approach using data from a testbed system. Our conclusions are presented in Section 5.

2. Architecture

This section describes the architecture of a system that compensates for propagation skews in temporal event correlation for distributed systems.

Figure 1 illustrates the characteristics of existing approaches to temporal correlation of events in distributed systems [7, 1, 5] as they relate to the problem of propagation skew. Event sources generate events (the solid circles) that traverse one or more networks. When an event arrives at the Management Station, a timestamp is applied. Then, the Management Station queues a copy of the event for each partially instantiated pattern for which there is a match with the incoming event (indicated by dotted circles). When a pattern is first instantiated for a rule, a timeout is specified with duration equal to the timer value for the rule. If the timeout occurs before matching the last event in the pattern, an alarm is generated.

Figure 2 illustrates the dynamics of correlating a temporal pattern consisting of the two events, $E1$ and $E2$. $E1$ is generated by Event Source 1 at time t_{first}, and $E2$ is generated by Event Source 2 at time t_{last}. Thus, the pattern generation time is $T_{gen} = t_{last} - t_{first}$.

Administrators write rules for temporal correlation based on pattern generation time. As in rules R1-R3, consider a timer value T_{tmr} that is chosen so that an alarm is be generated if $T_{gen} > T_{tmr}$. Since the Management Station does not know T_{gen}, it uses T_{elp} instead. From Figure 2, $T_{elp} = t^*_{last} - t^*_{first} = T_{gen} + \tau$, where τ is the propagation skew. Propagation skew is computed as follows. The propagation delay of the first and last events are $T_{prp,first} = t^*_{first} - t_{first}$ and $T_{prp,last} = t^*_{last} - t_{last}$. So, $\tau = T_{prp,last} - T_{prp,first}$.

The elapsed time of a pattern T_{elp} as seen at the Management Station differs from the pattern generation time by τ, the propagation skew. If $T_{prp,last} = T_{prp,first}$ then $\tau = 0$ and so $T_{elp} = T_{gen}$, which is the ideal case. However, in our experiments, τ varies considerably.

Figure 3 depicts the ways in which we extend the architecture in Figure 1 to compensate for propagation skew. There are four considerations.

1. instrumentation that creates events so that there are known pattern generation times for one or more Calibration Patterns;

2. a way to measure the propagation skew of the events generated in (1);

3. a mechanism for computing slack times that compensate for propagation skews; and

4. rules that use slack times to adjust timer values.

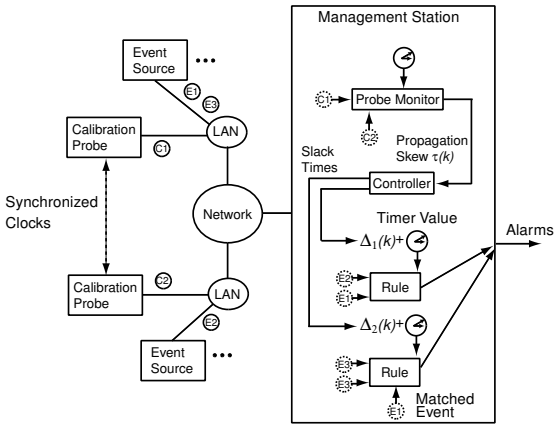

Figure 3. *Architecture that supports compensation for propagation skews by having: (1) Calibration Probes that create Calibration Patterns that have a known pattern generation time; (2) rules that use slack times to adjust timer values; (3) a Probe Monitor that computes propagation skews for Calibration Patterns; and (4) a Controller that computes values of slack time.*

Initialize calibration pattern i
1. Send `Start` message to calibration probe i.

First event in calibration pattern i
1. `Time(EventReceived)` = NOW.
2. $k_i = k_i + 1$.
3. Start `ExcessiveIntraframe` timer.

On receipt of last event in calibration pattern i
1. `Time(EventReceived)` = NOW.
2. Invoke `Controller` with
   $\tau_i(k_i)$ = `Time(LastEvent)`
   `- Time(FirstEvent)`.
3. Delete all matched events.

Timeout for calibration pattern i**.**
1. Delete all matched events.

Figure 4. *Operation of the Probe Monitor in the Management Station.*

Item (1) is addressed by the Calibration Probes. Calibration Probes run on systems that are part of the management infrastructure and so their clocks are reliable and synchronized (e.g., using the Network Timer Protocol). Calibration Probes generate Calibration Events that include the timestamps applied at the event source. Calibration Events also have a timestamp corresponding to the time of their arrival at the Management Station. For example, once Event $C2$ in Figure 2 arrives at the Management Station, it has timestamps corresponding to t_{last} and t^*_{last}. Calibration Events are selected so as to create a Calibration Pattern that is detected at the Management Station.

Item (2) is handled by the Probe Monitor on the Management Station. The Probe Monitor measures propagation skews for Calibration Patterns based on information in the Calibration Events. Figure 4 details the operation of the Probe Monitor.

Item (3) is addressed by the controller, which dynamically updates slack times as propagation skews are received. We discuss the controller at length in the next section.

Item (4) is handled by including a slack time for each partially instantiated pattern. As before, the timer value is specified by administrators based on their insights into the temporal pattern. The slack time is used to compensate for propagation skew. The operation of the system in Figure 1 is changed in that when the first event of a pattern is matched, the Management Station specifies a timeout equal to the *sum of the timer value and the slack time*. We note in passing that our architecture can readily be generalized to have multiple timer values and slack times if more complex temporal patterns are used.

3. Control Algorithm

This section develops the adaptive control algorithm that updates slack times to compensate for propagation skew. The algorithm is based on a simple technique from statistical hypothesis testing that uses non-parameteric statistics, a class of approaches that do not assume a particular probability distribution.

We want the control algorithm to choose slack times that maximize the probability of getting a correct result. There are two cases. In the first, pattern generation time $T_{gen,i}(k)$ for the k-th pattern of the i-th rule is larger than the timer value $T_{tmr,i}$ of i-th rule. Under these circumstances, the correct result is that an alarm is generated. In the second case, $T_{gen,i}(k)$ is less than $T_{tmr,i}$. Here, no alarm should be generated. In statistical hypothesis testing, these cases are expressed using negative logic. That is, an incorrect result in the first case is a undetected alarm, and an incorrect result in the second case is a false alarm. Herein, we simplify matters by focusing on the probability of a correct result.

We now show how the probability of a correct result relates to slack time. To simplify matters, we consider a single Calibration Pattern with generation time T_{gen}. We study the probability of a correct result for the i-th correlation rule whose if-part is satisfied by the Calibration Pattern. This rule has timer value $T_{tmr,i}$. We define the **timer offset** for this rule to be $\delta_i = T_{tmr,i} - T_{gen}$. Note that Rule i produces a correct result if it generates an alarm when $\delta_i < 0$, and it does not generate an alarm when $\delta_i > 0$.

The concept of the timer offset turns out to be central to the theory that underlies the selection of slack times. For the case in which an alarm should be generated, we have

$$\begin{aligned} &P(\text{Correct}|\text{Alarm should be generated}) \\ &= P(\text{Correct}|\delta_i < 0) \\ &= P(T_{elp,i}(k) > T_{tmr,i} + \Delta_i(k)|\delta_i < 0) \\ &= P(T_{gen,i} + \tau_i(k) > T_{tmr,i} + \Delta_i(k)|\delta_i < 0) \\ &= P(\tau_i(k) > \Delta_i(k) + \delta_i|\delta_i < 0) \end{aligned}$$

Observe that we increase the probability of a correct result if either the slack time is close to zero or the timer offset is more negative. The latter case means that we are more likely to raise an alarm if the pattern generation time is much smaller than the timer value. The case of when an alarm should not be generated is addressed in analogous manner.

$$\begin{aligned} &P(\text{Correct}|\text{Alarm should not be generated}) \\ &= P(\text{Correct}|\delta_i > 0) \\ &= P(T_{elp,i}(k) < T_{tmr,i} + \Delta_i(k)|\delta_i > 0) \\ &= P(T_{gen,i} + \tau_i(k) < T_{tmr,i} + \Delta_i(k)|\delta_i > 0) \\ &= P(\tau_i(k) < \Delta_i(k) + \delta_i|\delta_i > 0) \end{aligned}$$

Here, we increase the probability of a correct result if either slack times or the timer offset are large. The latter case means that the pattern generation time is much larger than the timer value. Observe that in both cases, when skew is close to zero, then the magnitude of slack time need not be large to get a correct result.

Figure 5 plots the probability of a correct result versus the timer offset for data we collected using the experimental setup described in the next section. There are three plots. The first is from an experiment in which no load was placed on the systems or network. Here, propagation skew is close to 0. Hence, there is a high probability of a correct result since slack time is 0. The second and third plots of the first row present data collected when there was substantial load. In the second plot, slack time is 0. We see that the probability of a correct result is larger for negative timer offsets δ_i (i.e., when an alarm should be generated), but the probability of a correct result is small when $\delta_i > 0$, at least until δ_i becomes fairly large. The reason for this asymmetry

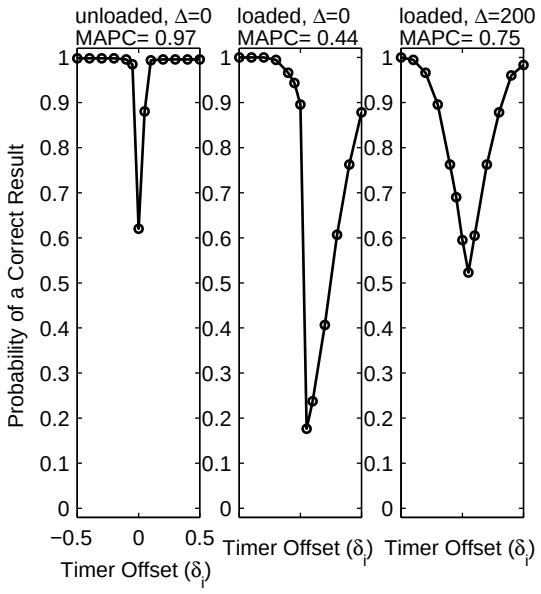

Figure 5. *Probability of a correct result for loaded and unloaded testbed configurations and different fixed settings of slack time (Δ). The horizontal axis is the timer offset, which is the difference between the timer value of a rule and the pattern generation time.*

can be explained the distribution of propagation skews. Its mean is approximately 0.2 second. As a result, if $\Delta_i(k) = 0$, then a larger δ_i is needed so that $\tau_i(k) < \delta_i$. We can compensate for this by setting $\Delta_i(k)$ to 0.2 second. which is done in the third plot. This results in a larger probability of a correct result when $\delta_i > 0$. However, it also reduces the probability of a correct result for $\delta_i < 0$.

The foregoing demonstrates a fundamental trade-off between false alarms and undetected alarms. We are assured of a correct result in the case where $\delta_i < 0$ by using a very large $\Delta_i(k)$. However, doing so results in poor performance when $\delta_i > 0$. The reverse applies as well.

We now introduce our metric for quantifying the performance of an approach to computing slack times. A way to take into account the trade-off just mentioned is to consider the minimum probability of a correct result for the two cases. That is, $\min\{P(\text{Correct}|\text{ Alarm should be generated}), P(\text{Correct}|\text{Alarm should not be generated})\} = \min\{P(\text{Correct}|\ \delta_i < 0), P(\text{Correct}|\delta_i > 0)\}$.

In our studies, we approximate the minimum probability of a correct result by averaging across multiple values of δ_i (both negative and positive) for known pattern generation times. We refer to this as the **minimum average probability of a correct result ($MAPC$)**. $MAPC$ is based on a set of timer values $T_{tmr,i} \in S_<$ such that $T_{tmr,i} < T_{gen}$ (in which case an alarm should be generated), and a set of timer

1. Add $\tau_i(k_i)$ to buffer the buffer for calibration pattern i.
2. Remove $\tau_i(k_i - N)$ from buffer i
(N is the size of the buffer.)
3. $\Delta_i(k_i)$ = middle value of buffer i.

Figure 6. *Operation of the Adaptive Control Algorithm.*

values $T_{tmr,j} \in S_>$ for which $T_{tmr,j} > T_{gen}$ (and hence no alarm should be generated). We use $AvgCorrect_{gen}$ to denote the average probability of a correct result in the first case, and $AvgCorrect_{nogen}$ to denote this metric in the second case.

$$MAPC = \min[AvgCorrect_{gen}, AvgCorrect_{nogen}] \quad (1)$$

Here, $AvgCorrect_{gen} = Average\left(\sum_{T_{tmr,i} \in S_<} \sum_k \{\tau_i(k+1) > \Delta_i(k)\}\right)$, $AvgCorrect_{nogen} = Average\left(\sum_{T_{tmr,j} \in S_>} \sum_k \{\tau_j(k+1) < \Delta_j(k)\}\right)$, and $\{x < y\} \in \{0, 1\}$ depending on whether the inequality is false or true. Note that since $MAPC$ is an average of probabilities, $0 \leq MAPC \leq 1$, with $MAPC = 1$ being a perfect control algorithm.

Figure 5 displays MAPC values in the titles of the three plots. In the first plot, $P(Correct) \approx 1$ except at $\delta = 0$, in which case $P(Correct) \approx 0.5$. Consistent with this, the $MAPC$ is $0.97 \approx 1$. In the second plot, $P(Correct)$ is low for $\delta_i > 0$. As a result, its $MAPC = 0.44$. In the third plot, slack time is adjusted to better balance P(Correct|Alarm should be generated) and P(Correct|Alarm should not be generated). Here, $MAPC = 0.75$.

The goal of our adaptive control algorithm is to maximize $MAPC$. Our intuition from Figure 5 is that this is achieved if slack time is chosen so as to balance P(Correct|Alarm should be generated) and P(Correct|Alarm should not be generated). More specifically, from the first and third plots in Figure 5, we want $P(Correct) \approx 0.5$ if $\delta_i = 0$. This observation allows us to characterize slack times. Specifically, we want $P(\tau_i(k) > \Delta_i(k) + \delta_i | \delta_i = 0) = 0.5$. And so $P(\tau_i(k) > \Delta_i(k) | \delta_i = 0) = 0.5$. That is, slack time should be chosen to be the median of the distribution of propagation skews. We note in passing that it may be that undetected alarms are more costly than false alarms, or the reverse. Hence, we might want to adjust the desired probability of an alarm when $\delta_i = 0$. This in turn means that the control algorithm estimates a different percentile of the skew distribution to compute slack time.

We compute slack time by using a non-parameteric procedure for estimating the median of the distribution of propagation skews [10]. By non-parameteric, we mean that the procedure makes no assumption about the distribution of the propagation skews (which is clearly an advantage for an environment that experiences considerable change). However, the procedure does assume that propagation skews are independent and identically distributed. Figure 6 provides the details. Our algorithm retains the last N propagation skews in a buffer. The median is the middle value of the sorted list.

The only parameter of the adaptive control algorithm is the buffer size N. For stationary skew distributions, a larger N reduces the variance of the estimate of the median and hence results in a higher probability of a correct result. However, non-stationarities arise if a file transfer is started that increases network delays or administrative tasks begin execution on the management station. In these cases a larger N is a disadvantage in that it takes longer for the buffer to be populated entirely by observations from the new distribution.

4. Experimental Results

We developed a testbed system based on the architecture depicted in Figure 3 in which the Management Station extends the Event Distiller [5] and the event transport is the Siena Publish/Subscribe bus [3]. We study a situation in which there are two event sources, both on the same system (so that we have very accurate measurements of pattern generation times), and the Management Station is on the same LAN as the event sources. Two configurations are considered. In the **unloaded configuration**, there are separate machines for Event Distiller and Siena. In the **loaded configuration**, Event Distiller and Siena are co-located on the same machine. In the experiments reported here, the pattern generation time is 2 seconds, and the Calibration Probes run on an AMD Anthlon XP 1800 with 1 GB RAM. The management station is a 3 GHz P4 running Windows XP with 1 GB RAM. In the unloaded configuration, Siena runs on a 1 GHz P3 with 512MB RAM and RedHat Linux 2.4.20.

Figure 7 reports data from two runs on our testbed, one for an unloaded configuration and the second for a loaded configuration. In the unloaded case, we see that the propagation skews are tightly clustered around 0, although there are a few large spikes. The second plot in the top row is the cumulative distribution function (CDF), which reinforces the view that values are tightly clustered. Also plotted are the autocorrelations between propagation skews. Note that all autocorrelations lie within the dashed lines, indicating that they are not statistically significant as determined by the Bartlett Test [2]. This fact bodes well for our use of non-parameteric statistics that require independent observations.

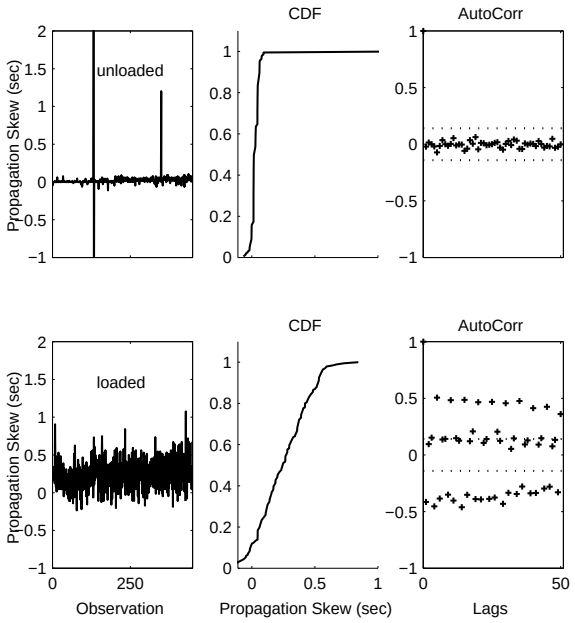

Figure 7. *Characteristics of the propagation skew data for unloaded and loaded configurations. CDF is the empirical cumulative distribution, and AutoCorr is the autocorrelation for a stationary segment of the propagation skews.*

The bottom row of Figure 7 reports results from a loaded configuration. Here, propagation skews are much more variable and considerably larger, a fact that is reflected in the CDF plot. We also see substantial autocorrelations (possibly due to periodic activities), a fact that undermines the assumption of independence of the propagation skews that the controller algorithm relies on.

Figure 8 assesses the effectiveness of using fixed slack times for the unloaded and loaded configurations reported in Figure 7. In all of the plots, the horizontal axis is the slack time Δ and the vertical axis is Minimum Average Probability of a Correct result ($MAPC$). We see that large $MAPC$ values are achieved with a fixed slack time near 0 in the unloaded cases. However, for the loaded configurations, $MAPC$ is maximized at larger fixed slack times. This can be explained by looking at the distribution of propagation delays. For example, the "unloaded 1" plot corresponds to the data plotted in the top row of Figure 7. We see that the median of this distribution (the skew value corresponding to the 50-th percentile) is approximately 0, which is the fixed slack time at which $MAPC$ is maximized. Similarly, the "loaded 1" plot corresponds to the bottom row of Figure 7. Here, the median of the skew distribution is a little more than 0.2 seconds, which is where $MAPC$ is maximized for these data.

The solid line in Figure 8 plots the $MAPC$ values

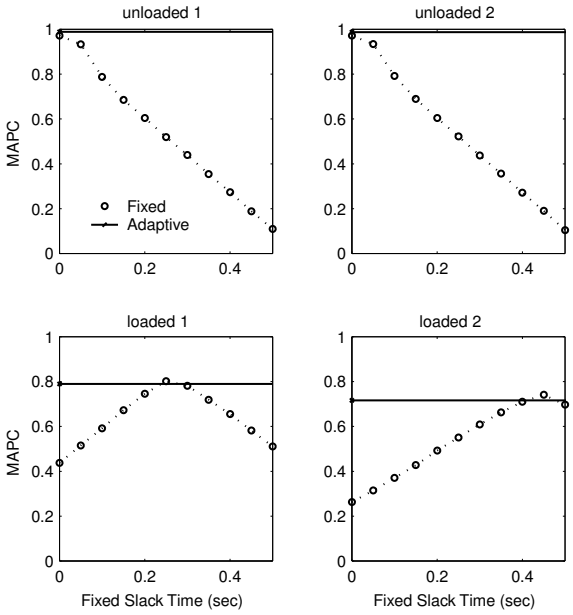

Figure 8. *Evaluation of fixed slack times (dashed line) and the adaptive control algorithm for* stationary *propagation skews. The horizontal axis is the value of the fixed slack time, and the vertical axis is the minimum average probability of a correct test result ($MAPC$). The adaptive control algorithm consistently does as well as the best fixed slack time.*

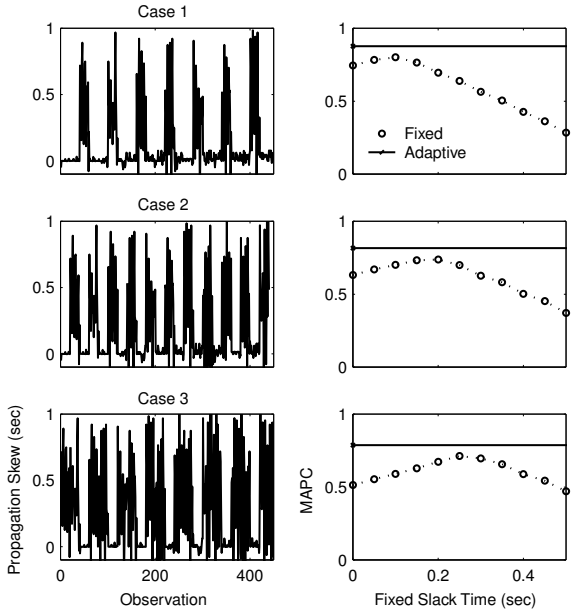

Figure 9. *Evaluation of fixed slack times (dashed line) and the adaptive control algorithm for* non-stationary *propagation skews. The plots in the first column are the propagation skews. The second column are plots that evaluate $MAPC$ in the same way as Figure 8.*

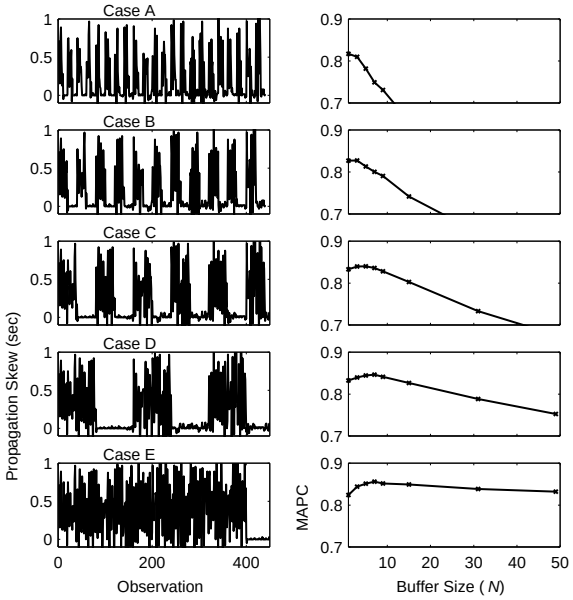

Figure 10. *Effect on MAPC of N, the size of the buffer used in the adaptive control algorithm. The plots in the first column are the propagation skews, and the second column plots $MAPC$ for different buffer sizes.*

achieved by our adaptive control algorithm ($N = 5$) that is described in Figure 6. We see that in all cases, the adaptive control algorithm selects slack times that are very close to the value of fixed slack time that maximizes $MAPC$. This is impressive in two respects. First, we did not have to parameterize or train the controller. That is, slack times are selected in a self-managing way. Second, we achieve near optimal results in the loaded configuration even though the data have significant autocorrelations, a situation that violates the independence assumption of the technique we use to estimate the median of the propagation skew distribution in the adaptive control algorithm.

Next, we consider situations in which the distribution of propagation skews changes. The data we use are synthesized by alternating between propagation skews obtained in our testbed for loaded and unloaded configurations. Figure 9 consists of six plots organized into two columns with three rows. Plots in the first column are propagation skews used to drive a simulated Management Station. The second column reports $MAPC$ for both fixed slack times and the adaptive control algorithm. We see that the adaptive algorithm consistently does better than the best setting of fixed slack time.

Last, we evaluate the impact on MAPC of the controller buffer size N. Figure 10 contains ten plots organized into two columns with five rows. As in Figure 9, the first column are synthesized traces of propagation skews from our experimental runs that are constructed by alternating blocks of data from different experiments. In case A, there are many changes in the distribution of propagation skew. Here, the optimal buffer size is small. The reason for this is that a smaller buffer size means there is less history and hence adaptation occurs faster. On the other hand, when changes in the skew distribution are infrequent (i.e., Case E), the optimal buffer size is larger. The insight here is that a larger buffer size provides a lower variance estimate of the median of the skew distribution. However, the reduction in variance itself declines rapidly as N increases. As a result, there is little value in having a buffer size much larger than 5 or 7, even for stationary data.

5. Conclusions

Distributed systems often require that events be correlated from multiple systems using temporal patterns. This paper addresses how to specify timer values for temporal patterns in distributed systems so as to manage the tradeoff between false alarms and undetected alarms. A central concern is addressing the variability in event propagation times due to contention for network and server resources and other factors. We propose an approach that makes Management Stations more autonomic by avoiding the need for

manual adjustments to timer values. Further, studies of a testbed system suggest that our approach produces results that are at least as good as an optimal fixed adjustment in timer values.

Our results are in three areas. First, we describe the problem of propagation skew for temporal event correlation in distributed systems, including measurements of propagation skew for a testbed system. These measurements show that propagation skews can be substantial, on the order of 50% of the pattern generation time in our testbed experiments. Second, we introduce an architecture that uses dynamically computed slack times to compensate for propagation skews. The architecture includes Calibration Probes, Probe Monitors, and a Controller. Last, we develop an adaptive control algorithm for computing slack times, and we assess the algorithm in terms of the probability of a correct result. Our measurements suggest that the algorithm adapts well to changes in propagation skews, typically doing better than the best result achieved by a fixed slack time.

Our future work will involve more extensive measurements of propagation skews and extensions to more complex temporal patterns.

Acknowledgements

Kaiser's Programming Systems Lab is funded in part by National Science Foundation grants CNS-0426623, CCR-0203876 and EIA-0202063.

References

[1] A. Adi, A. Biger, D. Botzer, O. Etzion, and Z. Sommer. Context awareness in amit. In *Autonomic Computing Workshop, 2003*, pages 160–166. IEEE Press, June 2003.

[2] G. E. P. Box and G. M. Jenkins. *Time Series Analysis Forecasting and Control*. Prentice Hall, 1976.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.*, 19(3):332–383, 2001.

[4] G. Jiang and G. Cybenko. Temporal and spatial distributed event correlation for network security.

[5] G. E. Kaiser, J. Parekh, P. Gross, and G. Valetto. Kinesthetics extreme: An external infrastructure for monitoring distributed legacy systems. In *Active Middleware Services*, pages 22–31, 2003.

[6] A. Konstantinou, D. Florissi, and Y. Yemini. Towards self-configuring networks. In *DARPA Active Networks Conference and Exposition (DANCE)*. IEEE Press, 2002.

[7] D. Luckham. *The Power of Events*. Addison–Wesley, 75 Arlington Street,Suite 300, Boston, MA 02116, first edition, 2002.

[8] K. Milliken, A. Cruise, R. Ennis, A. Finkel, J. Hellerstein, D. Loeb, D. Klein, M. Masullo, H. V. Woerkom, and N. Waite. YES/MVS and the autonomation of operations for large computer complexes. *IBM Systems Journal*, 25(2), 1986.

[9] O. C. O. Systems. Rootcause: Using a flight recorder to speed remote debugging and problem resolution.

[10] A. Walker. A note on the asymptotic distribution of sample quantiles. *Journal of the Royal Statistical Society*, 30:570–575, 1968.

[11] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *IEEE Communications Magazine*, 34(5):82–90, 1996.