A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems

Kalyanaraman Vaidyanathan and Kishor S. Trivedi
Center for Advanced Computing & Communication*
Dept. of Electrical & Computer Engineering
Duke University
Durham, NC 27708, USA
{kv,kst}@ee.duke.edu

Abstract

Software systems are known to suffer from outages due to transient errors. Recently, the phenomenon of "software aging" [5, 10], one in which the state of the software system degrades with time, has been reported. The primary causes of this degradation are the exhaustion of operating system resources, data corruption and numerical error accumulation. This may eventually lead to performance degradation of the software or crash/hang failure or both. Earlier work in this area to detect aging and to estimate its effect on system resources does not take into account the system workload [5]. In this paper, we propose a measurement-based model to estimate the rate of exhaustion of operating system resources both as a function of time and the system workload state. The semi-Markov reward model is constructed based on workload and resource usage data collected from the UNIX operating system. We first identify different workload states using statistical cluster analysis and build a state-space model. Corresponding to each resource, a reward function is then defined for the model based on the rate of resource exhaustion in the different states. The model is then solved to obtain trends and the estimated exhaustion rates and time to exhaustion for the resources. With the help of this measure, pro-active fault management techniques such as "Software rejuvenation" [10] may be employed to prevent unexpected outages.

1. Introduction

Several studies have now shown that outages in computer systems are more due to software faults than due to hardware faults [7, 22]. Software systems are known to suffer from outages due to transient errors. Recent studies have also reported the phenomenon of "software aging" [5, 10] in which the state of the software degrades with time. The primary causes of this degradation are the exhaustion of operating system resources, data corruption and numerical error accumulation. Eventually, this may lead to performance degradation of the software or crash/hang failure or both. Some common examples of "software aging" are memory bloating and leaking, unreleased file-locks, data corruption, storage space fragmentation and accumulation of round-off errors [5]. Aging has not only been observed in software used on a mass scale but also in specialized software used in high-availability and safety-critical applications [10]. Since aging leads to transient failures in software systems, environment diversity can be employed pro-actively to prevent degradation or crashes. This involves occasionally stopping the running software, "cleaning" its internal state and restarting it. Such a technique known as "software rejuvenation" is proposed in [10]. This counteracts the aging phenomenon in a pro-active manner by removing the accumulated error conditions and freeing up operating system resources. Garbage collection, flushing operating system kernel tables and reinitializing internal data structures are some examples by which the internal state of the software can be cleaned.

Earlier work in this area to detect/validate the aging phenomenon and quantify its effect on operating system resources does not take into account the system workload [5]. Several studies have investigated the workload dependency on system failures and reliability [12, 13] and have reported significant correlations. Hence, in this paper, we propose a measurement-based model to estimate the operating system resource exhaustion trends both as a function of time and the system workload state. The semi-Markov model is constructed based on workload and resource usage data collected from the UNIX operating system. We first iden-

* CACC core project supported in part by NSF, Bellcore and DoD

tify different workload states using statistical cluster analysis and build a state-space model. Corresponding to each resource, a reward function is then defined for the model based on the rate of resource exhaustion in the different states. The model is then solved to obtain trends and estimated rates and time to exhaustion of the resources. The results are then compared with the results obtained by using the purely time-based approach. The main contributions of this paper are (1) a measurement-based model for capturing the effect of system workload on operating system resources and (2) investigating the effect of workload on system resources, particularly with respect to exhaustion.

The rest of this paper is organized as follows. Section 2 gives an overview of related work in measurement-based software dependability evaluation and brings out some differences between this work and earlier work. A brief overview of performability analysis using semi-Markov reward models is given in Section 3. The experimental setup and data collection are briefly explained in Section 4. Section 5 discusses the clustering methodology for system workload characterization and identification of different workload states. Modeling of resource usage is explained in Section 6. The model behavior and results are discussed in Section 7, and Section 8 concludes the paper.

2. Related Work

Garg et. al. [5] present a general methodology for detecting and estimating trends and times to exhaustion of operating system resources due to software aging. The data collection methodology used in this work including the workload and resource usage variables monitored are the same as in their work. Whereas in [5] only time based trend detection and estimation of resource exhaustion are considered, we also take the system workload into account for building our model. Other previous work in measurement-based dependability evaluation is based on either measurements made at failure times [1, 12, 15] or at error observation times [13, 23, 24]. In our case, we need to monitor the system parameters continuously since we are interested in trend estimation and not in inter-failure times or identifying error patterns. Some of the above papers deal with hardware failures while we are concerned solely with software failures and in particular, failures due to resource exhaustion.

In [16], system parameters are constantly monitored to detect anomalies automatically. This is based on the premise that an anomaly or a deviation from "normal" behavior is usually a symptom of error. Their study focuses on network failures and smoothing techniques are applied to build a normal behavior. Our work concentrates on detecting and estimating trends rather than sudden abrupt changes in system parameter values.

Iyer at. al. [12] study the effect of system workload on failures through a measurement-based analysis and report significant correlations between permanent failures and increased system activity. A methodology for recognizing the symptoms of a persistent problems is proposed in [13]. They use a methodology for recognizing the symptoms of a persistent problem in large systems by identifying and statistically validating recurring patterns among error records produced in the system.

Clustering methods have been used in workload and resource usage studies [3, 11]. In [11], a semi-Markov reward model is used to estimate the cost of different types of errors. The clustering method they use is very similar to ours except that we concentrate on system activity parameters for clustering whereas they use parameters like CPU usage and memory usage as indicators of workload. Also, in their work, reward rates are assigned based on error rates while in our case, reward rates are attached to all states reflecting the rate of resource consumption in that state. A statistical approach based on clustering is used in [3] to predict resource usage levels for different kinds of processes. In [3], as in [11], the clustering is done on resource usage variables.

Markov and semi-Markov reward models have been used for performability analysis for computer systems rather widely [2, 17]. In [2], algorithms for computing the distribution of performability in a semi-Markov reward process have been proposed and in [17] performance and reliability of a system with a cumulative down time constraint is analyzed.

3. Semi-Markov Reward Models

For a homogeneous, continuous-parameter Markov chain, the sojourn time, i.e., the amount of time spent in a state, is exponentially distributed. If this restriction is removed and any distribution of the sojourn time is allowed, we get a class of models called *semi-Markov processes* (SMPs). In such a process, the rate of transition from state i to state j may depend on how long the chain has already been in state i. The memoryless property exhibited by the exponential distribution thus does not apply, but the transition rate still does not depend on anything that happened before the chain reached state i.

More formally, a semi-Markov chain is defined by a matrix **Q**(t), where each time-dependent entry $Q_{ij}(t)$ is the probability that after making a transition into state i, the chain next makes a transition into state j in an amount of time less than or equal to t [19]. The time t is the local time, i.e., measured from the time of entry into a state, and the matrix **Q**(t) is called the *kernel* of the semi-Markov process.

Another way to describe a semi-Markov chain , which applies only to so called independent SMPs [18], is to define a matrix **P** and a vector **H**(t). The previous definition using the matrix **Q**(t) is more general than this one. This is called the two-stage method [19] since the transitions of the SMP can be thought of taking place in two stages. Consider a transition from state i to state j. In the first stage, the chain stays in state i for some amount of time described by the sojourn time distribution $H_i(t)$. In the second stage, the chain moves to state j determined by the probability p_{ij}. We have used this two-stage method for constructing our model.

Markov or semi-Markov models are often solved for either steady-state or transient state probabilities. Weighted sums of state probabilities are then used to obtain measures of interest, with the weight attached to each state in reliability/availability models being either 0 or 1. If we extend the set of allowable weights to attach any[1] real number, called the *reward rate*, to each state of a Markov/semi-Markov chain, we obtain a Markov/semi-Markov reward process. A reward model has a structure state process that characterizes the evolution of the system through a set of states and a reward structure that characterizes the performance level associated with each state [21]. In our model, the structure state process captures the system workload dynamics and the reward structure characterizes the resource consumption rate for each state.

4. Experimental Setup and Data Collection

4.1 SNMP-based Distributed Resource Monitoring Tool

The SNMP (Simple Network Management Protocol)-based distributed resource monitoring tool described in [5] was used for our data collection. SNMP is an application protocol which offers network management services in the Internet protocol suite. The manager, the agent and the MIB (Management Information Base) form the main constituents of an SNMP-based management tool. The SNMP protocol defines a client-server relationship between a manager and an agent, and the MIB describes the information that can be obtained and/or modified through interactions between the manager and the agent.

4.2. Data Collection

The resource monitoring tool referred to previously was used to collect operating system resource usage and system activity data from nine heterogeneous UNIX workstations which were connected by an Ethernet LAN at the Duke Department of Electrical and Computer Engineering. In our setup, shown in Figure 1, a central monitoring station runs the manager program which sends *get* requests periodically to each of the agent programs running on the monitored workstations. The agent programs in turn obtain data for the manager from their respective machines by executing various standard UNIX utility programs like *pstat*, *iostat* and *vmstat*. Also shown in the figure along with the monitored workstations are their respective operating systems and primary functions in the department.

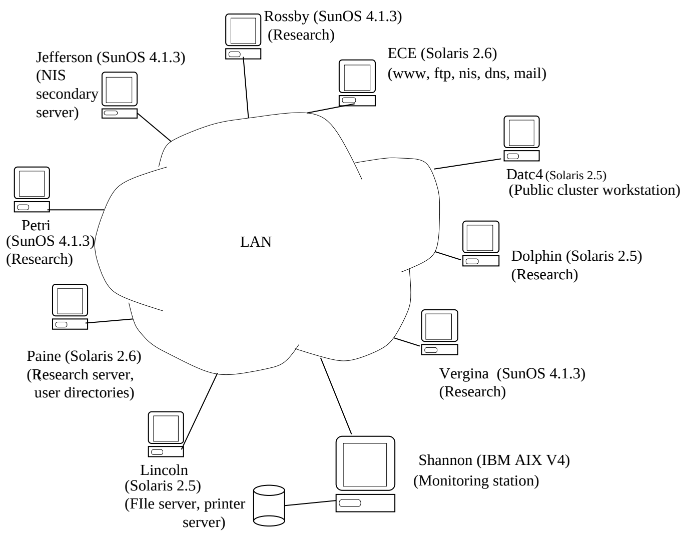

Figure 1. Experimental setup for data collection

The objects or parameters monitored on the workstations include those that describe the state of the operating system resources, state of the processes running, information on the /tmp file system, availability and usage of network related resources and information on terminal and disk I/O activity. More than 100 such parameters were monitored at regular intervals (10 min) for over 3 months.

In this paper, we only discuss the results for the data collected from the machine Rossby. The two resources selected for study are *usedSwapSpace* and *realMemoryFree*. Also, only the *longest* stretch of sample points in which no reboots or failures occurred were used for building the model.

5. Workload Characterization and Modeling

Since our objective is to estimate the effects of software aging on system resources as a function of the system workload, an important issue here is workload characterization and modeling. The system workload was characterized by obtaining a number of variables pertaining to CPU activity and file system I/O. We then identify a small number

[1] Reward rates are usually non-negative, but in our model we allow for negative reward rates also.

of representative workload states through statistical clustering techniques. Once the states are identified, the transition probabilities from one state to another and the distribution of sojourn time in each state are determined. Thus we obtain a complete state-transition model to describe the system workload dynamics.

In this study, we use the following variables to characterize the workload:

- *cpuContextSwitch* : The number of process context switches performed during the measurement interval
- *sysCall* : The number of system calls made during the interval
- *pageIn* : The number of page-in operations (pages paged in from file system or swap device) during the interval
- *pageOut* : The number of page-out operations (pages paged out from file system or swap device) during the interval.

Therefore, during any time interval, a point in a four-dimensional space, (*cpuContextSwitch, sysCall, pageIn, pageOut*), represents the measured workload. We next partition the data points into clusters which contain similar points based on some pre-defined criteria. A statistical clustering algorithm is used for achieving this.

5.1. Cluster Analysis

The goal of cluster analysis is to determine a partition of a given set of points into groups or clusters such that the points in a single cluster are more similar, according to a certain criterion, to each other than to points in a different cluster. In our case, we have used an iterative non-hierarchical clustering algorithm called the *Hartigan's k-means clustering algorithm* [8]. The objective of this algorithm is to divide a given set of points into *k* clusters so that the within-cluster sum of squares is minimized. The algorithm starts with *k* initial points (which may be random or pre-specified) which are taken to be centroids of *k* clusters. New points are assigned to these clusters based on the closest centroid and centroids are recomputed when all the points have been assigned. This procedure is repeated several times with the means from the one iteration taken to be the initial points for the next iteration. In other words, the algorithm finds a partition $\Pi = (C_1, C_2, ..., C_k)$ with *k* non-empty clusters such that the sum of the squares, *S* from each point to its corresponding centroid (center of mass) is minimized, i.e.

$$minimize\ S = \sum_{i=1}^{k} \sum_{x_j \in P_i} \mid x_j - \bar{x_i} \mid^2$$

where $x_{ij} \in P_i$ and $\bar{x}_i$ is the centroid of cluster P_i.

If the variables for clustering are not expressed in homogeneous units, a scale change or normalization must be performed. In our analysis, we used the normalization method based on the following transformations [4]:

$$x'_i = \frac{x_i - min_i\{x_i\}}{max_i\{x_i\} - min_i\{x_i\}}$$

where x'_i is the normalized value of x_i, $max_i\{x_i\}$ is the maximum of x_i and $min_i\{x_i\}$ is the minimum of x_i. This transformation restricts the values of all the variables to the range 0 to 1. We need to eliminate the outliers in the data before applying the scaling technique since they tend to distort the transformation. Outliers in the data were identified and eliminated by an analysis of the cumulative distribution of each parameter, although they were assigned to clusters in the final stage. The statistics for the workload variables measured are shown in Table 1.

Table 1. Statistics for the workload variables measured

Variable	Min	Median	Mean	Max
cpuContextSwitch	5598	10990	20650	386100
sysCall	19170	37960	41580	672900
pageIn	0	9	26.17	2522
pageOut	0	0	5.426	6227

The k-means clustering algorithm was applied to the workload data and this resulted in eleven clusters. The statistics for the eleven workload clusters are shown in Table 2. Also shown in the table are the percentage of the sample data points in each cluster. We observe that more than 75 percent of the points belong to clusters 7, 8 and 10 which are relatively light workload states. Clusters 1, 2, 3 and 11 are high workload states that contain significantly less data points.

5.2. The State-Transition Model

The next step, after the clusters and centroids are identified, is to build a state-transition model for the system workload. This is done by determining the transition probabilities from one state to another. The transition probability p_{ij} from a state i to a state j can be estimated from the sample data using the following formula [11]:

$$p_{ij} = \frac{observed\ no.\ of\ transitions\ from\ state\ i\ to\ state\ j}{total\ observed\ no.\ of\ transitions\ from\ state\ i}$$

Before this, the number of workload states was reduced to eight by merging clusters {1,2,3} and {4,5}. Thus we get

Table 2. Statistics for the workload clusters

Cl.	Cluster Center				%
no.	cpuCxtSwitch	sysCall	pageOut	pageIn	pts
1	48405.2	94194.7	5.2	677.8	1.0
2	54184.6	122229.7	5.4	81.4	0.8
3	34059.6	193927.0	0.0	136.7	1.0
4	20479.2	45811.7	0.5	243.4	1.9
5	21361.4	37027.4	0.3	12.6	7.2
6	15734.7	54056.3	0.3	14.5	6.6
7	37825.8	40912.2	0.9	12.2	11.8
8	11013.2	38682.5	0.0	10.4	42.9
9	67290.8	37246.8	7.6	19.9	4.9
10	10003.9	32067.2	0.0	9.6	21.2
11	197934.4	67822.5	415.7	184.4	1.0

$W_1 = \{1,2,3\}$, $W_2 = \{4,5\}$, $W_3 = \{6\}$, $W_4 = \{7\}$, $W_5 = \{8\}$, $W_6 = \{9\}$, $W_7 = \{10\}$ and $W_8 = \{11\}$. Clusters considered for merging were clusters whose centroids were relatively close to each other and clusters with a small percentage of data points in them. This was done mainly to reduce and simplify computations. State transition probabilities were computed for these eight states and the resulting (8 X 8) transition probability matrix, P, of the embedded discrete time Markov chain, is shown below:

$$P = \begin{bmatrix} 0.00 & 0.16 & 0.22 & 0.13 & 0.26 & 0.03 & 0.17 & 0.03 \\ 0.07 & 0.00 & 0.14 & 0.14 & 0.32 & 0.03 & 0.31 & 0.00 \\ 0.12 & 0.26 & 0.00 & 0.10 & 0.43 & 0.00 & 0.11 & 0.02 \\ 0.15 & 0.36 & 0.06 & 0.00 & 0.10 & 0.22 & 0.09 & 0.03 \\ 0.03 & 0.07 & 0.04 & 0.01 & 0.00 & 0.00 & 0.85 & 0.00 \\ 0.07 & 0.16 & 0.02 & 0.54 & 0.12 & 0.00 & 0.02 & 0.07 \\ 0.02 & 0.05 & 0.00 & 0.00 & 0.92 & 0.00 & 0.00 & 0.00 \\ 0.31 & 0.08 & 0.15 & 0.23 & 0.08 & 0.15 & 0.00 & 0.00 \end{bmatrix}$$

5.3. Sojourn Time Distribution

To completely specify the semi-Markov process, we need to determine the distribution of sojourn time in each state. This distribution for all the workload states was fitted to either 2-stage hyper-exponential or 2-stage hypo-exponential distribution functions. The sojourn time in workload states W_1, W_5 and W_7 was fitted to hypo-exponential distributions, while the sojourn time in all other states was fitted to hyper-exponential distributions. The fitted distributions were tested using the Kolmogorov-Smirnov test at a significance level of 0.01. Figure 2 shows the Kolmogorov-Smirnov test for sojourn time distribution in workload state 4 and the fitted distributions for all the workload states are listed in Table 3.

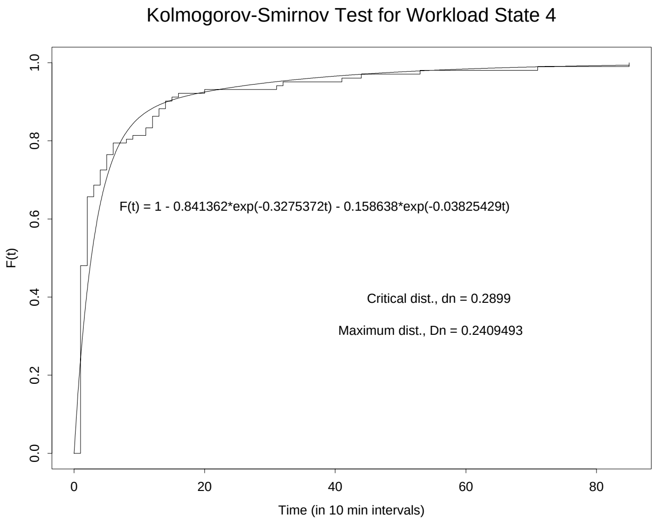

Figure 2. Kolmogorov-Smirnov test for sojourn time distribution in W_4

Table 3. Sojourn time distributions in the eight workload states

State	Sojourn Time Distribution, $F(t)$
W_1	$1 - 1.602919e^{-0.9t} + 0.6029185e^{-2.392739t}$
W_2	$1 - 0.9995e^{-0.4459902t} - 0.0005e^{-0.007110071t}$
W_3	$1 - 0.9952e^{-0.3274977t} - 0.0048e^{-0.0175027t}$
W_4	$1 - 0.841362e^{-0.3275372t} - 0.158638e^{-0.03825429t}$
W_5	$1 - 1.425856e^{-0.56t} + 0.4258555e^{-1.875t}$
W_6	$1 - 0.80694e^{-0.5509307t} - 0.19306e^{-0.03705756t}$
W_7	$1 - 2.86533e^{-1.302t} + 1.86533e^{-2t}$
W_8	$1 - 0.9883e^{-0.2655196t} - 0.0117e^{-0.02710147t}$

5.4. Model Validation

The semi-Markov model for the system workload needs to be validated before use. The validation was done in the same manner as done in [11]. The steady state probability of occupying a particular workload state computed from the model was compared to the actual probability from the observed data, i.e. the fraction of the length of time the system was in that workload state to the total length of the period of observation. The results are shown in Table 4.

It can be seen that the computed values from the model and the actual observed values match quite closely. This validates our model building methodology and so the semi-Markov process obtained can be taken to model the real system workload reasonably well.

Table 4. Comparison of state occupancy probabilities (expressed as percentage)

State	Observed value	Value from model	% Difference
W_1	2.664146	2.8110	5.512235
W_2	9.058096	8.3464	7.857015
W_3	6.548642	6.0576	7.498379
W_4	11.77381	10.8480	7.863300
W_5	42.86696	44.4310	3.648591
W_6	4.932967	4.5767	7.222165
W_7	21.22723	22.1030	4.125691
W_8	0.928154	0.82577	11.030928

6. Modeling Resource Usage

The previous section discussed the development of a semi-Markov process to describe the system workload. Since our original aim was to estimate the exhaustion of system resources as a function of workload, we need to incorporate the effect of workload on the resources in this model. Therefore, a reward function corresponding to each system resource considered for analysis is assigned to the model. The reward rate, r_{WR}, for *each* workload state, W and for *each* resource, R, is computed as the slope (rate of increase/decrease per unit time interval) of the resource R in the workload state W. We consider two resources here for our analysis - *usedSwapSpace* and *realMemoryFree*. The time plots of these two resources in machine Rossby between successive reboots/failures are shown in Figure 3. The time interval for which the plots are shown for the resources, corresponds to the time interval for which the workload data used for building the state-transition model (in the previous section) was sampled from.

6.1. Computing the slope

If a linear trend is present in the data, linear regression methods can be used to estimate the true slope by computing the least squares estimate of the slope. The slope obtained by this method can deviate greatly from the true slope if there are gross errors or outliers in the data [6]. Therefore, we estimate the true slope of a resource at every workload state by using a non-parametric procedure developed by Sen [20]. Sen's method is not greatly affected by gross data errors or outliers, and it can be computed even with missing data.

The procedure for estimating the slope by this method is as follows. First, N' slope estimates are calculated for all pairs of points at i and j for which $i > j$, as $Q_{ij} = (x_i - x_j)/(i - j)$. We thus obtain $N' = n(n-1)/2$ slope estimates, where n is the total number of data observations. The median of these N' values is the required slope estimate. It is also possible to obtain a two-sided confidence interval about the true slope.

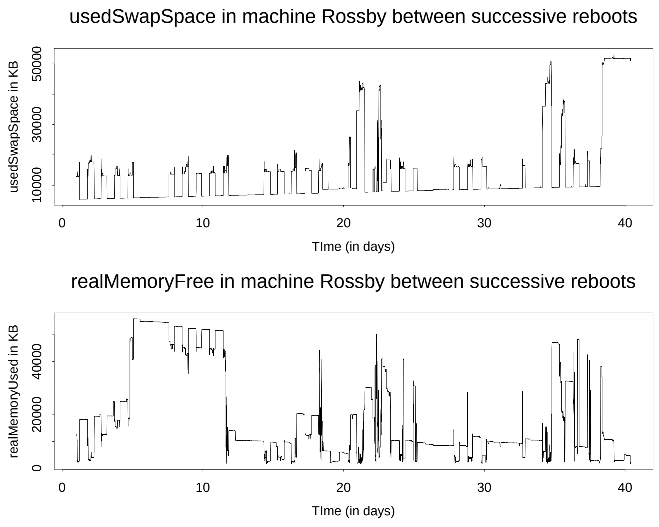

Figure 3. Time plots of resources *usedSwapSpace* and *realMemoryFree* in machine Rossby

For a particular workload state, it is possible to obtain the slope of a resource during different visits to the state. We expect the different slopes thus obtained to be relatively close. Figures 4 and 5 show the slopes of *usedSwapSpace* and *realMemoryFree* respectively at workload states 4 during two different visits. The 95 % confidence interval is given in brackets beside the slope estimates. We observe that the slope in both the instances are almost the same. Table 5 shows the slope obtained by Sen's method for these two resources, for all the eight workload states. The slopes shown in this table are the slopes of the respective resources during the *longest* interval for which the system was at that workload state. Thus, in our model, these slopes correspond to the reward rates for each workload state for *usedSwapSpace* and *realMemoryFree*.

The observation that slopes in a given workload state are almost the same (refer to Figures 4 & 5), together with the observation that slopes across different workload states are different (refer to Table 5) validates our assumption that resource usage *does* depend on the system workload and the rates of exhaustion vary with workload changes. We observe that the slopes for *usedSwapSpace* in all the workload states are non-negative, and the slopes for *realMemoryFree* are non-positive in all the workload states except in one. This shows that *usedSwapSpace* increases whereas *realMemoryFree* decreases over time. This validates the

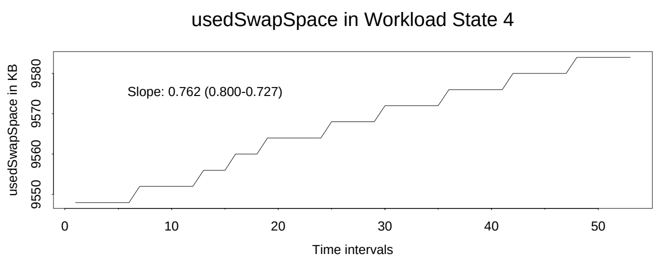

Figure 4. Slope (in KB/10 min) of *usedSwapSpace* **in** W_4 **during two different visits**

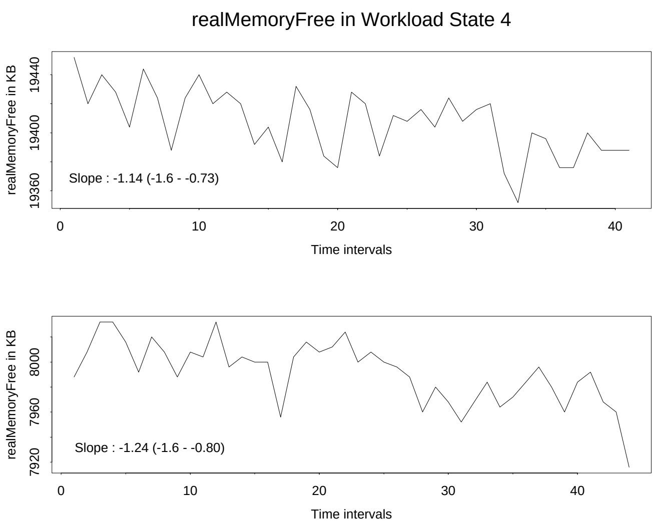

Figure 5. Slope (in KB/10 min) of *realMemoryFree* **in** W_4 **during two different visits**

"software aging" phenomenon described in [10, 5]. It is also generally observed that higher the system activity, higher is the resource utilization. The 95% confidence intervals for the slope are high for *usedSwapSpace* in workload state W_1 and for *realMemoryFree* in workload state W_8. This just reflects a high degree of variability.

7. Results

The semi-Markov reward model describing the system workload and resource utilization was solved using SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) [19] developed at Duke University. The irreducible semi-Markov reward model was broken down into an irreducible Markov reward model before solving since all state sojourn times have phase-type distributions. Measures like expected reward rate at steady state, expected instantaneous reward rate, cumulative expected reward at time t and time-averaged accumulated reward at time t were obtained.

7.1 Estimating trends and exhaustion rates

The expected accumulated reward over time with respect to a particular resource can be taken to be an estimator of the resource usage trend. Figures 6 and 7 show the time plots of *usedSwapSpace* and *realMemoryFree* in machine Rossby along with the workload and time based estimations. The workload based estimation is computed as the cumulative expected reward over time from the model. The time based estimation is computed using the Sen's slope estimate for data with seasons as proposed in [5]. This is a purely time based estimate approach since workload is not considered. Here, a day is considered a season and a week, a cycle. The initial points for all the estimations is taken to be the average value over the first few days.

It is apparent that workload based estimations give better trend estimates than just time based estimations. In Figure 6, the upper confidence level trend estimate is very close to the peaks in actual *usedSwapSpace* and in fact, almost coincides with it in the final time period. Hence this can be used a good estimator for the peaks in resource usage. The lower confidence level trend estimate almost coincides with the time based estimation. Therefore this gives the general overall trend without being affected by the peaks. The reason that peaks in the data are more or less covered by the workload based estimation is that workload model takes into account those workload states which are likely to generate peaks in the data. Hence the slope estimated by the workload model is larger. The time based estimation removes these peaks and gives us only a general overall trend.

In Figure 7 too, we get a slope estimated using system workload being better than that estimated using Sen's method. The workload based trend estimation coincides with the final values of *realMemoryFree* and the upper confidence level trend estimate coincides with the final peak values of *realMemoryFree*.

Table 6 gives the estimates for slope and time to exhaustion for *usedSwapSpace* and *realMemoryFree* computed using both the time based and the workload based approaches. The slope for the workload based estimation is computed as

Table 5. Slope estimates (in KB/10 min) for *usedSwapSpace* and *realMemoryFree* at different workload states

State	usedSwapSpace		realMemoryFree	
	Slope Est.	95 % Conf. Interval	Slope Est.	95 % Conf. Interval
W_1	119.3	5.54 - 222.4	-133.7	-137.7 - -133.3
W_2	0.57	0.40 - 0.71	-1.47	-1.78 - -1.09
W_3	0.76	0.73 - 0.80	-1.43	-2.50 - -0.62
W_4	0.57	0.00 - 0.69	-1.23	-1.67 - -0.80
W_5	0.78	0.75 - 0.80	0.00	-5.65 - 6.00
W_6	0.81	0.64 - 1.00	-1.14	-1.40 - -0.88
W_7	0.00	0.00 - 0.00	0.00	0.00 - 0.00
W_8	91.8	72.4 - 110.9	91.7	-369.9 - 475.2

the expected reward rate at steady state from the model. The 95% confidence interval for the workload based slope estimations is larger since it allows for a larger degree of variability that possibly takes into account peaks in the data.

The estimated times to exhaustion was computed using the linear formula $y = mx + c$, where m is the slope, c is the intercept or the initial values and y is the final (maximum or minimum) value [5]. The maximum value for *usedSwapSpace* in machine Rossby was 312724 KB and the minimum value for *realMemoryFree* was taken to be 0. We find that workload based estimations give a lower time to resource exhaustion than the corresponding slopes computed using time based estimations. This is true since it was observed that machines failed due to resource exhaustion much before the times to resource exhaustion estimated by the time based method. A comparison between the estimated times to exhaustion can help us ascertain which resources are important for us to monitor and manage; in this case, *realMemoryFree* being the more important or critical resource.

As pointed out in [5], these estimated times to exhaustion take into account the effect of aging on a particular resource alone and does not coincide with the actual failure times of the monitored machines. This may be due to several factors like the interaction between various resources and transient effects, which are not taken into account in our model. Furthermore, the machines might exhibit new kinds of failures not captured by the model. However, these measures are helpful to study and understand the effects of aging on various resources and may be used to develop new fault-management techniques. These become more important considering the fact that "software aging" plays a significant part in software failures.

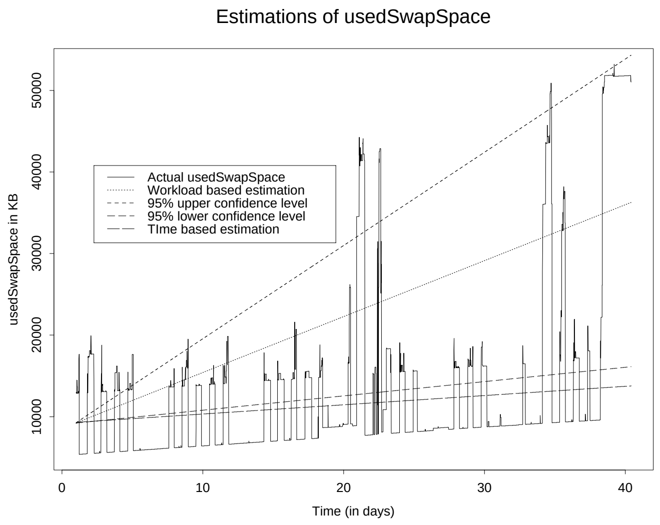

Figure 6. Time plot and trend estimations of resource *usedSwapSpace* in machine Rossby

8. Conclusions

In this paper, we proposed a methodology for constructing a model for estimating trends, rate and time to exhaustion of system resources based on resource usage and system activity data collected from the UNIX operating system. The main contribution of our work is a measurement-based model which incorporates the effect of system workload on operating system resources and a methodology to investigate its effect on resource exhaustion. Since many studies have suggested strong correlations between workload and system reliability/availability, this model is an improvement over the purely time based model proposed in [5]. We were able to clearly demonstrate the relation between the system workload and resource exhaustion and thus validate our assumption that workload does affect the way various system resources are utilized. Not only was the system workload dynamics captured in the model but also the effect of workload on resource usage was quantified by means of reward rates or slopes in the model. In doing this, we have also validated the "software aging" phenomenon with respect to resource exhaustion. The quantification of resource exhaustion for different workload states helps us in obtaining a better estimation for exhaustion rates and time to exhaustion than just time based estimation. The same kind of analysis of relative importance among different resources and estimated time to resource exhaustion, as done in [5], can be done here with greater accuracy. Although, the estimations obtained from these methodologies cannot be taken to be estimations of actual machine failure times since they might depend on various other factors, all these

Table 6. Estimates for slope (in KB/10 min) and time to exhaustion (in days) for *usedSwapSpace* and *realMemoryFree*

Method of Estimation	usedSwapSpace			realMemoryFree		
	Slope Estimate	95 % Conf. Interval	Est. Time to Exhaustion	Slope Estimate	95 % Conf. Interval	Est. Time to Exhaustion
Time based	0.787	0.786 - 0.788	2276.46	-2.806	-3.026 - -2.630	60.81
Workload based	4.647	1.191 - 7.746	453.62	-3.386	-9.968 - 2.592	50.39

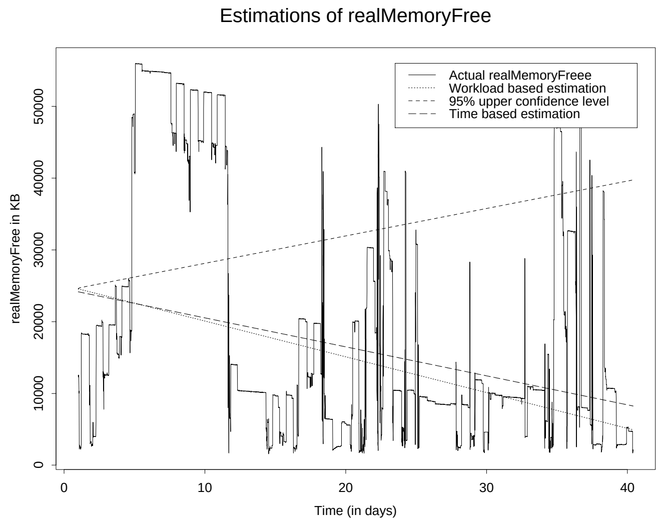

Figure 7. Time plot and trend estimations of resource *realMemoryFree* in machine Rossby

results take us a step further towards predicting actual failure times and may help us in proposing new or better preventive maintenance policies like "software rejuvenation".

A possible extension of this work could be to consider the system workload in a more fine-grained manner and to determine the relation between each individual process or a group of similar processes and resource consumption.

ACKNOWLEDGMENT

The authors would like to thank Dr. Katerina Goseva-Popstojanova, Dept. of Electrical & Computer Engineering, Duke University, for her useful comments and suggestions.

References

[1] R. Chillarege, S. Biyani, and J. Rosenthal. Measurement of Failure Rate in Widely Distributed Software. In *Proc. of 25th IEEE Intnl. Symposium on Fault Tolerant Computing*, pages 424–433, Pasadena, CA, July 1995.

[2] G. Ciardo, R. Marie, B. Sericola, K. S. Trivedi. Performability Analysis Using Semi-Markov Reward Processes. *IEEE Transactions on Computers*, 39(10):1251-1264, October 1990.

[3] M. V. Devarakonda, and R. K. Iyer. Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. *IEEE Transactions on Computers*, 15(12):1579-1586, December 1989.

[4] D. Ferrari, G. Serazzi, A. Zeigner. *Measurement and Tuning of Computer Systems*. Prentice Hall, Englewood Cliffs, NJ, 1983.

[5] S. Garg, A. van Moorsel, K. Vaidyanathan, K. Trivedi. A Methodology for Detection and Estimation of Software Aging. In *Proc. of 9th Intnl. Symposium on Software Reliability Engineering*, pages 282-292, Paderborn, Germany, November 1998.

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

[7] J. Gray and D. P. Siewiorek. High-availability Computer Systems. *IEEE Computer*, pages 39–48, September 1991.

[8] J. A. Hartigan. *Clustering Algorithms*. New York:Wiley, 1975.

[9] Y. Huang, P. Jalote, and C. Kintala. *Lecture Notes in Computer Science, Vol. 774*, chapter Two techniques for transient software error recovery, pages 159–170. Springer Verlag, Berlin, 1994.

[10] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton. Software Rejuvenation: Analysis, Module and Applications. In *Proc. of 25th Symposium on Fault Tolerant Computer Systems*, pages 381–390, Pasadena, California, June 1995.

[11] M. C. Hsueh, R. K. Iyer, and K. S. Trivedi. Performability Modeling Based on Real Data: A Case Study. *IEEE Transactions on Computers*, 37(4):478-484, April 1988

[12] R.K. Iyer and D.J. Rossetti. Effect of System Workload on Operating System Reliability: A Study on IBM 3081. *IEEE Transactions on Software Engineering*, SE-11(12):1438–1448, Dec. 1985.

[13] R. K. Iyer, L. T. Young, and P. V. K. Iyer. Automatic Recognition of Intermittent Failures: An Experimental Study of Field Data. *IEEE Transactions on Computers*, 39(4):525–537, April 1990.

[14] P. Jalote, Y. Huang, and C. Kintala. A Framework for Understanding and Handling Transient Software Failures. In *Proc.*

2nd ISSAT Intnl. Conf. on Reliability and Quality in Design, Orlando, Florida, 1995.

[15] I. Lee, R. K. Iyer, and A. Mehta. Identifying Software Problems using Symptoms. In *Proc. of 24th IEEE Intnl. Symposium on Fault Tolerant Computing*, Toulouse, France, June 1994.

[16] R. A. Maxion and F. E. Feather. A Case Study of Ethernet Anomalies in a Distributed Computing Environment. *IEEE Transactions on Reliability*, 39(4), 1990.

[17] V. Nicola, A. Bobbio, K. Trivedi. A Unified Performance Reliability Analysis of a System With a Cumulative Down Time Constraint. *Microelectron. Reliab.*, Vol. 32, No. 1/2, pp49-65, 1992.

[18] S. M. Ross. *Stochastic Processes*. John Wiley & Sons, New York, 1983.

[19] R. A. Sahner, K. S. Trivedi, A. Puliafito. *Performance and Reliability Analysis of Computer Systems - An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, Norwell, MA, 1996.

[20] P. K. Sen. Estimates of the Regression Coefficient Based on Kendall's Tau. *Journal of the American Statistical Association*, 63:1379–1389, 1968.

[21] R. M. Smith, K. S. Trivedi. The Analysis of Computer Systems using Markov Reward Processes. In *Stochastic Analysis of Computer and Communication Systems*, Ed. H.Takagi, Elsevier Science Publishers, 1989.

[22] M. Sullivan and R. Chillarege. Software Defects and Their Impact on System Availability - A Study of Field Failures in Operating Systems. In *Proc. 21st IEEE Intnl. Symposium on Fault-Tolerant Computing*, pages 2–9, 1991.

[23] D. Tang and R. K. Iyer. Dependability Measurement Modeling of a Multicomputer System. *IEEE Transactions on Computers*, 42(1), January 1993.

[24] A. Thakur and R. K. Iyer. Analyze-NOW – An Environment for Collection and Analysis of Failures in a Network of Workstations. In *Intnl. Symposium on Software Reliability Engineering*, pages 14–23, White Plains, NY, April 1996.