IBM Research Report

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

Rean Griffith[1], Joseph Hellerstein[2], Gail Kaiser[1], Yixin Diao[2]

[1]Computer Science Department
Columbia University
New York, NY

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

IBM

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

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

Rean Griffith
Computer Science Department
Columbia University
New York, New York
rg2023@cs.columbia.edu

Joseph Hellerstein
IBM Thomas J. Watson
Research Center
Hawthorne, New York
hellers@us.ibm.com

Gail Kaiser
Computer Science Department
Columbia University
New York, New York
kaiser@cs.columbia.edu

Yixin Diao
IBM Thomas J. Watson
Research Center
Hawthorne, New York
diao@us.ibm.com

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 delays that result from contention for network and server resources. We develop an approach to adjusting the timer values of event correlation rules based on propagation delays in order to reduce missed alarms and false alarms. Our approach has three parts: an infrastructure for real-time measurement of propagation delay, a statistical approach to estimating propagation delays, and a controller that uses estimates of propagation delays 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.**

I. 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, 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 delays, specifically the variability in event propagation delays due to contention for network and server resources and other factors.

Traditionally, event correlation is done using if-then rules (also called event-condition-action). 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 [1]). 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. For temporal rules, the if-part of the rule contains both a pattern that is to be matched by multiple events and 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 [2]). To illustrate, consider a temporal correlation rule that identifies network problems in an application cluster using rules in which ?x denotes variable "x": If there is no `Heartbeat` event from system ?S1 at location ?L1 within 1 minute of another `Heartbeat` event from system ?S2 ≠ ?S1 at location ?L1 and there is a `Heartbeat` event from system S3 at location ?L2 ≠ ?L1, then alert the Network Manager for location ?L1.

Central to temporal event correlation is the concept of **partial correlation instances**, which the refers to the context associated with a partial pattern match. If a partial correlation instance remains uncompleted for a sufficiently long time (specified by the **temporal constraints**), 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 time-out value. A too small time-out value results in 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 is compensating for inaccurate time stamps due to unsynchronized and/or inaccurate clocks in various nodes. One common practice of dealing with inaccurate time stamps is to use the arrival time of the event at the management station. However, 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). Therefore, 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).

Event correlation has been widely used to monitor and analyze networks, systems, and applications for the last twenty years (e.g., [3]). Commonly addressed issues include correla-

tion speed and accuracy [4], [1], [5] and the expressiveness of correlation patterns. For the latter, there has been particular interest in non-rule based approaches [1], probabilistic correlation [6], and temporal patterns [7], [8], [2]. Others have recognized the importance of temporal relationships in detecting security problems [9], 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], [8], [2] mention propagation skew. Hence, none of these systems provide the architectural or algorithmic support needed to compensate for 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 management infrastructure a capability to generate **calibration events** that are representative of general events . Estimation is accomplished by developing a statistical technique that is applied to the time stamps of calibration events. Correction is achieved by including in the Management Station mechanisms whereby timer values specified in rules are updated based on estimated 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 Event Generators, Calibration Event 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.

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

II. ARCHITECTURE

This section describes the architecture of a system that compensates for propagation delays in temporal event correlation for distributed systems. We start by formalizing key concepts used in this paper.

- The **Management Station** is responsible for event correlation such as by using if-then rules and temporal constraints.
- **Events** are messages generated in response to important state changes (e.g., large resource utilizations).
- **Calibration Events** are events generated by special elements called Calibration Event Generators. Sequences of calibration events are used to estimate propagation skew.
- A **Calibration frame** contains two (Calibration Events) events, E_1 and E_2, separated by a known generation time e.g. E_2 is generated 2000 msecs after E_1. The difference between the generation time and the difference in arrival
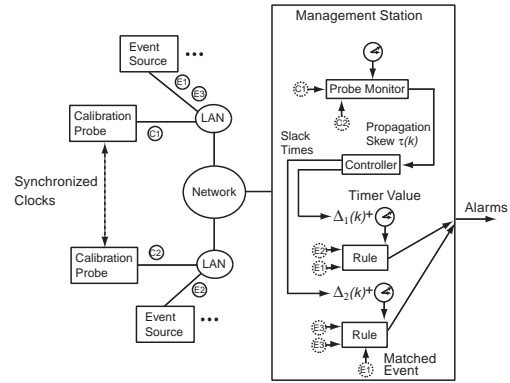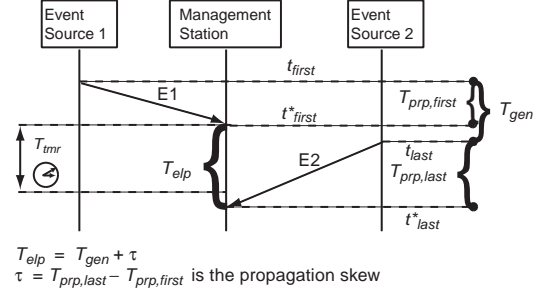

Fig. 1. Architecture that supports compensation for propagation skews.

$T_{elp} = T_{gen} + \tau$
$\tau = T_{prp,last} - T_{prp,first}$ is the propagation skew

Fig. 2. Interaction diagram for temporal event correlation.

times of E_1 and E_2 at the Management Station is the estimate of **propagation skew**.

Figure 1 shows our system architecture. 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 elements such as end-user desktops for which we have no assurance that clocks are synchronized and so time stamps may be inaccurate. The second source of events are management elements that are assured to have synchronized clocks and hence accurate time stamps. We simplify matters by assuming that time stamps from management elements 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 $T_{gen} = t_{last} - t_{first}$. Administrators write rules for temporal correlation based on pattern generation time. 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 1 depicts the way we compensate for propagation skew. This compensation is achieved by regulating **slack time**, the time added to timer values to compensate for propagation skew.

III. 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-parametric 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. We focus on the probability of a correct result (i.e., there is neither an undetected alarm or a false alarm).

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

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

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.

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

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.

There is 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.

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

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.

We compute slack time by using a non-parametric procedure for estimating the median of the distribution of propagation skews [10]. By non-parametric, 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. 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.

IV. EXPERIMENTAL RESULTS

Our testbed follows the architecture depicted in Figure 1. The Management Station employs Columbia University's previously developed temporal event correlation engine, called the Event Distiller [2]. The event transport is University of Colorado's Siena publish/subscribe bus [11]. Three components are deployed in our test-bed: A **Calibration Event Generator** produces pairs of *calibration events* E_1 and E_2, separated by a known pattern-generation time e.g. the E_2 is generated 2000 msecs after E_1. These pairs of events are also known as *calibration frames*. Events E_1 and E_2 contain four important fields: FPResolution – the time (in msecs) that should elapse between the generation of E_1 and E_2. FPSeqNum – a sequence number for a calibration frame. Both E_1 and E_2 in a calibration frame will share the same sequence number. FPStartSeq – a flag, set to one in E_1 indicating the beginning of a calibration frame. In E_2 it is set to zero. FPGenGap – only applicable for the end event, E_2, of a calibration frame, it records the actual time (in msecs) elapsed since the generation of the start event, E_1. It is expected that this value would be close to the FPResolution time depending on the current load of the machine where the Calibration Event Generator runs. Figure 3 shows a pair of calibration events, E_1 and E_2, each calibration event is represented as a Siena Notification [11] of size $\sim$80 bytes.

$E_1 = \{FPGenGap = "0" FPResolution = "2000" FPSeqNum = "1" FPStartSeq = "1" FPTest = "FPTest"\}$
$E_2 = \{FPGenGap = "2041" FPResolution = "2000" FPSeqNum = "1" FPStartSeq = "0" FPTest = "FPTest"\}$

Fig. 3. Calibration Frame

The **Event Distiller** (Management Station) receives calibration frames and records an arrival time stamp on each event comprising the calibration frame. The difference in the arrival times of E_2 and E_1 is compared to the FPResolution, adjusted based on FPGenGap if necessary, and used to estimate the propagation skew/delays in receiving pairs of calibration events. As an example, a calibration frame with FPResolution = 2000 msecs and FPGenGap = 2005 msecs indicates that the end event, E_2, of a calibration frame was delayed by 5 msecs. On the receiver end, we adjust the difference in arrival times by 5 msecs to compensate for the delay in event generation. Whereas it is possible for events generated $T_{gen} + \epsilon$ to have a difference in arrival times of T_{gen}, due to delays/congestion in the network or packet processing delays on the receiver end,

we take the conservative approach of adjusting arrival times at the receiver end primarily to mitigate any scheduling/load issues at the sender that may have delayed event generation.

Finally, a **Siena Event Router** is responsible for managing client subscriptions, receiving published events and routing them (based on their contents) to interested subscribers. The Event Distiller, is an example an interested subscriber of calibration events.

We study four configurations of these three components running on a mix of Windows XP and Linux platforms. A total of four machines in our CS network are used; Kathmandu.clic, Lisbon.clic, Amman.clic and Liberty.psl. Kathmandu, Lisbon and Amman each have a single 3.2 GHz Intel Pentium 4 processor, 1 GB RAM running a 2.6.9-22.0.2.EL Linux kernel. Liberty is a 3 GHz Pentium 4 with 1 GB RAM running Windows XP SP2.

Configuration A is a mixed-platform 3-machine configuration. The Calibration Event Generator runs on the Linux host, Kathmandu.clic, the Siena Event Router runs on the Linux host, Lisbon.clic, while the Event Distiller runs on the Windows XP host, Liberty.psl. **Configuration B** is a homogeneous-platform 3-machine configuration. The Calibration Event Generator, Siena Event Router and Event Distiller run on Linux hosts Kathmandu.clic, Lisbon.clic and Amman.clic respectively. Configurations C and D are 2-machine configurations where the Siena Router and Event Distiller are collocated on the same host. Collocation of the Event Distiller and the Siena Event Router is intended to mimic situations where there is contention for machine resources such as CPU, memory and/or network resources at the management station. **Configuration C** is a mixed-platform 2-machine configuration where the Calibration Event Generator runs on the Linux host, Kathmandu.clic, and the Siena Event Router and Event Distiller both run on the Windows XP host, Liberty.psl. **Configuration D** is a homogeneous-platform 2-machine configuration where the Calibration Event Generation runs on the Linux host, Kathmandu.clic, and the Siena Event Router and Event Distiller both run on the Linux host, Lisbon.clic. For each configuration all machines were located on the same campus LAN and exhibited ping times on the order of <1ms for 32 bytes of data between machines.

In our experiments we observed large variations in the propagation skews measured in configuration C as compared to those measured in configurations A, B and D, Figure 4. It seems counter-intuitive that propagation skews would be larger for a 2-machine configuration than for a 3-machine configuration. Our initial conjecture was that using the difference in arrival times of calibration events at the Event Distiller captures more than network delays. This time difference may also be influenced by contention for shared resources such as the CPU and network I/O stack, which the collocated Siena Event Router and Event Distiller compete for.

We considered it improbable that the relatively large propagation skew values observed in configuration C could be attributed to an extremely inefficient mechanism within Siena for delivering events to a local subscriber, especially when Siena is
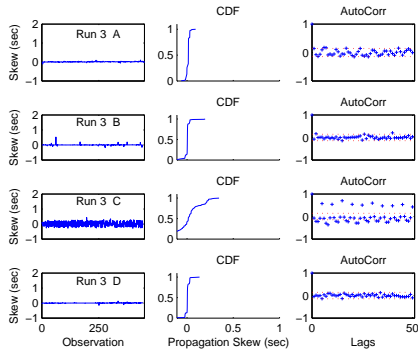
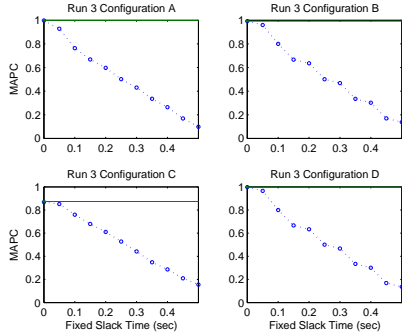Fig. 4. Characteristics of propagation skew data.

Fig. 5. Evaluation of fixed slack times (dashed line) and the adaptive control algorithm.

intended to be a scalable, wide-area event notification service [11]. Based on a side-by-side comparison of the 2-machine configurations, C and D, where the Siena Event Router and Event Distiller are collocated on a Windows XP machine and a Linux machine respectively we conclude that variations in propagation skew are more pronounced under Windows XP than under Linux and our measure of propagation skew is also influenced by resource contention/the current workload on the machine running the Event Distiller.

Figure 4 reports data from configurations A through D. In configurations A, B and D, 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 according to the Bartlett Test [12]. In 2-machine configuration C, propagation skews are much more variable and considerably larger. We also see substantial autocorrelations, possibly due to periodic activities and/or resource contention.

Figure 5 assesses the effectiveness of using fixed slack times. The horizontal axis is the slack time Δ and the vertical axis is Minimum Average Probability of a Correct result ($MAPC$). Large $MAPC$ values are achieved with a fixed slack time near 0 in configurations A, B and D. However, for the 2-machine configuration C, $MAPC$ is maximized at larger fixed slack times. This can be explained by looking

at the distribution of propagation delays. The solid line in Figure 5 plots the $MAPC$ values achieved by our adaptive control algorithm.In all cases, the algorithm selects slack times 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, i.e., slack times are selected in a self-managing manner. Second, we achieve near-optimal results in the 2-machine configurations, even though the data may have significant autocorrelations.

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

REFERENCES

[1] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, "High speed and robust event correlation," *IEEE Communications Magazine*, vol. 34, no. 5, pp. 82–90, 1996.

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

[3] 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*, vol. 25, no. 2, 1986.

[4] G. Jiang and G. Cybenko, "Temporal and spatial distributed event correlation for network security." [Online]. Available: http://www.ists.dartmouth.edu/ISTS/library/infrastructure-security/tsd0704.pdf

[5] O. C. O. Systems, "Rootcause: Using a flight recorder to speed remote debugging and problem resolution." [Online]. Available: http://www.ocsystems.com/rootcause_white_paper.html

[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*, 1st ed. 75 Arlington Street,Suite 300, Boston, MA 02116: Addison–Wesley, 2002.

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

[9] Y. Zhang and V. Paxson, "Detecting stepping stones," in *9th USENIX Security Symposium*, August 2000, pp. 171–184.

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

[11] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service." *ACM Trans. Comput. Syst.*, vol. 19, no. 3, pp. 332–383, 2001.

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