Two Dimensional Time-Series for Anomaly Detection and Regulation in Adaptive Systems

Mark Burgess

Faculty of Engineering, Oslo University College, Cort Adelers Gate 30, N-0254 Oslo, Norway
http://www.iu.hio.no/~mark, mark@iu.hio.no

Abstract. A two dimensional time approach is introduced in order to classify a periodic, adaptive threshold for service level anomaly detection. An iterative algorithm is applied to history analysis on this periodic time to provide a the smooth roll-off in the significance of the data with time. The algorithm described leads to an approximately ten-fold compression in data storage, and thousand fold improvement in computation cycles, compared to a naive time-series approach. The behaviour of this anomaly detector is discussed, and the result is implemented in cfengine for direct use in system management.

1 Introduction

Anomaly detection is usually performed using one of two techniques: off-line time-series analysis, or real-time event threshold processing. Although the current preference in the lierature is to look for anomalies in network behaviour, recently the trend has moved towards host-based anomaly detection. In the present work, anomalous behaviour is not presupposed to come from any particular source, but to be a potential concern, whatever its origin. This paper describes a method for detecting anomalous behaviour on a single host, with the aim of using the information for self-regulation, by initiating a counter-response.

The need to define the normal state of a computer system arises in several contexts. Traditionally, measurements of normal computer behaviour have been used to analyse event arrival times and lifetimes; they have often collected in connection with performance analyses[1,2]. Other studies of computer systems have been performed in connection with load balancing, expectations of communications over a network and interactions with users on teletype terminals. Security intrusions have long motivated discussions of anomalies also. Indeed, this has led to a tacit equivalence between the term anomaly detection and network packet analysis, which is unjustified.

Automated self-regulation in host management has been discussed in refs [3,4,5,6], as well as adaptive behaviour[7] and network intrusion detection[8,9]. Some authors have likened such mechanisms to the need for a generic immune system, striking the analogy between computers and other collective systems in sociology and biology[10,11,9,5]. In their insightful paper[12], Hoogenboom and

Lepreau anticipated the need for monitoring times series data with feedback regulation, before this idea was popular. Today much effort is aimed at detecting anomalies for security related intrusion detection rather than for general maintenance, or capacity planning. This has focused attention on mainly short term changes; however, long term changes can also be of interest in connection with maintenance of host state and its adaptability to changing demand.

The present work was performed in order to implement long term anomaly detection into the configuration agent system cfengine[3,13]. The result is a system which can detect signs of distress or unusual lethargy in the operation of a host, and allow the user to attach policy decisions to such conditions.

2 Adaptive Behaviour

The complexity of modern computer systems makes them best likened to simple organisms rather than mechanical tools: they live in a complex environment, provided by the network and by contact with users. In order to function effectively in such an environment, complex systems have to adapt. This has been discussed by several authors in the context of self adapting operating systems[7,5,14]. Other discussions of adaptive behaviour have occurred in relation to security[15,16].

In order to adapt a system to the conditions in the environment, one needs an idea of what state is optimal for a given environment. Such a notion can be defined by a principle of minimization or maximization, using an arbitrary criterion. Since computer systems are expected to fulfil a function other than simply adapting, it is natural to choose this criterion to be the local system policy of the host[6]. The policy reflects the purpose of the system.

Adapative behaviour, varying about a local minimum of policy-conformant behaviour, can lead to stable behaviour by introducing feedback between host state and environment. This requires a more detailed knowledge of how the system has been changing in the past. There are two basic kinds of change which are important to consider: a non-equilibrium change (slow, progressive variation) or steady state (equilibrum or limit cycle). If the system is approximately stable, then either of these can be characterized by the recent history of the system. This can be measured as a time series and analyzed[12] in order to provide the necessary information.

Time-series data consume a lot of space however, and the subsequent calculation of the ensemble averages costs a considerable amount of CPU time as the window of measurement increases. In this paper, it is shown how an approximately tenfold compression of the data can be achieved, and several orders of magnitude of computation time can be spared by the use of a random access database. The key to such a compression is to update a sample of data iteratively rather than using an off-line analysis based on a complete record. This means collecting data for each time interval, reading the database for the same interval and combining these values in order to update the database directly. The database can be made to store the average and variance of the data directly, for a fixed window, in this manner without having to retain each measurement individually.

An iterative method can be used, provided such iteration provides a good approximation to a regular sliding window, time-series data sample[17]. One obvious approach here is to use a convergent geometric series in order to define an average which degrades the importance of data over time. After a certain interval, the oldest points contribute only an insignificant fraction to the actual values, provided the series converges. This does not lead to a result which is identical to an offline analysis, but the offline analysis is neither unique nor necessarily optimal. What one can say however, is that the difference between the two is within acceptable bounds and the resulting average has many desirable properties. Indeed, the basic notion of convergence closely related to the idea of stability[5], so this choice is appropriate.

3 Two Dimensional Time

The approximate periodicity observed in computer resources allows one to parameterize time in topological slices of period P, using the relation

$$t = nP + \tau. \tag{1}$$

This means that time becomes cylindrical, parameterized by two interleaved coordinates (τ, n), both of which are discrete in practice. This parameterization of time means that measured values are multi-valued on over the period $0 \geq \tau < P$, and thus one can average the values at each point τ, leading to a mean and standard deviation of points. Both the mean and standard deviations are thus functions of τ, and the latter plays the role of a scale for fluctutations at τ, which can be used to grade their significance.

The cylindrical parameterization also enables one to invoke a compression algorithm on the data, so that one never needs to record more data points than exist within a single period. It thus becomes a far less resource intensive proposition to monitor system normalcy.

The desired average behaviour can be stored indefinitely by using a simple database format, covering only a single working week. In order to generate sequences it is not necessary to know all the details of past behaviour, only their sum effect is required. A fixed interval time-based coding is sufficient. Previous work on anomalies at the user level[17] shows that the auto-correlation times for significant changes, in user level behaviour, are of the order of five minutes or longer, thus a time granularity of five minute intervals is sufficient for the database. Values can therefore be stored in a database with key values of the form

```
Day:Hour:Minute_interval
```

For instance:

```
Mon:Hr15:Min00_05
Tue:Hr01:Min05_10
```

Since it is only necessary to record a complete week's worth of data, the values can be compressed into just a few megabytes. The data are then stored using the fast Berkeley database format[18]. The test set was taken to be a number of universal and easily measureable charaters:

- Number of users.
- Number of privileged processes.
- Number of non-privileged processes.
- Amount of free disk.
- Number of incoming sockets for a variety of well know services.
- Number of outgoing sockets from the same services.

These variables have been examined earlier and their behaviour is explained in[4, 17].

4 Computing Averages with Roll-Off

To maintain the database of averages and variances, an algorithm is required, satisfying the following properties:

- It should approximate an offline sliding-window time-series analysis over a weekly interval[17].
- It should present a minimal load to the system concerned.
- It must have a predictable error or uncertainty margin.

These goals can be accomplished straightforwardly as follows.

An iteration of the update procedure may be defined as the combination of a new data point q with the old estimate of the average $\overline{q}$.

$$\overline{q} \to \overline{q}' = (q|\overline{q}) \tag{2}$$

where

$$(q|\overline{q}) = \frac{w\,q_1 + \overline{w}\,\overline{q}}{w + \overline{w}}. \tag{3}$$

This is somewhat analogous to a Bayesian probability flow. The repeated iteration of this expression leads to a geometric progression in the parameter $\lambda = \overline{w}/(w+\overline{w})$:

$$\begin{aligned}(q_1|(q_2|\ldots(q_r|(\ldots|\overline{q}_n)))) = \frac{w}{w+\overline{w}}\,q_1 + \frac{\overline{w}w}{(w+\overline{w})^2}\,q_2 + \\ + \ldots \\ + \frac{w\,\overline{w}^{r-1}}{(w+\overline{w})^r}\,q_r + \ldots \frac{\overline{w}^n}{(w+\overline{w})^n}\,\overline{q}_n.\end{aligned} \tag{4}$$

Thus on each iteration, the importance of previous contributions is degraded by λ. If we require a fixed window of size N iterations, then λ can be chosen in such a way that, after N iterations, the initial estimate q_N is so demoted as to be insignificant, at the level of accuracy required. For instance, an order of magnitude drop within N steps means that $\lambda \sim |10^{-N}|$.

In order to approximately preserve the magntiude of the average value while the iterations are growing, N should be increased in step with the iterations up to the fixed window size and then held constant once this is achieved.

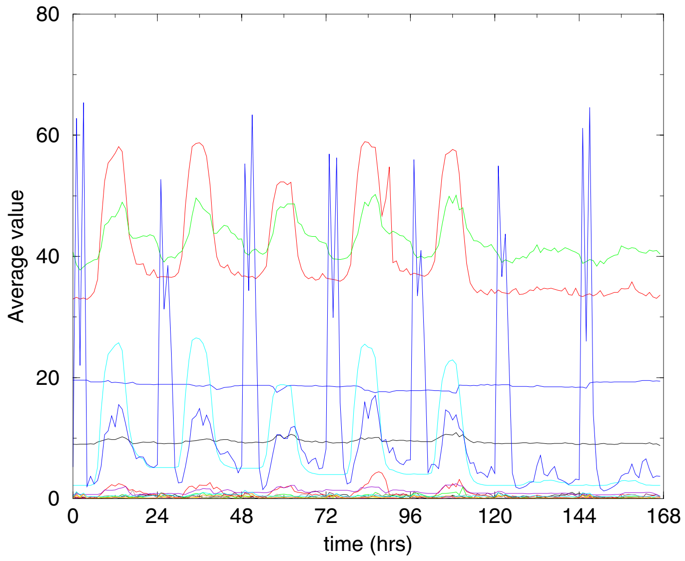

Fig. 1. A periodogram of some resource variables, scaled and smoothed, taken from cfengine. The lines show the effective average thresholds for normal behaviour. Note how each line has its own characteristic pattern of usage, which the system learns by empirical measurement.

Using the definition of the pseudo-fixed-window average $\langle ... \rangle_N$, we may now define the average standard deviation, or error, by

$$\begin{aligned} \langle \sigma(\tau) \rangle &\equiv \sqrt{\langle (\delta q(\tau))^2 \rangle_N} \\ \delta q(\tau) &= q(t) - \langle q(\tau) \rangle_N \end{aligned} \tag{5}$$

This has similar properties to the degrading average itself, though the square root makes the accuracy more sensitive to change. Based on these two definitions, one now sees what data must be stored in the database over a measurement period.

```
double q_bar[i];
double delta_q_bar[i];
double time_index;
```

The computation of comparisons should always be performed with dimensionless variables, $\delta q(t)/\langle \sigma(t) \rangle_N$.

It should be noted that, while a window size of several months is required to establish limiting behaviour, once the nature of the limiting behaviour has been established, it is no longer a requirement that this amount of data be used for comparison. Indeed, the window size of several months is clearly too large to see useful changes clearly over shorter times.

A compromise of two weeks is used here. This will not lead to a smooth fit of the data, but this is not required, since the object of the classification is to coarse grain (digitize) at only medium resolution.

Fine enough to see large sudden changes, coarse enough to be stable over small variations.

In order to satsify the requirements of a decaying window average, with determined sensitivity $\alpha \sim 1/N$, we require,

1. $\frac{w}{w+\overline{w}} \sim \alpha$, or $w \sim \overline{w}/N$.
2. $\left(\frac{\overline{w}}{w+\overline{w}}\right)^N \ll \frac{1}{N}$, or $\overline{w}N \ll w$.

Consider the ansatz $w = 1 - r$, $\overline{w} = r$, and the accuracy α. We wish to solve

$$r^N = \alpha \tag{6}$$

for N. With $r = 0.6, \alpha = 0.01$, we have $N = 5.5$. Thus, if we consider the weekly update over 5 weeks (a month), then the importance of month old data will have fallen to one hundredth. This is a little too quick, since a month of fairly constant data is required to find a stable average. Taking $r = 0.7, \alpha = 0.01$, gives $N = 13$. Based on experience with offline analysis, this is a reasonable arbitrary value to choose.

5 Characterizing Pseudo-Periodic Functions

The recent behaviour of a computer can be summarized by non-Markovian processes, during periods of change, and by hidden Markov models during steady state behaviour, but one still requires a parameterization for data points. Such models must be formulated on a periodic background[19], owing the importance of periodic behaviour of users. The precise algorithm for averaging and local coarse-graining is somewhat subtle, and involves naturally orthogonal time dimensions which are extracted from the coding of the database. It is discussed here using an ergodic principle: a bi-dimensional smoothing is implemented, allowing twice the support normally possible for the average, given a number of data points. This provides good security against "false positive" anomalies and other noise.

Consider a pseudo-periodic function, with pseudo-period P,

$$\begin{aligned} q(t) &= \sum_{n=0}^{\infty} q(nP + \tau) \qquad\qquad (0 \leq \tau < P) \\ &= \sum_{n=0}^{\infty} \chi_n(\tau). \end{aligned} \tag{7}$$

The time coordinate τ lives on the circular dimension. In practice, it is measured in p discrete time-intervals $\tau = \{\tau_1, \tau_2, \ldots \tau - p\}$. In this decomposition, time is a two-dimensional quantity. There are thus two kinds of sliding average which can be computed: average over corresponding times in different periods (topological average $\langle\chi(\tau)\rangle_T$), and average of neighbouring times in a single period (local average $\langle\chi(\tau)\rangle_P$):

$$\langle\chi(\tau)\rangle_T \equiv \frac{1}{T}\sum_{n=l}^{l+T}\chi_n(\tau)$$

$$\langle\chi(n)\rangle_P \equiv \frac{1}{P}\sum_{\ell=\tau}^{\tau+P}\chi_n(\ell) \tag{8}$$

where P, T are integer intervals for the averages, in the two time-like directions. Within each interval that defines an average, there is a corresponding defintion of the variation and standard deviation, at a point τ:

$$\sigma_T(\tau) \equiv \sqrt{\frac{1}{T}\sum_{n=l}^{n=l+T}\left(\chi_n(\tau) - \langle\chi(\tau)\rangle_T\right)^2}$$

$$\sigma_P(n) \equiv \sqrt{\frac{1}{P}\sum_{\ell=\tau}^{\ell=\tau+P}\left(\chi_n(\ell) - \langle\chi(\ell)\rangle_P\right)^2}. \tag{9}$$

Sliding window versions of these may also be defined, straightforwardly from the preceding section:

$$\overline{\chi}(n)_P \equiv (\chi|\overline{\chi}(n)_P)$$

$$\overline{\chi}(\tau)_T \equiv (\chi|\overline{\chi}(\tau)_T). \tag{10}$$

Here one simply replaces the evenly weighted sum, with an iteratively weighted sum. The difference between the average and the current value is expressed by the delta symbol $\delta_T\chi = \chi - \langle\chi\rangle_T$ etc.

Using these averages and deviation criteria, we have a two-dimensional criterion for normalcy, which serves as a control at two different time-scales. One thus defines normal behaviour as

$$\{\delta_T\chi(\tau), \delta_P\chi(n)\} < \{2\sigma_T(\tau), 2\sigma_P(n)\}. \tag{11}$$

These may be simply expressed in geometrical, dimensionless form

$$\varDelta(\tau, n) = \sqrt{\left(\frac{\delta_T\chi(\tau)}{\sigma_T(\tau)}\right)^2 + \left(\frac{\delta_P\chi(n)}{\sigma_P(n)}\right)^2}, \tag{12}$$

and we may classify the deviations accordingly into concentric, elliptical regions:

$$\delta(\tau, n) < \begin{cases} \sqrt{2} \\ 2\sqrt{2} \\ 3\sqrt{2} \end{cases}, \tag{13}$$

for all τ, n. which indicate the severity of the deviation, in this parameterization. This is the form used by cfengine's environment engine.

6 Experiences

The foregoing algorithm has been incorporated into the configuration agent cfengine, in order to provide a framework for testing its utility, and to gain experience with the use of average-state motivated responses. Cfengine is used on hundreds of thousands of systems all over the world, and is therefore an ideal testing ground for this. To incorporate the state information into cfengine, additional classes are automatically evaluated based on this average state of the host. This is accomplished by an additional *cfenvd* daemon, which continually updates the database of system averages, characterizing "normal" behaviour. The state of the system is examined and compared to the database, and the state is classified in coarse terms by comparing to average of equivalent earlier times. Some examples of classes which can become active in the cfagent:

```
RootProcs_low_dev2
netbiosssn_in_low_dev2
smtp_out_high_anomalous
www_in_high_dev3
```

The first of these tells us that the number of root processes is two standard deviations below the average for past behaviour. This might be fortuitous, or might signify a problem, such as a crashed server; we do not know the reason, only that an anomaly has occurred. The WWW item tells us that the number of incoming connections is three standard deviations above average. The smtp item tells us that the number of outgoing smtp connections is more than three standard deviations above average, perhaps signifying a mail flood. The setting of these classes is transparent to the user, but the additional information is only visible to the privileged owner of the cfengine work-directory, where the data are cached.

The notion of useful classifications will no doubt change as more experience is gained in their usage. For now, it remains to test these in combination with policy specifications. For instance, if the number of incoming SMTP connections suddenly became several standard deviations above normal, for the given time of week, the mail service could simply be shut down temporarily to avert a possible spam attack. Similarly other services could be revoked if overused. A low level of activity could be indicative of problems elsewhere. It remains to be seen exactly how these state classifications will be employed in site policy.

Using the environment daemon, we have seen anomalies mainly in well-known services such as HTTP and SMTP. Occasionally user processes peak in connection with service anomalies. A familar pattern that has been observed on numerous occasions is the Web-server scan for E-mail addresses, followed an SMTP peak during which spam-mail is sent to the addresses. Outgoing spam attacks by students have also been detected as a combination of outgoing SMTP and user processes.

Environmentally adaptive policy specification is an enticing prospect, particularly from a security standpoint, however, initial tests indicate that the tested averages are often too sensitive to be reliable guides to behaviour on hosts which are only used casually, e.g. desktop workstations. A single standard deviation is often not even resolvable on a lightly used host, i.e. it is less than the discrete nature of a single event; the appearence of a single new login might trigger a twice standard deviation from the norm. On more heavily loaded hosts, with persistent loading, more reliable measures of normality can be obtained, and the measures could be useful. Anomalies in widely used services, such as SMTP, HTTP and NFS are detectable. However, in tests they have only been short-lived events.

A question of the utmost significance is here is 'when is an anomaly worth reacting to'? The anomalies considered here are resource usage anomalies, which detect essentially 'spam' type events on a system. They can be used to detect certain events which are of interest to system management and, if sufficiently rapid detection is enabled, they could be used to trigger automatic defenses, such as temporary service denial. If that is to be the case, the agent responding (cfagent, for example) would have to answer the anomaly within only a few minutes in order to be effective. This is more often than most agent systems are run. One advantage with cfengine in this regard is that it *can* be run very frequently without 'spamming' the system itself, due to its time-limit locking and scheduling policies.

To date, only a handful of variables, based on resource usage, has been considered for test purposes. The same technique can be applied to other types of event such as symbolic string sequences; e.g. active process names, sequences of system calls or intrusion detection monitors, such as snort[20]. Tests using these types of events are being tested now. Sequences of system calls have already been analyzed by Forrest et. al in ref. [14] in order to provide a generic first defence against the occurrence of unknown sequences.

7 Comparison with Offline Analysis and Anomalies

The real time compression algorithm presented performs impressively, in terms of the resources spared, but the question remains as to how reliable it is compared to a full off-line time-series analysis. Since there is no right or wrong way to do anomaly detection, there is no question that the two methods should agree exactly. At the very least, however, one would like to see a qualitative agreement between the two very different methods.

Figure 2 shows detailed, high resolution averages for process count data on a given host, over an interval of two months. At this level of detail one sees a jagged curve with error bars which vary considerably in magntitude. Figure 3 shows an image created using the iterative algorithm, on the same data, at the same resolution. The main features of the curve are still visible. Error bars have been suppressed to avoid clutter.

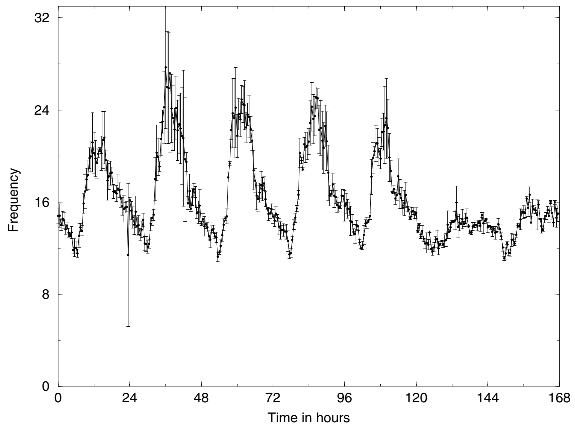

Fig. 2. A high resolution plot of process count behaviour, using the time-series sliding window. This may be taken as a reference point. Data size is 21114875 bytes

These data show that a high resolution viewpoint is too jittery to form a sensible guide as an anomaly threshold. A low resolution version of these graphs is calculated by locally averaging over the two time dimensions. This double averaging leads to improved 'false positive' rejection.

The greatest problem for anomaly detection is in dealing with periods of lowactivity. In a period of low activity, every event is "abnormal". These cold-spots pull the averages down and over-sensitize to the detection of random events. In such a case, only a policy decision can renormalize the threshold level to avoid the detection of events with no significance.

In cases of oversensitivity, one can easily introduce an optimization in the following manner. Thresholds can be rounded up to the nearest disrecte value, and those which are of the order unity can be ignored completely. Another approach, which has additional benefits, would be to measure not only values but rates of change in the data; it is known that both values and rates of change are needed to characterize any dynamical behaviour. In measuring the data (e.g. numbers of processes) one assumes that a resolution of one process is meaningful. However, many programs start multiple processes (window manager logins start maybe twenty processes at a time), so that the effective resolution is often far less. With the approach presented here, each such change would be lead to a deviation which was several standard deviations. One would never see small changes. This is clearly a problem to be addressed. It is, however, a problem with the present method illuminates clearly.

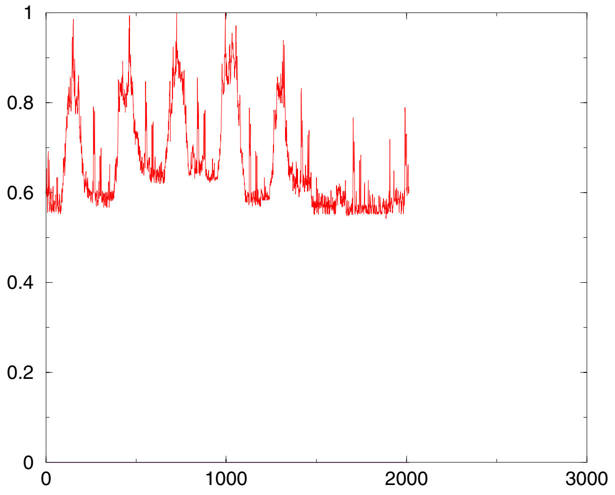

Fig. 3. A high resolution plot using the iterative algorithm using the the same process count data as in figure 2. Data size is 1531904 bytes (13.8 times smaller).

8 Conclusions

A two dimensional time slice approach used for anomaly detection has been presented. The approach lends itself to an iterative algorithm, which has been encoded into the environment daemon in cfengine. The daemon collects results both qualitatively and quantitatively similar to off-line studies previous made using regular time-series methods, for a fraction of the resources. This work therefore provides proof of concept for the algorithm, in a form well suited for immediate use by adaptive policy engines.

The final aim of this research is to have a turn-key, plug'n'play solution to this problem of anomaly detection, into which users need only insert their policy requirements. While a suitable framework has been created, it will likely be several years before a working solution can be considered adequate for adaptive regulation. The issue of what to do when an anomaly is detected is also a rather important one, which will require more ingenuity than technology. Another challenge which remains is to extend this notion of compressed, long-term memory to include discrete pattern events, such as those generated by intrusion detection software (e.g. snort[20] and pH[14]). This is likely possible using a spectral approach, combined with a small-fragment associative-recognition database. Amusingly, this is somewhat analogous to the current understanding of the biochemistry of the sense of smell.

Availability. GNU Cfengine may be obtained from http://www.cfengine.org.

References

1. J.L. Hellerstein, F. Zhang, and P. Shahabuddin. An approach to predictive detection for service management. *Proceedings of IFIP/IEEE INM VI*, page 309, 1999.
2. J. Cradley Chen, Y. Endo, D. Mazieres, A. Dias, M. Seltzer, and M.D. Smith. The measured performance of personal computer operating systems. *ACM transactions on computing systems and Proceedings of the 15th ACM symposium on Operating System Principles*, 1995.
3. M. Burgess. A site configuration engine. *Computing systems (MIT Press: Cambridge MA)*, 8:309, 1995.
4. M. Burgess. Automated system administration with feedback regulation. *Software practice and experience*, 28:1519, 1998.
5. M. Burgess. Computer immunology. *Proceedings of the Twelth Systems Administration Conference (LISA XII) (USENIX Association: Berkeley, CA)*, page 283, 1998.
6. M. Burgess. Theoretical system administration. *Proceedings of the Fourteenth Systems Administration Conference (LISA XIV) (USENIX Association: Berkeley, CA)*, page 1, 2000.
7. M.I. Seltzer and C. Small. Self-monitoring and self-adapting operating systems. *Proceedings of the Sixth workshop on Hot Topics in Operating Systems*, 1997.
8. M.J. Ranum et al. Implementing a generalized tool for network monitoring. *Proceedings of the Eleventh Systems Administration Conference (LISA XI) (USENIX Association: Berkeley, CA)*, page 1, 1997.
9. S. A. Hofmeyr, S. Forrest, and P. D'haeseleer. An immunological approach to distributed network intrusion detection. *Paper presented at RAID'98 - First International Workshop on the Recent Advances in Intrusion Detection Louvain-la-Neuve, Belgium September.*, 1998.
10. J.O. Kephart. A biologically inspired immune system for computers. *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems. MIT Press. Cambridge MA.*, page 130, 1994.
11. S. Forrest, S. Hofmeyr, and A. Somayaji. *Communications of the ACM*, **40**:88, 1997.
12. P. Hoogenboom and J. Lepreau. Computer system performance problem detection using time series models. *Proceedings of the USENIX Technical Conference, (USENIX Association: Berkeley, CA)*, page 15, 1993.
13. M. Burgess. Cfengine www site. *http://www.iu.hio.no/cfengine.*
14. S. A. Hofmeyr, A. Somayaji, and S.Forrest. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.
15. M. Burgess and D. Skipitaris. Adaptive locks for frequently scheduled tasks with unpredictable runtimes. *Proceedings of the Eleventh Systems Administration Conference (LISA XI) (USENIX Association: Berkeley, CA)*, page 113, 1997.
16. M. Carney and B. Loe. A comparison of methods for implementing adaptive security policies. *Proceedings of the 7th security conference (USENIX Association: Berkeley, CA).*
17. M. Burgess, H. Haugerud, T. Reitan, and S. Straumsnes. Measuring host normality. *ACM Transactions on Computing Systems*, 20:125–160, 2001.
18. Sleepcat Berkeley db project. *http://www.sleepycat.com.*
19. M. Burgess. The kinematics of distributed computer transactions. *International Journal of Modern Physics*, **C**12:759–789, 2000.
20. Snort. Intrusion detection system. *http://www.snort.org.*