Analysis of Software Aging in a Web Server

Michael Grottke, *Member, IEEE*, Lei Li, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi, *Fellow, IEEE*

***Abstract*—Several recent studies have reported & examined the phenomenon that long-running software systems show an increasing failure rate and/or a progressive degradation of their performance. Causes of this phenomenon, which has been referred to as "software aging", are the accumulation of internal error conditions, and the depletion of operating system resources. A proactive technique called "software rejuvenation" has been proposed as a way to counteract software aging. It involves occasionally terminating the software application, cleaning its internal state and/or its environment, and then restarting it. Due to the costs incurred by software rejuvenation, an important question is when to schedule this action. While periodic rejuvenation at constant time intervals is straightforward to implement, it may not yield the best results. The rate at which software ages is usually not constant, but it depends on the time-varying system workload. Software rejuvenation should therefore be planned & 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 & activity parameters. Non-parametric statistical methods are then applied toward detecting & 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*—Apache web server, Linux, non-parametric trend analysis, performance monitoring, prediction of resource utilization, software aging, software rejuvenation, time series analysis.**

ACRONYMS[1]

AR	autoregressive
BIC	Bayesian information criterion
OS	operating system

Manuscript received July 17, 2005; revised January 18, 2006. 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. Associate Editor: J. C. Lu.

M. Grottke is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA, on leave of absence from the Chair of Statistics and Econometrics, University of Erlangen-Nuremberg, Germany (e-mail: grottke@ee.duke.edu).

L. Li is with the Wireless & Mobile Systems Group, Freescale Semiconductor, Inc., Austin, TX 78735 USA (e-mail: leili@freescale.com).

K. Vaidyanathan is with the RAS Computer Analysis Laboratory, Sun Microsystems, San Diego, CA 92121 USA (e-mail: kalyan.vaidyanathan@sun.com).

K. S. Trivedi is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: kst@ee.duke.edu).

Digital Object Identifier 10.1109/TR.2006.879609

[1]The singular and plural of an acronym are always spelled the same.

NOTATION

$\mathrm{gilb}(\cdot)$	greatest integer lower bound
$\mathrm{sgn}(\cdot)$	signum function
$\mathrm{Var}(\cdot)$	Variance (square of standard deviation)
s-	implies: statistical(ly)
λ_p	$100p\%$ quantile of the standard s-normal distribution
$\binom{n}{r}$	$n!/(n-r)!r!$: binomial coefficient

I. INTRODUCTION

It has now been well-established that failures of computer systems are more often due to software faults than due to hardware faults [19], [35]. 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 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" [14]. Some common causes of software aging are memory leaks, unreleased file-locks, and round-off errors. Huang *et al.* [24] proposed the technique of software rejuvenation in order to counteract this phenomenon. It involves occasionally stopping the software application, cleaning its internal state and/or its environment, and then restarting it. By removing the accrued error conditions and freeing up or defragmenting operating system (OS) resources, this technique proactively prevents unexpected 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., in the middle of the night. Meanwhile, rejuvenation has been implemented in various types of systems, like billing data collection systems [24], telecommunication systems [5], transaction processing systems [8], cluster servers [9], and spacecraft systems [36]. For recent introductions to software rejuvenation, see [6], and [21]. 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.

0018-9529/$20.00 © 2006 IEEE

Approaches used for analysing 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.* [24], who assumed that once a software system switches to the failure probable state, the time until rejuvenation is carried out follows an exponential distribution. To deal with periodic rejuvenation, and deterministic intervals between successive rejuvenations, Garg *et al.* [15] applied a Markov regenerative Petri net model. For analysing two rejuvenation policies (purely time based vs. instantaneous load & time based) in a transactions based software system, Garg *et al.* [16] used a queuing model. Dohi *et al.* [11]–[13] 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.* [37] considered two kinds of preventive maintenance in operational software systems. Likewise, Xie *et al.* [40] generalized the semi-Markov model presented in [13] 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 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] analysed the exhaustion of resources like real memory, and swap space in a network of UNIX workstations. All those metrics, observed on three computers, showed s-significant trends over time. Using a non-parametric technique, Garg *et al.* determined the global trends, and calculated the estimated time to exhaustion via linear extrapolation for each resource. Vaidyanathan and Trivedi [38] 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.

Shereshevsky *et al.* [34] chose a completely different approach to the analysis of aging in memory resources. Instead of modeling and predicting memory utilization directly, they monitored the Hölder exponent (a measure of the local rate of fractality) of the system parameters. Their findings indicated that system crashes are often preceded by the second abrupt increase in this measure.

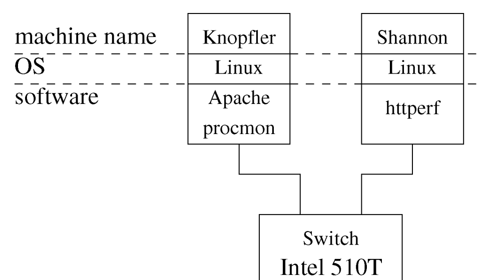

Fig. 1. Experimental setup.

In this paper, we analyse resource usage data collected on a typical long-running software system: a web server. Ideally, a web server should operate without any interruptions. However, due to unfixed bugs in the application or the OS, the system may show software aging. Some system administrators deal with this problem in a reactive manner, restarting the web server only after it has crashed or slowed down so much that it has become unusable. Many administrators proactively reboot the whole system or restart the web server program from time to time, setting the interval between restarts based on their experience. Because 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 Linux OS, 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 II, 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 analysed in Section III. Section IV contains concluding remarks, and discusses possible directions for future research.

II. Experiments

A. Experimental Setup

Apache is currently the most popular web server software [30]. The setup used for our experiments 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 structure of this experimental setup are illustrated in Fig. 1. For collecting resource usage information of the web server, we make use of the fact that the Linux OS stores an abundance of system information in the `/proc` virtual file system. For example, the file `/proc/meminfo` contains information

about the usage of physical memory and swap space, while information on the system load can be found in the file `/proc/loadavg`. From the `/proc` file system, we periodically extract information with the help of the Linux monitoring tool `procmon` developed in our research group by Rajiv Poonamalli. The parameters to be monitored, the interval at which `procmon` extracts their current values, and the format of the ASCII file in which `procmon` stores all data can easily be specified in a configuration file.

In our experiments, we use `httperf` [29] 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 on the server. The corresponding probabilities of accessing the files are 0.35, 0.5, 0.14, 0.009, and 0.001, respectively. `httperf` is not only a workload generator, but it can also be employed for monitoring performance information. The measurements provided include the reply rate (i.e., the number of responses received from the server per unit time), the response time (i.e., the interval from the time `httperf` sends out the first byte of a request until it receives the first byte of reply), and the number of timeout errors (i.e., the total number of requests for which no response was received from the server due to timeout errors). While the reply rate is a fundamental index of capacity; response time and timeout error rate (i.e., the number of timeout errors that occurred during the experiment per unit time) are important performance indicators of the web server.

B. Determining the Capacity of the Web Server

With the first experiment, we determine the capacity of our Apache web server. Toward 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 this child process has handled `MaxRequestsPerChild` requests. This behavior may be beneficial because "it 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" [2]. If `MaxRequestsPerChild` is set to its default value zero, then a process never expires; i.e., this value really represents infinity.

A second parameter, `MaxClients`, determines the number of child processes that can be running concurrently; i.e., it sets a limit on the number of clients that can simultaneously connect to the server. It makes sure that an overloaded Apache server does not create more child processes to serve requests, which would slow down and eventually completely overload the entire system. The default value of this parameter is 256 [2]. This is also the maximum value that can be chosen without changing the source code, and recompiling the Apache server, because in the Apache 1.3.14 distribution the compile-time constant `HARD_SERVER_LIMIT`, which acts as an upper bound for

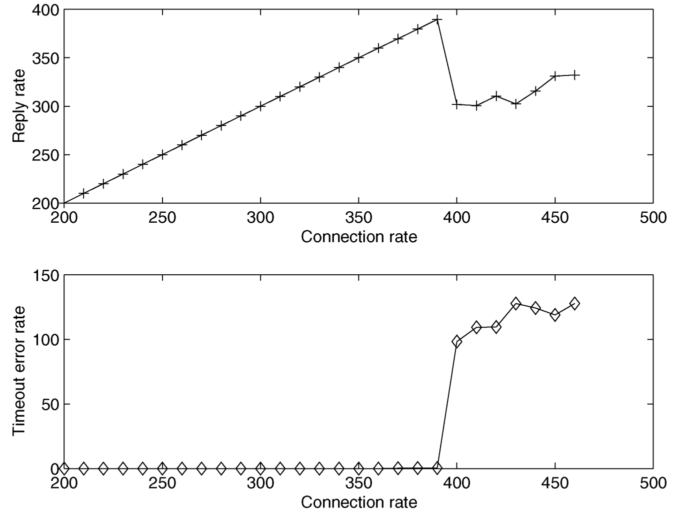

Fig. 2. Capacity of the web server.

TABLE I
WEB SERVER CAPACITY UNDER DIFFERENT CONFIGURATIONS

`MaxClients` → / `MaxRequestsPerChild` ↓	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

`MaxClients`, is set to 256 [1]. This constant limits the amount of (shared) memory that has to be reserved for several static data structures.

To assess the impact of `MaxRequestsPerChild` & `MaxClients` (within the pre-defined hard limit), we determine the capacity of the web server for different settings of the two configuration parameters, always using multiples of ten. For example, Fig. 2 depicts the results obtained for `MaxRequestsPerChild` $= 0$, and `MaxClients` $= 250$. Above the peak at 390 replies per second, the reply rate deviates from the connection rate. We therefore conclude that under this setting the capacity of the web server is about 390 requests per second. The results shown in Table I 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 then respawn child processes with a higher frequency, which increases the overhead on the system. Therefore, the capacity of the web server decreases. This finding agrees with the results obtained by Arlitt & Williamson [4]. Their experiments revealed that restricting the number of requests per

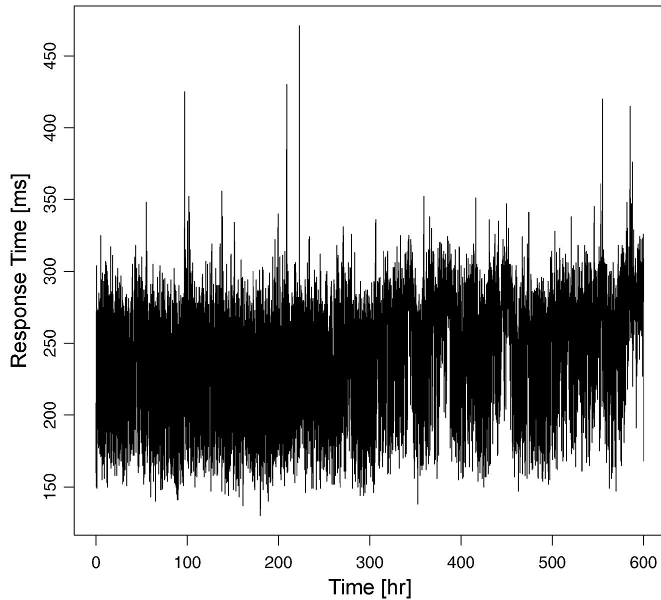

Fig. 3. Response time.

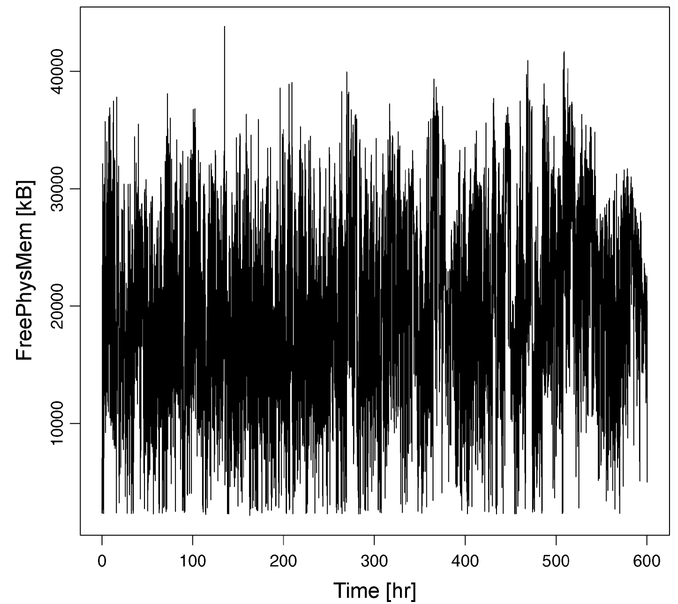

Fig. 4. Free physical memory.

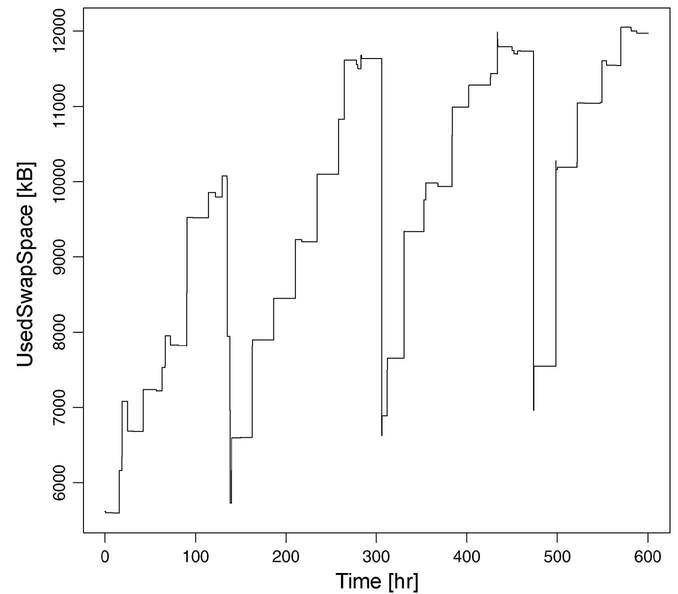

Fig. 5. Used swap space.

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 by a child process.

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 & respawning child processes. Both effects lead to a lower server capacity.

During the following analyses, we set MaxRequestsPerChild to 0, and MaxClients to 250.

C. 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), the free physical memory (FreePhysMem), and the used swap space (UsedSwapSpace). The three time series are shown in Figs. 3 to 5. Because the spacing between two consecutive data points is five minutes, each time series consists of 7208 observations.

From Fig. 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 for 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 spawn a new set of children to handle requests [3]. 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. Fig. 5 also shows that considerable increases in UsedSwapSpace often (but not always) occur at fixed intervals. This is especially clear for the second week of data collection, about 135 to 300 hours into the experiment. Again, this pattern is caused by the cron daemon; among the commands it invokes by default on a daily basis 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

TABLE II
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% s-confidence interval	(0.055 ms/hr, 0.067 ms/hr)	(7.287 kB/hr, 9.472 kB/hr)

within a short time. However, during our experiments, the web server is constantly in an overload state because 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 & termination of those processes that had been started but never finished throughout the week.

Fig. 5 suggests that even in the presence of this incidental rejuvenation responsible for the weekly pattern, there remains an overall trend in UsedSwapSpace. However, from the plot alone we cannot determine whether this residual aging is s-significant or not. Moreover, due to the variability as well as the existence of outliers, it is difficult to visually discern any trends from the plots of ResponseTime (Fig. 3), and FreePhysMem (Fig. 4). Because mere eyeballing does not suffice, we will apply statistical techniques for trend test and trend estimation in the following section.

III. STATISTICAL ANALYSIS OF RESOURCE-USAGE DATA

A. *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.

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 Figs. 3, and 4, respectively).

For testing the null hypothesis that a sample $y_1, y_2, \dots, y_n$ does not exhibit a trend, Mann [28] used a linear function of a test statistic originally developed by Kendall [25] for testing whether two sets of rankings are s-independent. The direct application of this test statistic W for our purposes is known as the Mann-Kendall test for trend. In this context, the value of the test statistic is computed as

$$w = \sum_{k=1}^{n-1} \sum_{l=k+1}^{n} \operatorname{sgn}(y_l - y_k).$$

Of all $n' = \binom{n}{2} = n(n-1)/2$ pairs of observations $y_k, y_l (k < l)$, w counts those 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 a value of w 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 w, tied pairs, i.e., those pairs for which $y_k = y_l$, are not taken into account. However, the existence of such tied pairs does 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 jth set, then the variance of W is given by [26, p. 43]

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

Under the null hypothesis, the distribution of W is always symmetric, and the expected value of W is equal to zero. Moreover, for n approaching infinity, the distribution of W converges to the s-normal distribution. Allowing for a continuity correction [26, pp. 41–42], the value of the test statistic

$$Z = \frac{W - \operatorname{sgn}(W)}{\sqrt{\operatorname{Var}(W)}} \tag{1}$$

can be compared to the quantiles of the standard s-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 II. Both are larger than $\lambda_{0.975} = 1.960$, the 97.5% quantile of the standard s-normal distribution. Consequently, in each case, the null hypothesis that the time series contains no trend can be rejected at a Type I error level (i.e., a long-term probability of rejecting the null hypothesis when it is true) of 5%. This means that, although we have not been able to visually discover any trends in Figs. 3 & 4, trends are present in the data, and these trends are s-significant. Moreover, the positive signs of the z values show that the two trends are increasing.

To determine estimates for the slopes, we apply a non-parametric procedure developed by Sen [32]. This method is not affected by outliers, and it is robust to missing data. Like the calculation of the value of the test statistic W, 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 $100 \cdot (1 - \alpha)\%$ s-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 s-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 $c_\alpha = \lambda_{1-\alpha/2}\sqrt{\operatorname{Var}(W)}$.

TABLE III
TREND TEST AND ESTIMATION FOR USEDSWAPSPACE

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

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% s-confidence intervals are shown in Table II. As anticipated after the calculation of the values of the Z statistic, the estimated slope is positive for both ResponseTime, and FreePhysMem. Moreover, none of the s-confidence intervals contains the value zero; this corroborates the earlier finding that the trends are s-significant at a Type I error level of 5%.

For ResponseTime, the result is consistent with the symptoms of software aging: the longer the web server has been operating, the more time it tends to need for reacting to a request. On the average, the response time as perceived by the user increases by about 0.06 ms every hour. While this value is rather small, we have to keep in mind that we are dealing with a long-running system. Because the slope estimate is based on more than 3.5 weeks worth of data, we can be sure that it reflects not merely some local variation but rather the average change that is in effect over a long period of time. For example, in the course of four weeks, the expected increase in the response time of one request is 41 ms, about 18% of the average response time of one request measured at the beginning of the experiments.

With regard to the FreePhysMem time series, we would have expected a decreasing trend rather than the increasing one that we detected. A possible explanation for the observed behavior is the fact that free physical memory in the system cannot be lower than a certain threshold. Because 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. Fig. 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.

2) Data With a Seasonal Pattern: For seasonal data, Hirsch *et al.* [23] 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 ith season, denoted by $y_{i1}, y_{i2}, \ldots, y_{in_i}$, where n_i is the total number of observations for this season, the value

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

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

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

assuming that the data of the ith season consist of o_i different values with the number of data points taking the jth value given by t_{ij}.

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

Because the automatic "rejuvenation" carried out by the system takes place once a week (i.e., after $7 \times 24 \times 60 = 10080$ minutes), and measurements are taken every five minutes, there are $m = 10080/5 = 2016$ seasons to be considered. The calculation of the values of the 2016 statistics W_i, and their variances, leads to a z of 68.443, which is larger than $\lambda_{0.975} = 1.960$, and therefore indicates the existence of a (positive) trend in the data at a Type I error level of 5%.

According to van Belle & Hughes [39], a different trend test based on a rank order test developed by Sen [33] 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 spanning more than 3.5 weeks, the computation can only be based on the $3 \times 2016 = 6048$ data points related to three full weeks, 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 a Type I error level of 5%.

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 analysed 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 $\operatorname{Var}(W)$ computed as the sum of the m variances $\operatorname{Var}(W_i)$, the $100 \cdot (1 - \alpha)\%$ s-confidence interval for the slope estimate is derived in the same way as described for the (basic) Sen's slope estimator.

For the slope of UsedSwapSpace, the estimate as well as the 95% s-confidence interval are listed in Table III. Due to the periods in which swap space usage remains constant, many of the individual slopes q_{ikl} are identical; this explains why the lower bound of the s-confidence interval has the same value

as the slope estimate itself. The fact that the s-confidence interval merely contains positive values confirms the earlier result that the amount of swap space used features an increasing global trend. The estimated slope is of the same magnitude as the one for free physical memory shown in Table II. In fact, its 95% s-confidence interval is completely contained in the 95% s-confidence interval derived for the latter quantity. This gives rise to the question of 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, we have to remember that the global trend already incorporates the effects caused by the weekly log rotation, which we discussed above in Section II-C. 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 considerably higher. For example, based on the 1619 earliest observations of the data set, collected in the 135 hours before the first weekly rejuvenation, Sen's slope estimate amounts to 34.442 kB/hr.

But even in the presence of the incidental rejuvenation, the depletion in swap space is appreciable. Although the overall slope of 7.714 kB/hr may seem negligible, the expected increase of swap space usage within four weeks is about 5184 kB, almost as much as the initial amount of used swap space at the beginning of the experiments. If this trend continues, the resource will ultimately be exhausted. To avoid this event, and the associated system failure, a more deliberate, effective software rejuvenation than the one connected to the log rotation may be called for. A good scheduling decision requires the prediction of future resource usage. In addition to the considerable "residual aging", used swap space also shows a lot of local variation. Therefore, it does not suffice to predict the amount of used swap space based on the overall slope alone: A sudden local increase might lead to a complete resource exhaustion. Because the local variation has a distinct weekly pattern, we will explore the applicability of seasonal time series models for fitting and forecasting swap space usage in the following section. We will also employ such models for FreePhysMem, and ResponseTime, to check our assumption that they do not feature any seasonality, as well as to determine whether there are any dependencies between consecutive observations.

B. Time Series Analysis

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_i are fixed parameters. It is assumed that the u_t, 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 analysed 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 $\gamma_1, \ldots, \gamma_m$, each representing the effect of one season. The resulting model has the form

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

with the dummy variable d_{jt} being equal to one if the tth observation belongs to the jth 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) because 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

$$\boldsymbol{y} = \boldsymbol{X\alpha} + \boldsymbol{D\gamma} + \boldsymbol{u}, \quad (3)$$

where

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

$\boldsymbol{\alpha} = (\alpha_1, \phi_1, \ldots, \phi_p)^T$, $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_m)^T$, 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 $[\boldsymbol{XD}]^T[\boldsymbol{XD}]$, 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, the residuals are used in a subsequent regression 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 $\mu_j = (2\pi j/m)$ $(j = 1, \ldots, \mathrm{gilb}(m/2))$ can be employed, modeling the seasonal influence at time t by [22, p. 41]

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

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 $\mathrm{gilb}(m/2)$, 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 sufficient to use just the lower-order frequencies $j = 1, 2, \ldots, f < \text{gilb}(m/2)$. The resulting model in vector form is

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

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 over-fitting. Therefore, model order selection criteria effectively penalize for the number of freely estimated parameters in the model, r. The Bayesian information criterion (BIC) proposed by Schwarz [31], for example, takes the form [27]

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

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 the most appropriate one.

For the model order selection in a model with deterministic terms, Lütkepohl [27] proposed to estimate these terms in a first step, subtract the estimated deterministic function from the data, and apply the order selection procedure to the adjusted data. However, we do not know the number of lower-order frequencies f to be used. We therefore employ the following approach that allows us to jointly optimize f & p: for all value combinations $(f, p) \in \{0, 1, \ldots, 100\}^2$, we fit the model to the data and calculate the BIC, replacing r by $(2 + 2 \cdot f + p)$. (The two additional parameters considered in the model order r are related to the mean, and the trend component.)

For our three time series, Table IV lists the model orders selected based on the BIC. To permit model validation, all calculations are merely based on the first 4000 observations of each time series, i.e., the data collected during roughly 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. While observations made 7 hours and 40 minutes ago $(92 \text{ obs} \times 5 \text{ min/obs})$ 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.

TABLE IV
SELECTED MODEL ORDERS

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

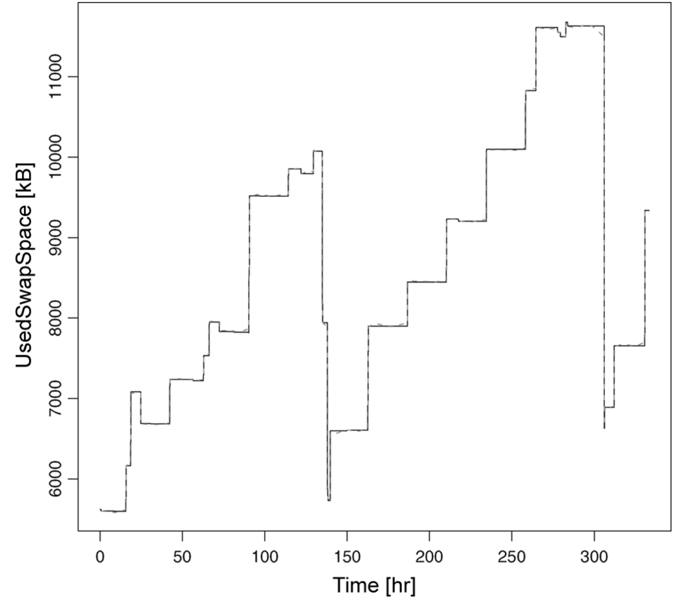

Fig. 6. UsedSwapSpace, first 4000 observations (—) and fitted values (- -).

The benefit 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 Fig. 6. 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.

To validate the model, we use the parameter estimates from the first 4000 observations, as well as the 4000th 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 Fig. 7. The comparison seems to indicate that the model is not only able to fit the data, but that it also provides a good forecast. Merely relying on the data collected during the first two weeks of experiments, we

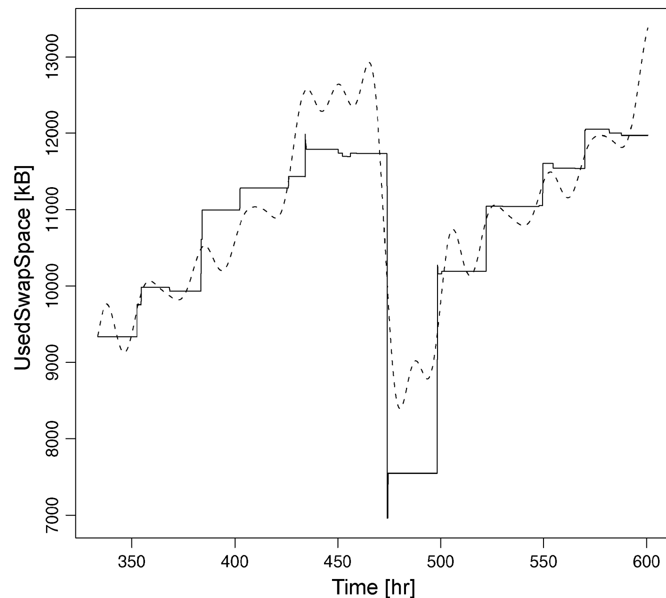

Fig. 7. UsedSwapSpace, last 3208 observations (—) and predicted values (- -).

could have predicted the main features of the time series for the following 1.5 weeks.

We note that the previous literature on software aging had never explicitly modeled the seasonality in resource usage and system performance. Neglecting such patterns may result in straightforward but inaccurate predictions, and hence in sub-optimal decisions regarding when to rejuvenate. Our results are encouraging, as they seem to indicate that a relatively simple, parsimonious model can produce adequate forecasts.

IV. 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 the settings of the web server itself, as well as those of the operating system.

For analysing the collected data, we employed both non-parametric statistical techniques, and parametric time series models. Because 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. They also indicate the potential for future 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. On the other hand, multivariate time series models can be employed for investigating the interactions between various system resources. The ultimate goal of all this research is an optimization model that uses the predictions of resource exhaustion as well as further information (e.g., the costs of planned, and unplanned downtime) for deriving the best rejuvenation schedule.

Acknowledgment

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.

References

[1] “Apache 1.3 API Documentation,” Apache Software Foundation [Online]. Available: http://httpd.apache.org/dev/apidoc/apidoc_HARD_SERVER_LIMIT.html
[2] ——, “Apache Core Features,” [Online]. Available: http://httpd.apache.org/docs/1.3/mod/core.html
[3] ——, “Stopping and Restarting Apache,” [Online]. Available: http://httpd.apache.org/docs/1.3/stopping.html
[4] M. Arlitt and C. Williamson, “Understanding web server configuration issues,” *Software—Practice and Experience*, vol. 34, no. 2, pp. 163–186, 2004.
[5] A. Avritzer and E. J. Weyuker, “Monitoring smoothly degrading systems for increased dependability,” *Empirical Software Engineering*, vol. 2, no. 1, pp. 59–77, 1997.
[6] L. Bernstein and C. M. R. Kintala, “Software rejuvenation,” *CrossTalk*, vol. 17, no. 8, pp. 23–26, 2004.
[7] G. Candea, J. Cutler, and A. Fox, “Improving availability with recursive microreboots: a soft-state system case study,” *Performance Evaluation*, vol. 56, no. 1–4, pp. 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*, 2002, pp. 478–482.
[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*, vol. 45, no. 2, pp. 311–332, 2001.
[10] “Software Rejuvenation,” Department of Electrical and Computer Engineering, Duke University [Online]. Available: http://www.software-rejuvenation.com/
[11] 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*, 2000, pp. 25–34.
[12] ——, “Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule,” in *Proc. International Pacific Rim Symposium on Dependable Computing*, 2000, pp. 77–84.
[13] ——, “Estimating software rejuvenation schedules in high assurance systems,” *Computer Journal*, vol. 44, no. 6, pp. 473–482, 2001.
[14] T. Dohi, K. Goševa-Popstojanova, K. Vaidyanathan, K. S. Trivedi, and S. Osaki, “Software rejuvenation: modeling and applications,” in *Handbook of Reliability Engineering*, H. Pham, Ed. London: Springer, 2003, pp. 245–263.
[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*, 1995, pp. 24–27.
[16] ——, “Analysis of preventive maintenance in transactions based processing systems,” *IEEE Trans. Computers*, vol. 47, no. 1, pp. 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*, 1998, pp. 283–292.
[18] R. O. Gilbert, *Statistical Methods for Environmental Pollution Monitoring*. New York: Van Nostrand Reinhold, 1987.
[19] J. Gray and D. P. Siewiorek, “High-availability computer systems,” *IEEE Computer*, vol. 24, no. 9, pp. 39–48, 1991.
[20] W. H. Greene, *Econometric Analysis*, 5th ed. Upper Saddle River: Prentice-Hall, 2003.

[21] M. Grottke and K. S. Trivedi, "Software faults, software aging and software rejuvenation," *Journal of the Reliability Engineering Association of Japan*, vol. 27, no. 7, pp. 425–438, 2005.

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

[23] R. M. Hirsch, J. R. Slack, and R. A. Smith, "Techniques of trend analysis for monthly water quality data," *Water Resources Research*, vol. 18, no. 1, pp. 107–121, 1982.

[24] 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*, 1995, pp. 381–390.

[25] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[26] ——, *Rank Correlation Methods*. London: Charles Griffin & Company Ltd., 1948.

[27] H. Lütkepohl, "Univariate time series analysis," in *Applied Time Series Econometrics*, H. Lütkepohl and M. Krätzig, Eds. Cambridge: Cambridge University Press, 2004, pp. 8–85.

[28] H. B. Mann, "Nonparametric tests against trend," *Econometrica*, vol. 13, no. 3, pp. 245–259, 1945.

[29] D. Mosberger and T. Jin, "httperf: a tool for measuring web server performance," in *Proc. First Workshop on Internet Server Performance*, 1998, pp. 59–67.

[30] Netcraft Ltd., Netcraft: February 2006 Web Server Survey [Online]. Available: http://news.netcraft.com/archives/2006/02/02/february_2006_web_server_survey.html

[31] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[32] P. K. Sen, "Estimates of the regression coefficient based on Kendall's tau," *Journal of the American Statistical Association*, vol. 63, no. 4, pp. 1379–1389, 1968.

[33] ——, "On a class of aligned rank order tests in two-way layouts," *Annals of Mathematical Statistics*, vol. 39, no. 4, pp. 1115–1124, 1968.

[34] M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu, "Software aging and multifractality of memory resources," in *Proc. International Conference on Dependable Systems and Networks*, 2003, pp. 721–730.

[35] 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*, 1991, pp. 2–9.

[36] A. T. Tai, L. Alkalaj, and S. N. Chau, "On-board preventive maintenance: a design-oriented analytic study for long-life applications," *Performance Evaluation*, vol. 35, no. 3–4, pp. 215–232, 1999.

[37] 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*, 2002, pp. 286–295.

[38] K. Vaidyanathan and K. S. Trivedi, "A comprehensive model for software rejuvenation," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 124–137, 2005.

[39] G. van Belle and J. P. Hughes, "Nonparametric tests for trend in water quality," *Water Resources Research*, vol. 20, no. 1, pp. 127–136, 1984.

[40] W. Xie, Y. Hong, and K. S. Trivedi, "Analysis of a two-level software rejuvenation policy," *Reliability Engineering & System Safety*, vol. 87, no. 1, pp. 13–22, 2005.

Michael Grottke received the M.A. degree in economics (1999) from Wayne State University, Detroit, MI, and the Diploma degree in business administration (2000) as well as the Dr. rer. pol. degree in statistics (2003) from the University of Erlangen-Nuremberg, Nürnberg, Germany. For his dissertation, he was awarded the Promotional Award for Science of the State Bank of Bavaria. He is currently working as a research associate in the Department of Electrical and Computer Engineering at Duke University, Durham, NC, on a Fellowship from the German Academic Exchange Service. His research interests include software reliability, software process maturity, software rejuvenation, as well as stochastic point processes and combinatorial problems.

Lei Li received the B.S. (1996), and M.S. (1999) degrees in electrical engineering from Peking University, Beijing, China; as well as the M.S. (2002), and Ph.D. (2004) degrees in electrical and computer engineering from Duke University, Durham, NC. He is currently working in the Wireless Group of Freescale Semiconductor Inc., Austin, TX, where he is a Senior Engineer. His interests are in the field of integrated circuit design, verification, and design-for-test.

Kalyanaraman Vaidyanathan received the B.E. degree in computer science (1996) from the University of Madras, India, and the M.S. (1999) and Ph.D. (2002) degrees in electrical and computer engineering from Duke University. He was a recipient of the IBM Graduate Fellowship Award in 2000. His research interests include software reliability, and performance and dependability evaluation of computer systems. He is currently a research engineer in the Scalable Systems Group, Sun Microsystems, San Diego, CA, exploring proactive fault monitoring techniques through telemetry and pattern recognition.

Kishor S. Trivedi holds the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He has been on the Duke faculty since 1975. He is the author of a well-known text entitled Probability and Statistics with Reliability, Queuing, and Computer Science Applications with a thoroughly revised second edition being published by John Wiley. His research interests are in reliability and performance assessment of computer and communication systems. He has made seminal contributions in software rejuvenation, solution techniques for Markov chains, fault trees, stochastic Petri nets, and performability models. He has actively contributed to the quantification of security and survivability.