Analysis of Software Aging in a Web Server

Michael Grottke[†][*], Lei Li[‡], Kalyanaraman Vaidyanathan[♯], and Kishor S. Trivedi[†]

[†]Department of Electrical and Computer Engineering
Duke University
Durham, NC 27708, USA
{grottke,kst}@ee.duke.edu

[‡]Wireless & Mobile Systems Group
Freescale Semiconductor, Inc.
Austin, TX 78735, USA
leili@freescale.com

[♯]RAS Computer Analysis Laboratory
Sun Microsystems
San Diego, CA 92121, USA
kalyan.vaidyanathan@sun.com

Abstract

A number of recent studies have reported the phenomenon of "software aging", characterized by progressive performance degradation and/or an increased occurrence rate of hang/crash failures of a software system due to the exhaustion of operating system resources or the accumulation of errors. To counteract this phenomenon, a proactive technique called "software rejuvenation" has been proposed. It essentially involves stopping the running software, cleaning its internal state and/or its environment and then restarting it. Software rejuvenation, being preventive in nature, begs the question as to when to schedule it. Periodic rejuvenation, while straightforward to implement, may not yield the best results, because the rate at which software ages is not constant, but it depends on the time-varying

[*]Corresponding author, on leave of absence from the Chair of Statistics and Econometrics, University of Erlangen-Nuremberg, Germany.

system workload. Software rejuvenation should therefore be planned and initiated in the face of the actual system behavior. This requires the measurement, analysis and prediction of system resource usage.

In this paper, we study the development of resource usage in a web server while subjecting it to an artificial workload. We first collect data on several system resource usage and activity parameters. Non-parametric statistical methods are then applied for detecting and estimating trends in the data sets. Finally, we fit time series models to the data collected. Unlike the models used previously in the research on software aging, these time series models allow for seasonal patterns, and we show how the exploitation of the seasonal variation can help in adequately predicting the future resource usage. Based on the models employed here, proactive management techniques like software rejuvenation triggered by actual measurements can be built.

Index terms: Software aging, software rejuvenation, Linux, Apache, web server, performance monitoring, prediction of resource utilization, non-parametric trend analysis, time series analysis

1 Introduction

It has now been well established that software faults are the major cause of computer system failures [19, 33]. While there are many tools and techniques for supporting software developers and testers it is practically impossible to guarantee that software products do not contain any residual faults at the time of their release.

There are two main approaches to coping with the unavoidable presence of software faults. On the one hand, one can strive for efficient ways of recovering the software system *after* a failure has occurred, e.g. via fine-grained "microreboots" [7] of only the affected application components.

On the other hand, if the failure rate of software should be increasing, it can be worthwhile to implement some sort of *preventive* maintenance. Indeed, researchers have recently reported and studied the fact that software applications executing continuously for a long period of time show a degraded performance and/or an increased occurrence rate of hang/crash failures. This phenomenon has been called "software aging" [11]. Some common causes of software aging are memory leaks, unreleased file descriptors and numerical round-off errors. In order to counteract this problem, Huang et al. [23] proposed the technique of software rejuvenation, which involves occasionally stopping the software application, removing the accrued error conditions and then restarting the application in a clean environment. This process removes the accumulated errors and frees up or defragments operating system resources, thus preventing, in a proactive manner, unplanned and potentially expensive future system outages. Unlike downtime caused by sudden failure occurrences, the downtime related to software rejuvenation can be scheduled at the discretion of the user/administrator, e.g., for the middle of the night. Meanwhile, rejuvenation has been implemented in various types of systems, like billing data collection systems [23], telecommunication systems [4], transaction processing systems [8], cluster servers [9] and spacecraft systems [34]. For a recent introduction to software rejuvenation, see [5]. Many research papers on this topic can be found at [10].

In the face of an accumulation of errors and an increasing failure rate on the one hand and the direct and indirect costs of rejuvenating a software system on the other hand, an optimal timing of software rejuvenation should be sought.

Approaches used for analyzing and solving this optimization problem can be grouped into *model-based* ones and *measurement-based* ones.

The *model-based* approaches are aimed at building analytic models of system degradation and solving these models for determining the effectiveness of software rejuvenation as well as for deriving optimal rejuvenation schedules.

A simple degradation model was introduced by Huang et al. [23], who assumed that once a software system switches to the failure probable state the time until rejuvenation is carried out follows an exponential distribution. In order to deal with periodic rejuvenation and deterministic intervals between successive rejuvenations, Garg et al. [15] applied a Markov regenerative Petri net model. As a response to criticism of the coarse nature of these early models, Bobbio et al. [6] took into account that system performance can degrade in a fine-grained way; they modeled the degradation process as a sequence of additive random shocks. Garg et al. [16] used a queuing model for analyzing two rejuvenation policies (purely time based vs. instantaneous load and time based) in a transactions based software system. Dohi et al. [12, 13, 14] formulated software rejuvenation models as semi-Markov reward processes, which do not depend on specific failure-time distributions. All models mentioned so far only dealt with single-level rejuvenation, i.e. one kind of rejuvenation (usually full system restart). Vaidyanathan et al. [35] considered two kinds of preventive maintenance in operational software systems. Likewise, Xie et al. [38] generalized the semi-Markov model presented in [14] by introducing the possibility of service-level rejuvenation in addition to system-level rejuvenation.

The basic idea of *measurement-based* approaches is to directly monitor attributes that are subject to software aging. For example, in a system with memory leaks not all of the memory allocated to a task is necessarily released after its completion, leading to an increasing trend in memory usage. Based on periodically collected data, measurement-based approaches try to assess the current "health" of the software system and to obtain predictions about possible impending failures due to resource exhaustion.

Garg et al. [17] analyzed the exhaustion of resources like real memory and swap space in a network of UNIX workstations. All those metrics, observed on three computers, showed a significant trend over time. Using a non-parametric technique, Garg et al. determined the global trend and calculated the estimated time to exhaustion via linear extrapolation for each resource. Vaidyanathan and Trivedi [36] took into account the possibly differing rates of resource depletion over time by identifying eight states of system workload, and they determined an individual trend for each of them. Modeling the workload states as a semi-Markov chain, they were able to calculate the expected development of resource exhaustion. However, their predictions were again linear functions of time. Castelli et al. [9] examined software aging in a cluster of servers. For the prediction of resource exhaustion they fitted a (piecewise) linear trend to the measurements taken within a fitting window, or to the logarithm of these measurements.

None of these previous measurement-based approaches explicitly models seasonal patterns occurring in the measurement data. The predictions of future resource usage (or its logarithm) are linear functions of time.

A completely different approach to the analysis of aging in memory resources was chosen by Shereshevsky et al. [32]. These authors did not model and predict memory utilization, but they monitored the local rate of fractality of the system parameters via the Hölder exponent and concluded that the second abrupt increase in this measure indicates an imminent system crash.

In this paper, we analyze resource usage data collected on an Apache web server. A web server is a typical long running software system which should ideally operate without interruptions. However, due to unfixed bugs in the application or system software the performance is prone to degrade over time. System administrators usually reboot the whole system or just restart the web server program occasionally to deal with this problem. Most administrators set the interval between restarts according to experience or restart the web server only after it crashes. Since the downtime of business-critical web services like online sales directly translates into financial losses, a better understanding of web server aging leading to a more appropriate scheduling of software rejuvenation is crucial.

The main contribution of this paper is the detailed examination of the development of response time and memory usage of an Apache web server subjected to a synthetic load for 25 days. Our analyses reveal the influence of settings related to both the operating system as well as Apache itself on the aging phenomenon. Unlike previous research in which the predictions of resource usage were linear functions of time even in the presence of seasonality, we parsimoniously model the existing seasonal pattern and exploit it for forecasting the future behavior.

The rest of the paper is organized as follows: In Section 2, we discuss the experimental setup as well as the experiments carried out in our study and describe the data sets collected. The resource usage data is then statistically analyzed in Section 3. Section 4 contains concluding remarks and discusses possible directions for future research.

2 Experiments

2.1 Experimental setup

Apache is the most popular web server software currently being used [29]. Our experimental setup consists of a server running Apache version 1.3.14 on a Linux platform and a client connected via an Ethernet local area network. The components and the system structure are illustrated in Figure 1. For collecting resource usage information of the web server we make use of the fact that in the Linux operating system all the system information is stored in the `/proc` virtual file system.

For example, the file `/proc/stat` contains information about CPU usage, disk I/O, paging, swap space, interrupts, context switches and processes. We employ a Linux monitoring tool called `procmon`, developed in our research group by Rajiv Poonamalli, to monitor the web server. `procmon` periodically extracts information from the `/proc` file system and stores it in a designated file in a certain format. The tool settings, such as the interval of extracting information and the parameters to monitor are specified in a configuration file.

In our experiments, we use `httperf` [28] in order to generate requests with constant time intervals between two requests. Each request accesses one of five specified files of sizes 500 bytes, 5 kB, 50 kB, 500 kB and 5 MB from the server. The corresponding probabilities of accessing the files are 0.35, 0.5, 0.14, 0.009 and 0.001, respectively.

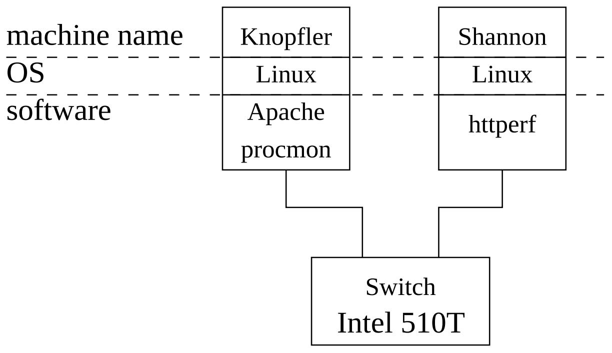

Figure 1: *Experimental setup*

`httperf` is not only a workload generator, but it can also be employed for monitoring performance information. The measurements provided include reply rate, response time and the number of timeouts. While the reply rate is a fundamental index of capacity, response time and timeout rate are important in measuring the performance of the web server.

2.2 Determining the capacity of the web server

With our first experiment we determine the capacity of our Apache web server. For this end, we observe its reply rate and timeout error rate while increasing the connection rate of requests generated by `httperf` until the web server is overloaded.

It should be clear that the web server capacity depends on its configuration. We study the influence of two parameters with which Apache provides some features similar to software rejuvenation. First, if the configuration variable `MaxRequestsPerChild` is set to a positive value, then the parent process of Apache kills a child process as soon as `MaxRequestsPerChild` requests have been handled by this child process. By doing this, Apache "limits the amount of memory that [one] process can consume by (accidental) memory leakage" and "helps reduce the number of processes when the server load reduces" [1]. If `MaxRequestsPerChild` is set to its default value 0, then a process never expires, i.e., this value is really meant to represent infinity.

A second parameter, `MaxClients`, sets the limit on the number of child processes that can be running concurrently, i.e., the number of clients that can simultaneously connect to the server. It acts as a brake to keep a runaway server from taking the system with it as it spirals down. The default value of this parameter is 250.

Figure 2 depicts the results obtained for the default settings of `MaxRequestsPerChild` and `MaxClients`, 0 and 250, respectively. Above the peak at 390 replies per second, the reply rate deviates from the connection rate. We therefore conclude that under the default setting, the capacity of the web server is about 390 requests per seconds.

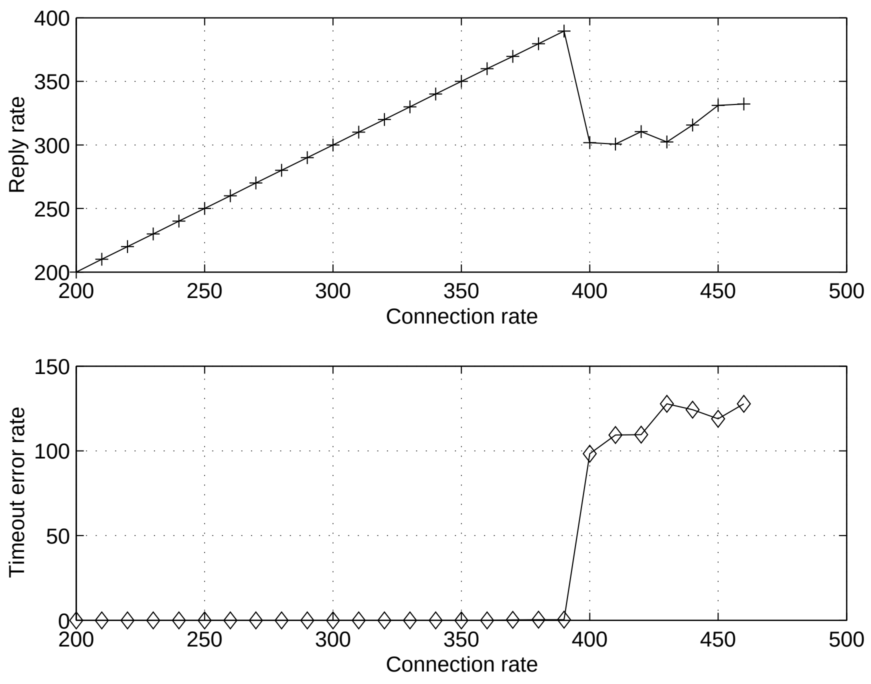

Figure 2: *Capacity of the web server*

In order to assess the impact of the two configuration parameters, we repeat the experiment with different settings. The results shown in Table 1 indicate that decreasing either `MaxClients` or `MaxRequestsPerChild` will eventually have adverse effects on the web server capacity.

The explanation of this behavior with respect to the former parameter is as follows: If `MaxRequestsPerChild` is set to a smaller value, then the number of requests processed by each child process quickly attains this maximum value even for a lower workload. As a consequence, the Apache parent process has to kill and respawn child processes with a high frequency, which increases the overhead on the system. Therefore, the capacity of the web server decreases.

Table 1: *Web server capacity under different configurations*

`MaxClients` $\rightarrow$ `MaxRequestsPerChild` $\downarrow$	250	100	50	30
0 (∞)	390	390	390	380
3000	390	390	390	380
1000	390	390	390	380
750	390	390	390	380
500	380	380	380	380
250	370	370	370	360

This finding agrees with the results obtained by Arlitt and Williamson [3]. Their experiments revealed that restricting the number of requests per child process can considerably decrease the reply rate achieved by the Apache web server. The authors therefore concluded that the configuration parameter `MaxRequestsPerChild` should only be changed to a non-zero value if absolutely necessary. (Recall that a zero value corresponds to no limit on the number of requests processed.)

If the second parameter, `MaxClients`, is set to a small value, then only a small number of clients can connect simultaneously, and the possibility of dropping a request becomes larger. Moreover, the processing rate of each child process needs to be increased to handle the workload, which results in a higher frequency of killing and respawning child processes. Both effects lead to a lower server capacity.

During the following analyses, we set both `MaxRequestsPerChild` and `MaxClients` to their default values (0 and 250, respectively).

2.3 Collection of resource usage data

For collecting resource usage data over a long time period, we employ a shell program to run `httperf` periodically. As the connection rate we choose a value of 400 requests per second, which puts the web server in an overload state and should speed up software aging. Among the system parameters of the web server monitored during a period of more than 3.5 weeks are the response time of the web server (ResponseTime), i.e., the interval from the time `httperf` sends out the first byte of request until it receives the first byte of reply, the free physical memory (FreePhysMem) and the used swap space (UsedSwapSpace). The three time series are shown in Figures 3 to 5. Since the spacing between two consecutive data points is five minutes, each time series consists of 7208 observations.

From Figure 5, it is obvious that swap space usage follows a seasonal pattern. In our statistical data analysis, we will need to account for this phenomenon. Searching for the reason of the periodicity, further investigation reveals that the abrupt decreases in used swap space are related to the log rotation, which by default is one of the routines started by the `cron` daemon on Sunday mornings at about 4:00 a.m. In the course of the archiving of the Apache access log files, the `HUP` signal is sent to the Apache parent process, triggering it to kill all of its child processes, re-open any log files and spawning a new set of children to handle requests [2]. This means that the system autonomously invokes a software rejuvenation mechanism by regularly killing the Apache child processes even if the `MaxRequestsPerChild` parameter is set to zero.

Figure 5 also shows that significant increases in UsedSwapSpace often - but not always - occur at fixed intervals. This is especially clear for the second week of data collection, about 130 to 300 hours into the experiment.

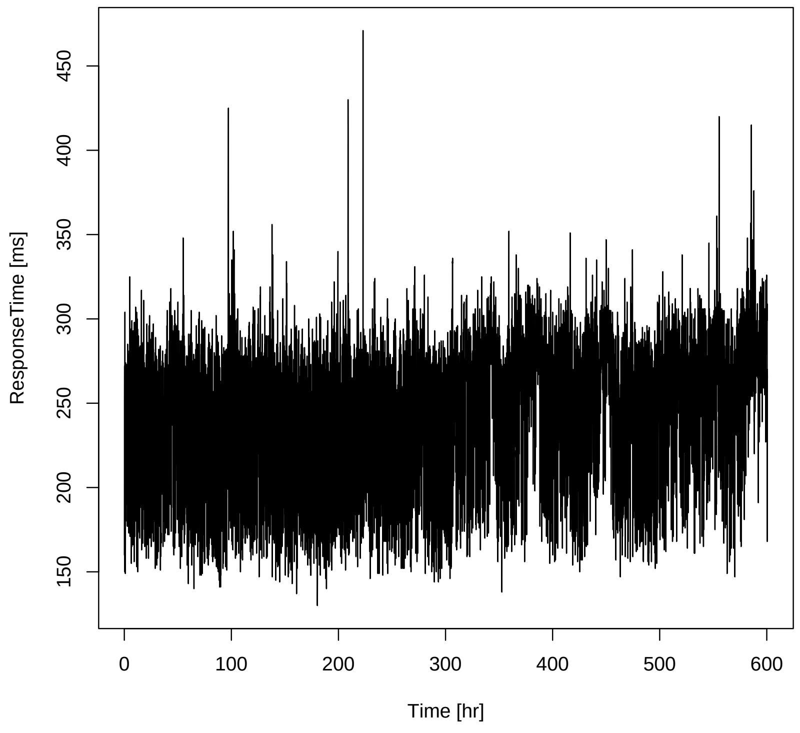

Figure 3: *Response time*

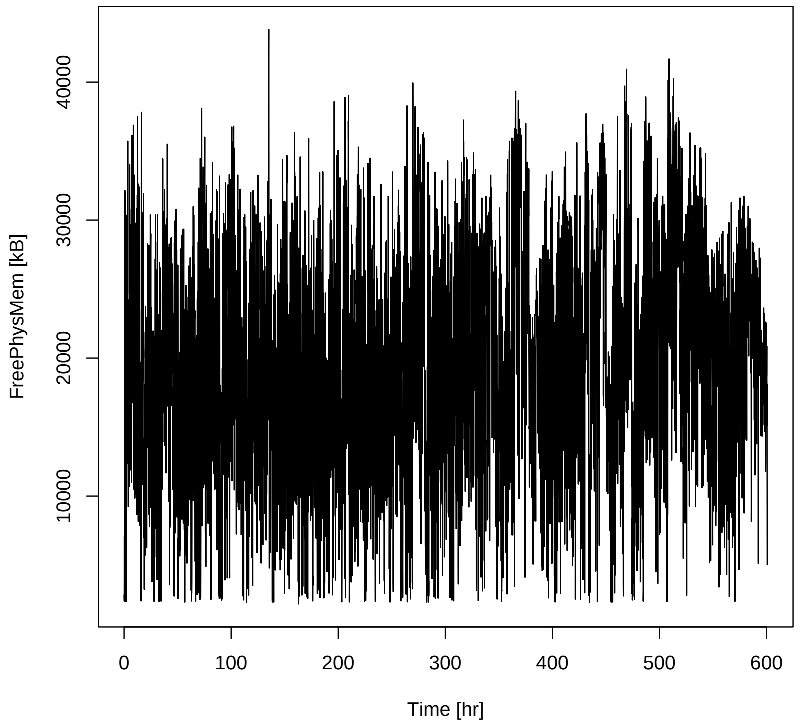

Figure 4: *Free physical memory*

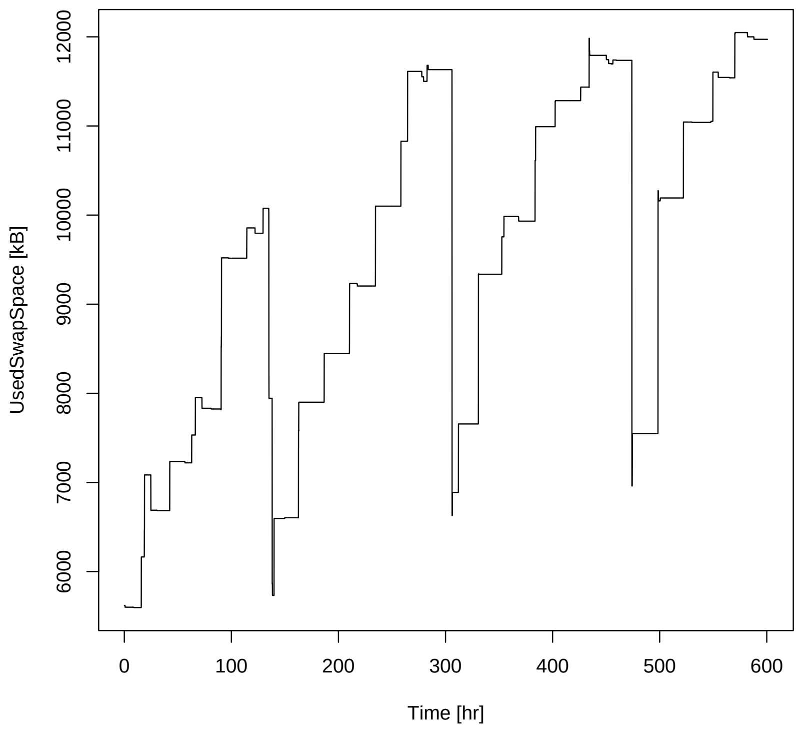

Figure 5: *Used swap space*

Again, this pattern is caused by the `cron` daemon: Among the commands it invokes on a daily basis by default is the update mechanism of the database used by `slocate` for quickly searching for files in the system. Although the updating is started with a low priority, under usual circumstances it would be finished within a short time. However, during our experiments the web server is constantly in an overload state, since the request rate exceeds its capacity. As a consequence, the memory requirements of the database update and (eventually) the process itself are directed to the swap space. This explains, why the increases in UsedSwapSpace are not noticeably accompanied by increases in FreePhysMem. Furthermore, the incidental rejuvenation carried out once a week frees the resources for the execution and termination of those processes that had been started but never finished throughout the week.

3 Statistical analysis of resource-usage data

3.1 Trend test and estimation

In our first analyses, we wish to determine if the data collected indicate that the response time of the web server degrades or that the memory is depleted over time. Toward this end, we apply a set of non-parametric statistical methods. Analyses to be used depend on whether seasonal patterns are present in the respective time series examined.

3.1.1 Data without a seasonal pattern

In this section we deal with the two time series showing no signs of seasonality, ResponseTime and FreePhysMem (see Figures 3 and 4).

For testing the null hypothesis that a sample $y_1, y_2, ..., y_n$ does not exhibit a trend, Mann [27] used a linear function of the test statistic S originally developed by Kendall [24] for testing whether two sets of rankings are independent. The direct application of S for our purposes is known as the Mann-Kendall test for trend. In this context, the value of the test statistic is computed as

$$s = \sum_{k=1}^{n-1} \sum_{l=k+1}^{n} \text{sgn}(y_l - y_k),$$

where sgn$(\cdot)$ denotes the signum function. Of all $\binom{n}{2} = n(n-1)/2$ pairs of observations y_k, y_l $(k < l)$, s counts all pairs for which the earlier observation y_k is smaller than y_l and subtracts the number of pairs for which the latter observation is smaller. While an s value close to zero suggests that there is no trend in the data, a high absolute value of the test statistic hints at the existence of a trend. For the calculation of s, tied pairs, i.e., those pairs for which $y_k = y_l$, are not taken into account. However, such tied pairs do influence the variance of the test statistic. If the time series is composed of o distinct sets of values, with t_j observations in the j^{th} set, then the variance of S is given by [25, p. 43]

$$\text{Var}(S) = \frac{1}{18}\left[n(n-1)(2n+5) - \sum_{j=1}^{o} t_j(t_j-1)(2t_j+5)\right].$$

Under the null hypothesis the distribution of S is always symmetric, and the expected value of S is equal to zero. Moreover, for n approaching infinity, the distribution of S converges to the normal distribution. Allowing for a continuity correction [25, pp. 41–42], the value of the test statistic

$$Z = \frac{S - \text{sgn}(S)}{\sqrt{\text{Var}(S)}} \tag{1}$$

can be compared to the quantiles of the standard normal distribution in order to check whether the null hypothesis of no trend in the data can be rejected.

The values of Z calculated for the time series ResponseTime and FreePhysMem are listed in Table 2.

Table 2: *Trend test and estimation for ResponseTime and FreePhysMem*

	ResponseTime	FreePhysMem
z	21.569	14.873
Estimated slope	0.061 ms/hr	8.377 kB/hr
95% confidence interval	(0.055 ms/hr, 0.067 ms/hr)	(7.287 kB/hr, 9.472 kB/hr)

Both are considerably larger than $\lambda_{0.975} = 1.960$, the 97.5% quantile of the standard normal distribution. Consequently, the null hypothesis that the time series contains no trend can be rejected in each case at an error level of 5%. The positive signs of the z values show that the two trends are increasing. In order to determine estimates for the slopes, we apply a non-parametric procedure developed by Sen [30]. This method is not affected by outliers, and it is robust to missing data. Like the calculation of the value of the test statistic S the approach focuses on all pairs of data points y_k, y_l with $k < l$. For each of these pairs, the slope $q_{kl} = (y_l - y_k)/(l - k)$ is calculated. Sen's slope estimate is defined as the median of the $n' = n(n-1)/2$ slopes obtained.

A two-sided $100 \cdot (1-\alpha)\%$ confidence interval for the estimated slope can be derived by the procedure described in [18, p. 218]: After sorting the n' slopes in increasing order, the lower limit of the confidence interval is given by the $((n' - c_\alpha)/2)^{th}$ largest of these slopes, while the upper limit is given by the $((n' + c_\alpha)/2 + 1)^{th}$ largest slope, where $c_\alpha = \lambda_{1-\alpha/2}\sqrt{\text{Var}(S)}$. Again, λ refers to the quantiles of the standard normal distribution.

Although the procedure is simple enough, its memory requirements are quite demanding for our time series consisting of $n = 7208$ observations, because almost 26 million slopes have to be computed and sorted. The slope estimates and their respective 95% confidence intervals are shown in Table 2. As anticipated, after the calculation of the values of the Z statistic, the estimated slope is positive for both ResponseTime and FreePhysMem.

For the former time series, the result is consistent with the symptoms of software aging: On average, the response time as perceived by the user increases by about 0.06 ms every hour. The longer the web server has been operating, the more time it tends to need for reacting to a request. With regard to the FreePhysMem time series, however, we would rather have expected a decreasing trend. A possible explanation for the observed behavior is the fact that free physical memory in the system cannot be lower than a certain threshold. Since the connection rate of 400 connections per second exceeds the capacity of the web server, the physical memory is close to its lower limit from the very beginning of the experiment. Therefore, there is little if any scope left for a further decrease in free physical memory. Rather, the system tries to free some of the used physical memory. Indeed, the fluctuations in the time series (cf. Figure 4) are much more drastic than if the web server is operated within its capacity, which hints at the constant activities undertaken by the system to reclaim physical memory. The overall increase in free physical memory may therefore be due to the successful paging out of inactive processes initially blocking the resource.

In the plot of UsedSwapSpace an upward trend is clearly visible, although it is superimposed with the seasonal pattern caused by the weekly "rejuvenation". We will further investigate this global trend in the used swap space in the next section.

3.1.2 Data with a seasonal pattern

For seasonal data, Hirsch et al. [22] proposed a modified Mann-Kendall test. The main idea is to separately treat the data of each of the m seasons. From the subsample pertaining to the i^{th} season, denoted by $y_{i1}, y_{i2}, ..., y_{in_i}$, where n_i is the total number of observations for this season, the value

$$s_i = \sum_{k=1}^{n_i-1} \sum_{l=k+1}^{n_i} \text{sgn}(y_{il} - y_{ik})$$

is calculated. Under the null hypothesis of no trend, the statistic S_i is asymptotically normal with variance

$$\text{Var}(S_i) = \frac{1}{18}\left[n_i(n_i-1)(2n_i+5) - \sum_{j=1}^{o_i} t_{ij}(t_{ij}-1)(2t_{ij}+5)\right],$$

assuming that the data of the i^{th} season consists of o_i different values with the number of data points taking the j^{th} value given by t_{ij}.

If the null hypothesis is true, then the S_i statistics are mutually independent, and their sum $S = \sum_{i=1}^{m} S_i$ follows a normal distribution with expectation zero and variance $\text{Var}(S) = \sum_{i=1}^{m} \text{Var}(S_i)$. Therefore, the hypothesis can be tested by computing the value of the statistic Z, defined according to equation (1), and comparing it to the respective percentiles of the standard normal distribution.

Since the automatic "rejuvenation" carried out by the system takes place once a week, there are $m = 2016$ seasons to consider. The calculation of the values of the 2016 statistics S_i and their variances leads to a z of 68.443, which clearly indicates the existence of a positive trend in the data.

According to van Belle and Hughes [37], a different trend test based on a rank order test developed by Sen [31] is more powerful than the seasonal Mann-Kendall test. However, this test requires an identical number of observations for each season. As a consequence, for our data the computation can only be based on $3 \cdot 2016 = 6048$ data points, while 1160 observations have to be discarded. Nevertheless, we did carry out this test and received a result similar to the one of the seasonal Mann-Kendall test: The null hypothesis of no trend can be rejected at high significance levels.

Like the Mann-Kendall test, Sen's slope estimator can be adapted in the presence of a seasonal pattern, cf. [18, pp. 227–228]. Again, the m seasons are at first analyzed separately, calculating the slope $q_{ikl} = (y_{il} - y_{ik})/(l-k)$ for each of the $n'_i = n_i(n_i-1)/2$ pairs of data points y_{ik}, y_{il} $(k < l)$ belonging to season i. The overall slope estimate is then the median of all $n' = \sum_{i=1}^{m} n'_i$ slopes. With n' given above and $\text{Var}(S)$ computed as the sum of the m variances $\text{Var}(S_i)$, the $100 \cdot (1-\alpha)\%$ confidence interval for the slope estimate is derived in the same way as described for the (basic) Sen slope estimator.

For the slope of UsedSwapSpace, the estimate as well as the 95% confidence interval are listed in Table 3.

Table 3: *Trend test and estimation for UsedSwapSpace*

	UsedSwapSpace
z	68.443
Estimated slope	7.714 kB/hr
95% confidence interval	(7.714 kB/hr, 7.786 kB/hr)

Due to ties in the data the lower bound is identical to the estimate itself. The results confirm that the amount of swap space used tends to increase over time. The estimated slope is of the same magnitude as the one for free physical memory shown in Table 2. In fact, its 95% confidence interval is completely contained in the 95% confidence interval derived for the latter quantity. This raises the question whether the system ages at all with respect to memory usage, because the amplified usage of swap space could be explained by the swapping of unused processes from the physical memory. However, the global trend incorporates the effects caused by the weekly log rotation, which we discussed above in Section 2.3. Without the incidental rejuvenation prompted by this mechanism, accumulated memory leaks related to the Apache child processes would not be released, and the increase in swap space usage would be significantly higher.

3.2 Time series analysis

After the initial trend analysis, we now apply time series models to our observations. Our goals are to gain further insight into the structure of the time series and to explore whether the seasonal structure of swap space usage can be exploited for predicting future values.

A simple class of models for time series in which consecutive observations are correlated is the autoregressive (AR) model. In an autoregressive model of order p, each observation y_t is explained by the p previous values of the time series:

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + u_t,$$

where the ϕ_is are fixed parameters.

It is assumed that the u_ts, the typically unobservable deviations from the perfect autoregressive relationship, are samples from a white noise process with zero mean and constant variance σ_U^2.

If the trend and the seasonal pattern of the time series analyzed are stable over time, they can be modeled deterministically. One way to account for a seasonal structure of period m is via the inclusion of the coefficients $\gamma_1, ..., \gamma_m$, each representing the effect of one season. The resulting model has the form

$$y_t = \alpha_1 b(t) + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{m} \gamma_j d_{jt} + u_t, \tag{2}$$

with the dummy variable d_{jt} being equal to one if the t^{th} observation belongs to the j^{th} season and zero otherwise. While the parameter α_1 needs to be estimated, the function $b(t)$ stands for the known part of the deterministic trend component; possible choices include $b(t) = t$ and $b(t) = \sqrt{t}$. An intercept term α_0 must not be added to the model (2) since it would lead to perfect multicollinearity [20, p. 118]. For the column vector of observations with transpose $\boldsymbol{y}^T = (y_1, \ldots, y_n)$ the model can be written in matrix form as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\alpha} + \boldsymbol{D}\boldsymbol{\gamma} + \boldsymbol{u}, \tag{3}$$

where

$$\boldsymbol{X} = \begin{pmatrix} b(1) & y_0 & \cdots & y_{1-p} \\ b(2) & y_1 & \cdots & y_{2-p} \\ \vdots & \vdots & \ddots & \vdots \\ b(t) & y_{t-1} & \cdots & y_{t-p} \end{pmatrix}, \qquad \boldsymbol{D} = \begin{pmatrix} d_{11} & \cdots & d_{m1} \\ \vdots & \ddots & \vdots \\ d_{1t} & \cdots & d_{mt} \end{pmatrix},$$

$\boldsymbol{\alpha} = (\alpha_1, \phi_1, \ldots, \phi_p)^T$, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_m)^T$ and $\boldsymbol{u} = (u_1, \ldots, u_n)^T$. If the observations span k complete seasonal cycles, then the matrix $\boldsymbol{D}$ can be constructed by stacking k identity matrices; i.e., with $\boldsymbol{I}_m$ denoting an $m \times m$ matrix with a diagonal of ones and off-diagonal elements of zero, $\boldsymbol{D} = [\boldsymbol{I}_m \boldsymbol{I}_m \cdots \boldsymbol{I}_m]^T$.

For the UsedSwapSpace data the period of the seasonal pattern, m, is equal to 2016. The estimation of (3) via the least squares method requires the inversion of the huge matrix $[\boldsymbol{X}\boldsymbol{D}]^T[\boldsymbol{X}\boldsymbol{D}]$, consisting of $(2017 + p)$ columns and the same number of rows. This task can be avoided by carrying out a partitioned regression [20, pp. 26–27]: In a first step, $\boldsymbol{y}$ and each of the columns in $\boldsymbol{X}$ are regressed on the dummy variables in the matrix $\boldsymbol{D}$, and the residuals are determined; in fact, this amounts to subtracting the seasonal means from all values. In a second step, these residuals are used in a subsequent regression in order to calculate $\hat{\boldsymbol{\alpha}}$. Based on this estimated parameter vector as well as the seasonal means of the time series, $\hat{\boldsymbol{\gamma}}$ can then be obtained.

While parameter estimation is possible, modeling the seasonal pattern with 2016 parameters may not be the most parsimonious approach. As an alternative, trigonometric terms at the seasonal frequencies $\mu_j = \frac{2\pi j}{m}$ $(j = 1, \ldots, \lfloor m/2 \rfloor)$ can be employed, modeling the seasonal influence at time t by [21, p. 41]

$$\sum_{j=1}^{\lfloor m/2 \rfloor} \left(\beta_j \cos\left(\mu_j t\right) + \delta_j \sin\left(\mu_j t\right)\right).$$

This formulation does not implicitly include the level of the time series; therefore, an intercept α_0 should be added to the model. If the summation over j runs from 1 to $\lfloor m/2 \rfloor$ (the largest integer smaller than or equal to $m/2$), like in the previous expression, then the number of parameters used for modeling the seasonal structure plus the level is equal to m, just like in the dummy variable approach. However, it is often possible to make do with the lower-order frequencies $j = 1, 2, ..., f < \lfloor m/2 \rfloor$. The resulting model in vector form is

$$\boldsymbol{y} = \boldsymbol{X}^*\boldsymbol{\alpha}^* + \boldsymbol{D}^*\boldsymbol{\gamma}^* + \boldsymbol{u}, \tag{4}$$

with the matrices

$$\boldsymbol{X}^* = \begin{pmatrix} 1 & b(1) & y_0 & \cdots & y_{1-p} \\ 1 & b(2) & y_1 & \cdots & y_{2-p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & b(t) & y_{t-1} & \cdots & y_{t-p} \end{pmatrix} \quad \text{and}$$

$$\boldsymbol{D}^* = \begin{pmatrix} \cos(\mu_1) & \sin(\mu_1) & \cdots & \cos(\mu_f) & \sin(\mu_f) \\ \cos(2\mu_1) & \sin(2\mu_1) & \cdots & \cos(2\mu_f) & \sin(2\mu_f) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \cos(\mu_1 t) & \sin(\mu_1 t) & \cdots & \cos(\mu_f t) & \sin(\mu_f t) \end{pmatrix},$$

as well as the vectors $\boldsymbol{\alpha}^* = (\alpha_0, \alpha_1, \phi_1, \ldots, \phi_p)^T$ and $\boldsymbol{\gamma}^* = (\beta_1 \delta_1, \ldots, \beta_f, \delta_f)^T$.

Typically, neither the number of frequencies f to be used for the seasonal pattern nor the autoregressive order p are known. Choosing the model that fits the data best - i.e., that achieves the lowest estimated error variance $\hat{\sigma}_U^2 = n^{-1} \sum_{i=1}^n \hat{u}_i^2$ in least squares estimation or the largest likelihood value in maximum likelihood estimation - may result in overfitting. Therefore, model order selection criteria effectively penalize for the number of freely estimated parameters in the model, r. The Bayesian information criterion proposed by Schwarz, for example, takes the form [26]

$$\text{BIC} = \ln\left(\hat{\sigma}_U^2\right) + \frac{\ln(n)}{n} \cdot r. \tag{5}$$

for a wide variety of models, assuming that the disturbances of the model estimated via least squares estimation follow a Gaussian white noise process. The model attaining the smallest BIC value is considered most appropriate.

For the model order selection in a model with deterministic terms, Lütkepohl [26] proposed to estimate these in a first step, subtract the estimated deterministic function from the data and apply the order selection procedure to the adjusted data. With regard to our model (4) this basically means that for a *fixed* value of f the model is fitted to the data with varying autoregressive orders p. The value of p to be chosen is the one that minimizes the Bayesian information criterion, substituting p for r in equation (5).

However, we do not know the number of lower-order frequencies f to be used. We therefore fit the model to the data sets varying *both* f and p between 0 and 100, replacing r by $(2+2\cdot f+p)$ in each calculation of the BIC. (The two additional parameters considered in the model order r are related to the mean and the trend component.) Obviously, for any given f this approach leads to the same choice of the autoregressive order p, but it allows to jointly optimize f and p.

Table 4 lists the selected orders for our three time series. In order to permit model validation, all calculations are merely based on the first 4000 observations of each time series, i.e., roughly on the data collected during the first two weeks of experiments. The trend component employed for UsedSwapSpace is the square root trend; for the other two time series, we include a linear trend.

Table 4: *Selected model orders*

Time Series	AR order p	Frequency order f
FreePhysMem	8	0
ResponseTime	92	0
UsedSwapSpace	1	10

While observations made 7 hours and 40 minutes ago (92 obs $\times$ 5 min/obs) significantly influence the behavior of ResponseTime, the best model for UsedSwapSpace according to the BIC merely makes use of the last observation for predicting the future of the time series. The selected frequency orders confirm the impressions conveyed by the plots: For FreePhysMem and ResponseTime no parameters modeling a (weekly) seasonality are included. The frequency order determined for UsedSwapSpace is $f = 10$. Obviously, 20 parameters are enough for capturing the main features of the seasonal pattern spanning 2016 observations. The value of explicitly including a seasonal component for modeling a time series in which seasonality is present can now be illustrated with the help of UsedSwapSpace. Again using the first 4000 observations, the parameters of the model with an autoregressive order of 1 and a frequency order of 10 are estimated for UsedSwapSpace via ordinary least squares estimation. The observations employed for estimation as well as the respective fitted values are shown in Figure 6.

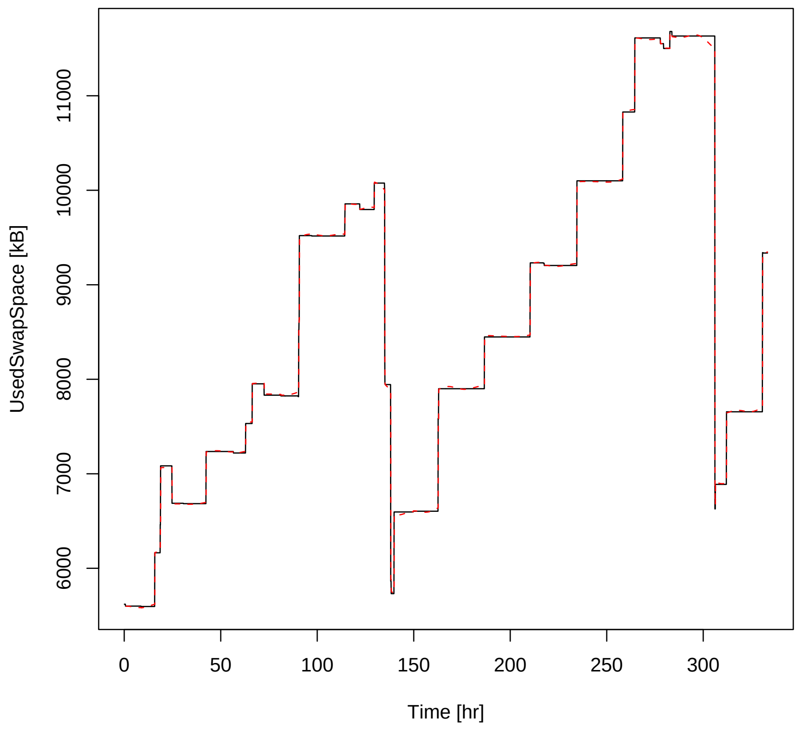

Figure 6: *UsedSwapSpace - first 4000 observations (–) and fitted values (- -)*

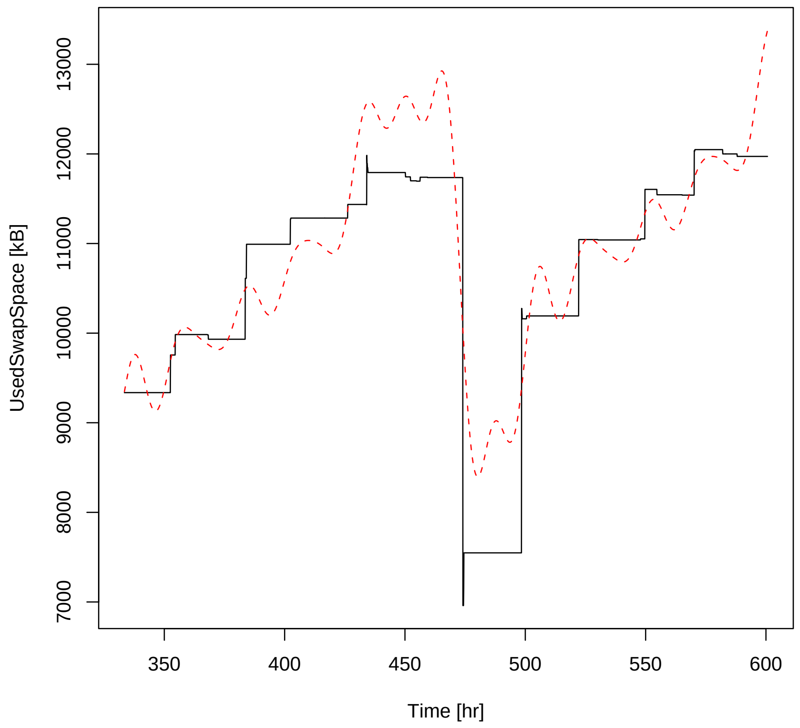

Figure 7: *UsedSwapSpace - last 3208 observations (—) and predicted values (- -)*

The model fit attained with a total of 23 parameters is very good; in fact, the observations and the fitted model can hardly be distinguished.

In order to validate the model, we use the parameter estimates from the first 4000 observations as well as the 4000^{th} observation itself and recursively predict the future behavior of the time series based on this information only. The predictions and the data actually measured are plotted in Figure 7. The comparison seems to indicate that the model is not only able to fit the data, but that it also provides good forecasts. Merely relying on the data collected during the first two weeks of experiments, we could have predicted the main features of the time series in the following two weeks.

4 Conclusions

In this paper, we have investigated the development of resource utilization in an Apache web server. Data collected during experiments in which the web server was put in an overload condition indicated the presence of software aging. However, the periodical killing of all Apache child processes in the course of the weekly log rotation (partly) offsets the aging phenomenon and can therefore be considered a kind of incidental software rejuvenation. This insight plus our findings obtained while trying to determine the capacity of the web server highlight the necessity to study the influence of settings of the web server itself as well as of the operating system.

For analyzing the collected data we employed both non-parametric statistical techniques and parametric time series models. Since previous research had not accounted for seasonality in the prediction of resource depletion, we focused on how to model seasonal patterns and determine the model order. For the swap space used, the one data set exhibiting weekly seasonality, we showed that a parsimonious model of the seasonal structure is able to adequately predict the future behavior for a period of more than 1.5 weeks.

These results are promising, but they indicate the need for further research. On the one hand, the influence of the configuration of the operating system and the Apache web server on the aging behavior should be studied in detail. For this end, we are planning to collect resource usage data under different combinations of parameter settings derived from a factorial design. On the other hand, we will employ multivariate time series models in order to investigate the interactions between various system resources. Moreover, we will build an optimization model using the predictions of resource exhaustion as well as further information for deriving the best rejuvenation schedule.

Acknowledgments

The authors would like to thank Bennett Crowell, Department of Electrical and Computer Engineering at Duke University, for his helpful comments on a previous version of this paper. Funding for this research was provided in part by fellowships from the German Academic Exchange Service (Postdoc Program) as well as from Aprisma and NCNI, and by Telcordia and NSF under a CACC core project.

References

[1] Apache Software Foundation. Apache MPM Common Directives. http://httpd.apache.org/docs-2.1/mod/mpm_common.html (Link verified on July 3, 2005).

[2] Apache Software Foundation. Stopping and restarting Apache. http://httpd.apache.org/docs-2.1/stopping.html (Link verified on July 3, 2005).

[3] M. Arlitt and C. Williamson. Understanding web server configuration issues. *Software – Practice and Experience*, 34(2):163–186, 2004.

[4] A. Avritzer and E. J. Weyuker. Monitoring smoothly degrading systems for increased dependability. *Empirical Software Engineering*, 2(1):59–77, 1997.

[5] L. Bernstein and C. M. R. Kintala. Software rejuvenation. *CrossTalk*, 17(8):23–26, 2004.

[6] A. Bobbio, A. Sereno, and C. Anglano. Fine grained software degradation models for optimal rejuvenation policies. *Performance Evaluation*, 46(1):45–62, 2001.

[7] G. Candea, J. Cutler, and A. Fox. Improving availability with recursive microreboots: a soft-state system case study. *Performance Evaluation*, 56(1-4):213–248, 2004.

[8] K. J. Cassidy, K. C. Gross, and A. Malekpour. Advanced pattern recognition for detection of complex software aging in online transaction processing servers. In *Proc. International Conference on Dependable Systems and Networks*, pages 478–482, 2002.

[9] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert. Proactive management of software aging. *IBM Journal of Research and Development*, 45(2):311–332, 2001.

[10] Department of Electrical and Computer Engineering, Duke University. Software rejuvenation. http://www.software-rejuvenation.com/ (Link verified on July 3, 2005).

[11] T. Dohi, K. Goseva-Popstojanova, K. Vaidyanathan, K. S. Trivedi, and S. Osaki. Software rejuvanation: Modeling and applications. In H. Pham, editor, *Handbook of Reliability Engineering*, pages 245–263. Springer, London, 2003.

[12] T. Dohi, K. Goševa-Popstojanova, and K. S. Trivedi. Analysis of software cost models with rejuvenation. In *Proc. International Symposium on High Assurance Systems Engineering*, pages 25–34, 2000.

[13] T. Dohi, K. Goševa-Popstojanova, and K. S. Trivedi. Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule. In *Proc. International Pacific Rim Symposium on Dependable Computing*, pages 77–84, 2000.

[14] T. Dohi, K. Goševa-Popstojanova, and K. S. Trivedi. Estimating software rejuvenation schedules in high assurance systems. *Computer Journal*, 44(6):473–482, 2001.

[15] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of software rejuvenation using Markov regenerative stochastic Petri net. In *Proc. Sixth International Symposium on Software Reliability Engineering*, pages 24–27, 1995.

[16] S. Garg, A. Puliafito, M. Telek, and K. S. Trivedi. Analysis of preventive maintenance in transactions based processing systems. *IEEE Trans. Computers*, 47(1):96–107, 2001.

[17] S. Garg, A. van Moorsel, K. Vaidyanathan, and K. S. Trivedi. A methodology for detection and estimation of software aging. In *Proc. Ninth International Symposium on Software Reliability Engineering*, pages 283–292, 1998.

[18] R. O. Gilbert. *Statistical Methods for Environmental Pollution Monitoring*. Van Nostrand Reinhold, New York, 1987.

[19] J. Gray and D. P. Siewiorek. High-availability computer systems. *IEEE Computer*, 24(9):39–48, 1991.

[20] W. H. Greene. *Econometric Analysis*. Prentice-Hall, Upper Saddle River, 5th edition, 2003.

[21] A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge, 1989.

[22] R. M. Hirsch, J. R. Slack, and R. A. Smith. Techniques of trend analysis for monthly water quality data. *Water Resources Research*, 18(1):107–121, 1982.

[23] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton. Software rejuvenation: Analysis, module and applications. In *Proc. Twenty-fifth International Symposium on Fault-Tolerant Computing*, pages 381–390, 1995.

[24] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[25] M. G. Kendall. *Rank Correlation Methods*. Charles Griffin & Company Ltd., London, 1948.

[26] H. Lütkepohl. Univariate time series analysis. In H. Lütkepohl and M. Krätzig, editors, *Applied Time Series Econometrics*, pages 8–85. Cambridge University Press, Cambridge, 2004.

[27] H. B. Mann. Nonparametric tests against trend. *Econometrica*, 13(3):245–259, 1945.

[28] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *Proc. First Workshop on Internet Server Performance*, pages 59–67, 1998.

[29] Netcraft Ltd. Netcraft: July 2005 Web Server Survey. http://news.netcraft.com/archives/2005/07/01/july_2005_ web_server_survey.html (Link verified on July 3, 2005).

[30] P. K. Sen. Estimates of the regression coefficient based on Kendall's tau. *Journal of the American Statistical Association*, 63(4):1379–1389, 1968.

[31] P. K. Sen. On a class of aligned rank order tests in two-way layouts. *Annals of Mathematical Statistics*, 39(4):1115–1124, 1968.

[32] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu. Sofware aging and multifractality of memory resources. In *Proc. International Conference on Dependable Systems and Networks*, pages 721–730, 2003.

[33] M. Sullivan and R. Chillarege. Software defects and their impact on system availability - A study of field failures in operating systems. In *Proc. Twenty-first International Symposium on Fault-Tolerant Computing*, pages 2–9, 1991.

[34] A. T. Tai, L. Alkalaj, and S. N. Chau. On-board preventive maintenance: a design-oriented analytic study for long-life applications. *Performance Evaluation*, 35(3-4):215–232, 1999.

[35] K. Vaidyanathan, D. Selvamuthu, and K. S. Trivedi. Analysis of inspection-based preventive maintenance in operational software systems. In *Proc. Twenty-first International Symposium on Reliable Distributed Systems*, pages 286–295, 2002.

[36] K. Vaidyanathan and K. S. Trivedi. A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, 2(2):124–137, 2005.

[37] G. van Belle and J. P. Hughes. Nonparamteric tests for trend in water quality. *Water Resources Research*, 20(1):127–136, 1984.

[38] W. Xie, Y. Hong, and K. S. Trivedi. Analysis of a two-level software rejuvenation policy. *Reliability Engineering & System Safety*, 87(1):13–22, 2005.