Dynamic Adaptation of Temporal Event Correlation for QoS Management in Distributed Systems

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

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

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

February 20, 2006

Abstract

Temporal event correlation is essential to managing quality of service in distributed systems, especially correlating events from multiple components to detect problems with availability, performance, and denial of service attacks. Two challenges in temporal event correlation are: (1) handling lost events and (2) dealing with inaccurate clocks. We show that both challenges are related to event propagation skew, the difference in the time required for events to propagate from the event source to the management station at which correlation takes place (e.g., as a result of variations in resource utilizations). We develop an approach to adjusting the timer values of event correlation rules based on propagation skew in order to reduce missed alarms and false alarms. Our approach has three parts: an infrastructure for real-time measurement of propagation skew, a statistical approach to estimating propagation skew, and a controller that uses estimates of propagation skew to update timer values in temporal rules. Our approach eliminates the need for manual adjustments of timer values. Further, studies of a prototype implementation suggest that our approach produces results that are at least as good as an optimal fixed adjustment in timer values.

1. Introduction

Maintaining quality of service (QoS) in complex Information Technology (IT) environments requires a capability for correlation of temporal events. 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 address issues with lost events and inaccurate clocks. A central concern is addressing propagation skew, the variability in event propagation delays due to contention for network and server resources and other factors. We develop an approach that uses propagation skew to dynamically adjust timer values in temporal correlation rules so as to reduce missed alarms and false alarms. Our approach consists of: an infrastructure for real-time measurement of propagation skew, a statistical technique that estimates propagation skew, and a controller that uses estimates of propagation skew to update timer values in temporal rules. One appeal of this approach is that it eliminates the need for manual adjustments of timer values. Further, studies of a prototype implementation 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.

Managing distributed systems often requires correlation rules that 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 no `Heartbeat` event from system ?S1 at location ?L1 within 1 minute of another `Heartbeat` event from system ?S2 $\neq$?S1 at location ?L1 and there is a `Heartbeat` 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.

Implementing temporal event correlation requires an appropriate runtime infrastructure. Centeral to this is the concept of a **partial correlation instance,** the context created while correlation matching is underway for a rule. A partial correlation instance is created on the arrival of the first event that matches an event type in a rule. Additional events are included in the partial correlation instance if they satisfy two kinds of constraints. First, the event must match the rule's **data constraints**, such as $?S2 \neq ?S1$ in Rule 1. Second, the event must comply with the rule's **temporal constraints,** such as arriving within 1 minute of another event. If all events specified in the rule are instantiated, then the partial correlation instance becomes a completed correlation instance, at which time the rule's action is executed and the correlation instance is removed. If a partial correlation instance remains uncompleted for a sufficiently long time (e.g., as specified by a time-out value), the partial correlation instance is discarded.

There are two challenges in managing the lifecycle of partial correlation instances. The first is dealing with lost events. This requires a good choice for the value of the time-out of the partial correlation instance. If the time-out value is too small, then there will be undetected alarms since partial correlation instances will be discarded before all the matching events arrive. On the other hand, if the time-out value is too large, there may be considerable memory and processing overheads due to long-lived partial correlation instances.

The second challenge in managing the lifecycle of partial correlation instances is compensating for inaccurate timestamps due to unsynchronized and/or inaccurate clocks. Our perspective here is based on a business model used by many organizations in which the management of the IT infrastructure is outsourced to **Management Service Providers (MSPs)**. The MSP could be an external organization such as IBM, EDS, and Accenture. Or, the MSP could be a corporate IT organization. In either case, the MSP creates a management infrastructure consisting of a Management Station and associated MSP (e.g., agents, probes, network sniffers) in which timestamps are accurate. Since the MSP typically has little control over software running on customer-managed elements (e.g., application servers and desktops), events generated by customer-managed element may contain inaccurate timestamps. Indeed, the customer-managed element could be infected with a virus or worm that affects the accuracy of event timestamps. The impact of inaccurate event timestamps is to reduce the accuracy with which temporal constraints can be evaluated, which in turn can result in undetected alarms or false alarms.

We assume that timestamps are correctly synchronized within MSP elements since doing so is central part of the MSP responsibilities. Our strategy for obtaining accurate event timestamps is to replace the timestamps provided by customer-managed elements with the arrival time of the event at MSP elements. These arrival times are affected by the delay to propagate the event from its source (including any software overheads in the node from which the event originated). If propagation delays are the same for all events in a completed correlation instance, the true elapsed time of event pattern is the same as the elapsed time of the event pattern as measured by the MSP infrastructure. Unfortunately, there may be substantial **propagation skew**, a term we use to refer to the variation in propagation delays within an event pattern. Experiments we conducted reveal propagation skews that are within 50% of the pattern elapsed time, a fact that can greatly increase the rate of missed alarms and false alarms. Among the reasons for propagation skews are transients in resource usage and contention with administrative tasks (e.g., Java garbage collection).

It turns out that lost events also can be addressed by correctly compensating for propagation skew. For example, the left-hand side of Rule 1 is satisfied if no second `Heartbeat` event arrives within a minute of the first. Thus, the time-out value of the event pattern should be at least one minute plus the propagation skew.

There are three parts to our approach to compensating for propagation skew: measurement, estimation, and correction. Our approach to measurement is to incorporate into the MSP infrastructure a capability to generate calibration events that are representative of events generated by customer-managed elements. Estimation is accomplished by developing a statistical technique that is applied to the timestamps of calibration events. Correction is achieved by including in the Management Station (or other parts of the MSP infrastructure) mechanisms whereby timer values specified in rules are updated based on estimated propagation skew.

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]. Others have recognized the importance of temporal relationships in detecting security problems [12], but have not addressed the specifics of propagation skew. 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 and an assessment of the algorithm in terms of the probability of a correct result.

One appeal of our approach is that it eliminates the need for manual adjustments of timer values. Further, our studies of a prototype implementation suggest that our approach produces results that are at least as good as an optimal fixed adjustment in 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.

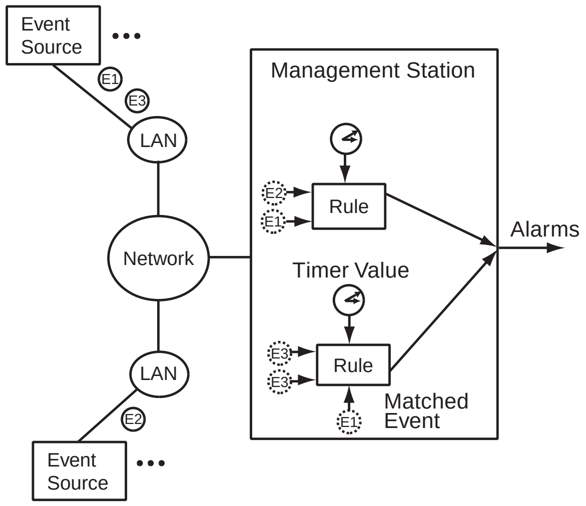

Figure 1. *Figure 1: Architecture of a Management Station that supports temporal event correlation.*

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. There are two types of event sources. The first are events generated by customer-managed elements for which we have no assurance that clocks are synchronized and so timestamps may be inaccurate. The second, which will be described shortly, are MSP elements that are assured to have synchronized clocks and hence accurate timestamps. In the sequel, we simplify matters by assuming that MSP timestamps are applied at the Management Station, although clearly this can be done elsewhere as well. 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 $T_{gen} = t_{last} - t_{first}$.

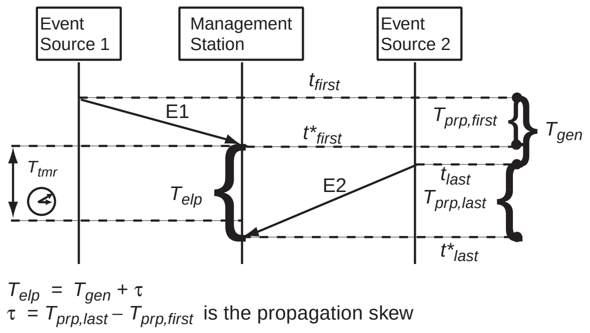

Figure 2. *Figure 2: Interaction diagram for temporal event correlation. Timer values are specified based on the time to generate a pattern T_{gen}, but the estimate of this at the Management Station is T_{elp}.*

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. This compensation is achieved by regulating **slack time**, the time added to timer values 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.

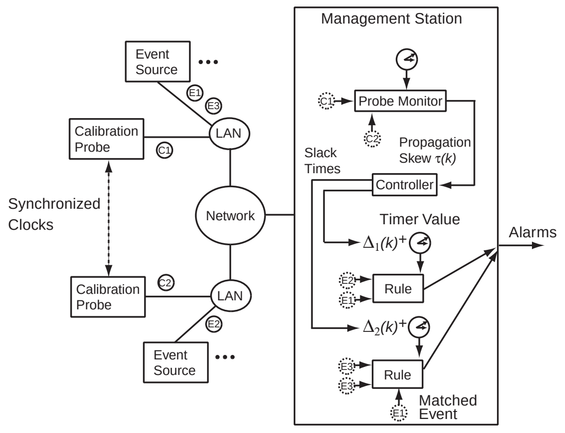

Figure 3. *Figure 3: Architecture that supports compensation for propagation skews by having: (1) Calibration Probes that create Calibration Patterns that have a known pattern generation time; (2) rules that use slack times to adjust timer values; (3) a Probe Monitor that computes propagation skews for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

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

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.

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.

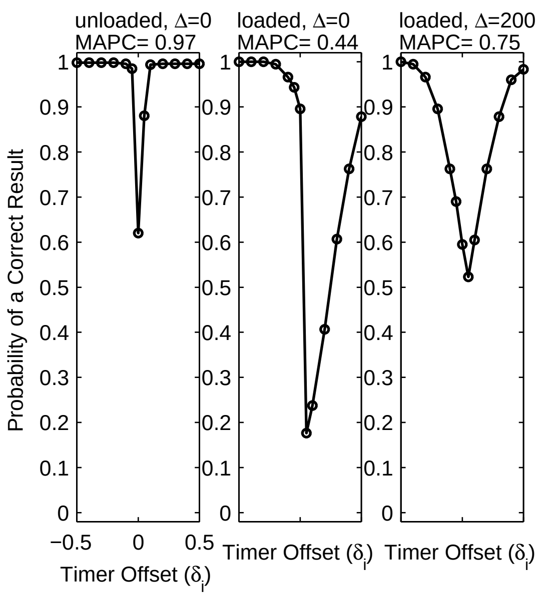

Figure 5. *Figure 5: Probability of a correct result for loaded and unloaded testbed configurations and different fixed settings of slack time (Δ). The horizontal axis is the timer offset, which is the difference between the timer value of a rule and the pattern generation time.*

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 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 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.

$$MAPC = \min[AvgCorrect_{gen}, AvgCorrect_{nogen}] \tag{1}$$

Here, $AvgCorrect_{gen} = Average_{i,k}\{\tau_i(k+1) > \Delta_i(k)\}$, $AvgCorrect_{nogen} = Average_{j,k}\{\tau_j(k+1) < \Delta_j(k)\}$, 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, $P(\text{Correct}) \approx 1$ except at $\delta = 0$, in which case $P(\text{Correct}) \approx 0.5$. Consistent with this, the $MAPC$ is $0.97 \approx 1$. In the second plot, $P(\text{Correct})$ is low for $\delta_i > 0$. As a result, its $MAPC = 0.44$. In the third plot, slack time is adjusted to better balance $P(\text{Correct}|\text{Alarm should be generated})$ and $P(\text{Correct}|\text{Alarm should not be generated})$. 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 $P(\text{Correct }|\text{Alarm should be generated})$ and $P(\text{Correct }|\text{ Alarm should not be generated})$. More specifically, from the first and third plots in Figure 5, we want $P(\text{Correct}) \approx 0.5$ 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

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

```
1. Add $\tau_i(k_i)$ to 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 $i$.
```

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

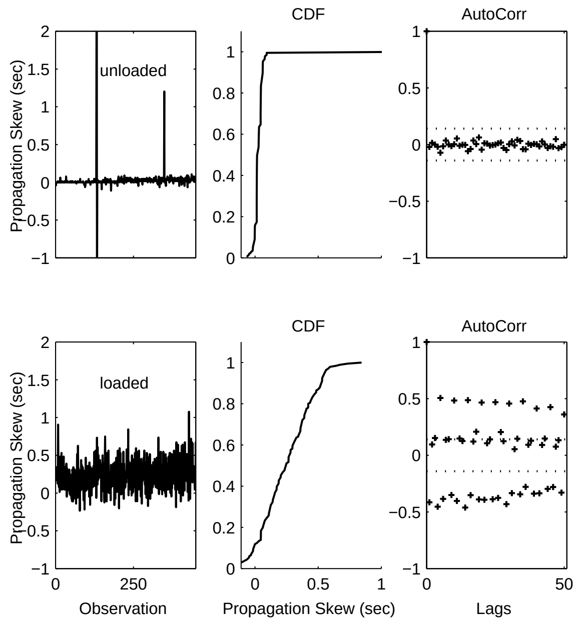

Figure 7. *Figure 7: Characteristics of the propagation skew data for unloaded and loaded configurations. CDF is the empirical cumulative distribution, and AutoCorr is the autocorrelation for a stationary segment of the propagation skews.*

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.

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.

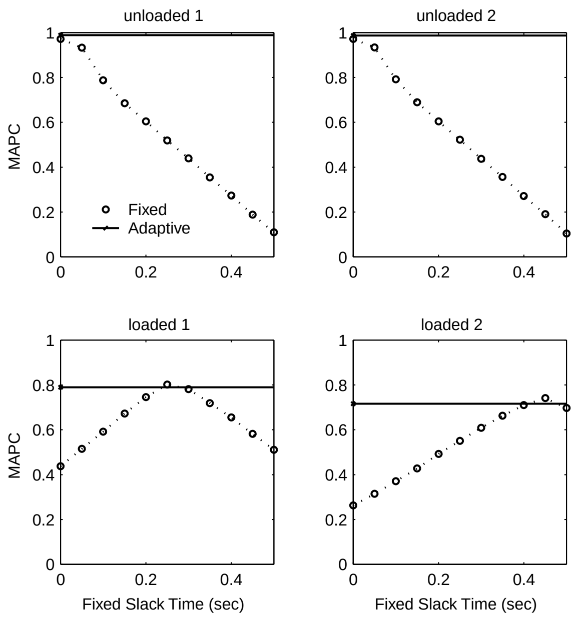

Figure 8. *Figure 8: Evaluation of fixed slack times (dashed line) and the adaptive control algorithm for* stationary *propagation skews. The horizontal axis is the value of the fixed slack time, and the vertical axis is the minimum average probability of a correct test result ($MAPC$). The adaptive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

Figure 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 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

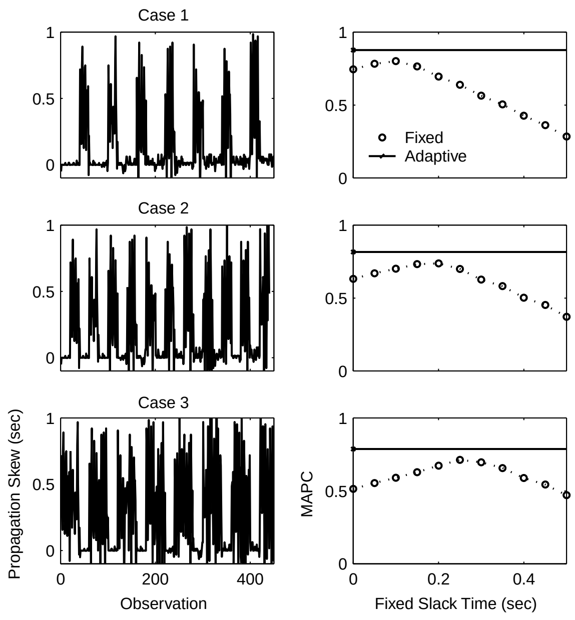

Figure 9. *Figure 9: Evaluation of fixed slack times (dashed line) and the adaptive control algorithm for* non-stationary *propagation skews. The plots in the first column are the propagation skews. The second column are plots that evaluate* $MAPC$ *in the same way as Figure 8.*

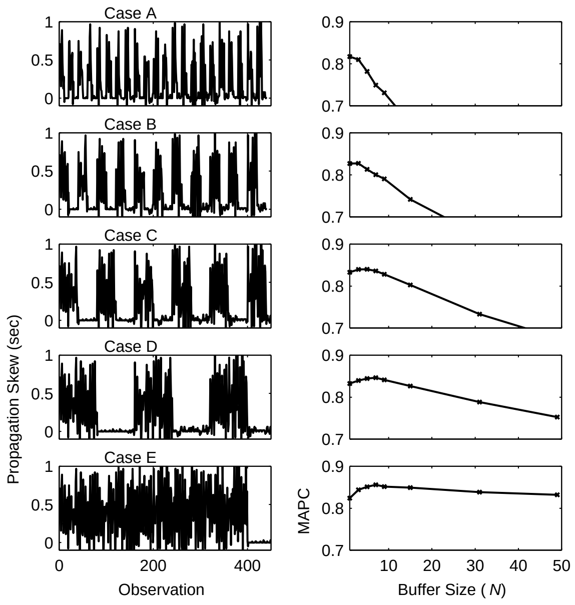

Figure 10. *Figure 10: Effect on MAPC of* N*, the size of the buffer used in the adaptive control algorithm. The plots in the first column are the propagation skews, and the second column plots* $MAPC$ *for different buffer sizes.*

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

Achieving QoS in distributed systems often requires that events be correlated from multiple systems using temporal patterns. This paper addresses how to specify timer values for temporal patterns so as to reduce missed alarms and false alarms caused by lost events and unsynchronized clocks. A central concern is addressing propagation skew, the variability in event propagation times due to contention for network and server resources and other factors. We develop a three part approach to adjusting timer values based on propagation skew: (1) an infrastructure for real-time measurement of propagation skew, (2) a statistical approach to estimating propagation skew, and (3) a controller that uses estimates of propagation skew to update timer values in temporal rules.

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 in that there is no missed alarm or false alarm. Testbed measurements suggest that our 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.

Acknowledgments

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.

[12] Y. Zhang and V. Paxson. Detecting stepping stones. In *9th USENIX Security Symposium*, pages 171–184, August 2000.