Ensuring the performance of Apache HTTP server affected by aging

Jing Zhao, Kishor S. Trivedi, Michael Grottke, Javier Alonso, YanBin Wang

Abstract—Failures due to software aging are typically caused by resource exhaustion, which is often preceded by progressive software performance degradation. Response time as a customer-affected metric can thus be used to detect the onset of software aging. In this paper, we propose the distribution-based rejuvenation algorithm (DBRA), which uses a validated M/E/1/K queuing model of the Apache HTTP server to decide when to trigger rejuvenation. We compare the performance of the DBRA with the one of the static rejuvenation algorithm with averaging (SRAA) presented by Avritzer et al. Simulation results show the effectiveness of the DBRA and its advantages over the SRAA in reducing the average response time. However, the DBRA generally tends to trigger rejuvenation more frequently than the SRAA, which increases the request blocking probability.

Index Terms—queueing model, response time distribution, distribution-based rejuvenation algorithm, static rejuvenation algorithm with averaging, software aging detection

1 INTRODUCTION

It is well known that system outages are more due to software faults than due to hardware faults [1]. Software faults have been classified into three types according to their potential manifestation characteristics: Bohrbugs, non-aging-related Mandelbugs, and aging-related bugs [2]. Software aging is the phenomenon of progressive performance degradation of the running software, which may lead to system crashes or undesirable hangs [3]. It can happen due to the exhaustion of system resources, such as memory leaks, unreleased locks, non-terminated threads, shared-memory pool latching, storage fragmentation, and the like [4]. This undesired phenomenon exists not only in commercial software such as Web and application servers, but also in critical applications requiring high reliability/availability. Software aging could also cause great losses in safety-critical systems [5], including the loss of human lives [6]. It does not make software fail immediately once started, but instead it typically leads to the accumulation of internal error conditions, which is often accompanied by progressive performance degradation of the software until it, finally, hangs or crashes. To counteract software aging, Huang et al. [4] proposed a proactive approach called software rejuvenation. It involves occasionally stopping the software, cleaning its internal state and restarting it in order to release system resources, so that the software performance is recovered. Thus, software rejuvenation mends the system before it fails. It has been implemented successfully in various systems, such as billing data collection systems [4], telecommunication systems [7], transaction processing systems [8] and spacecraft systems [9].

The software aging behavior can be captured by one or more indicators [10]. Such aging indicators are measurable metrics of the software system likely to be influenced by software aging. Software aging and performance degradation can be gauged by monitoring the consumed system resources at application and system levels. Measurable metrics of system/application resources are amount of memory free/used, swap space free/used, number of threads in use, etc., while response time (RT) is a key measure of the performance at the user/application level. The variation of RT can be used to infer the evolving process of software aging. From the client perspective, a gradually increasing RT may be an evidence of software aging causing the performance degradation of the server. The RT values obtained by continuous monitoring can thus be used to detect the need for rejuvenation so as to counteract the effect of software performance degradation.

All three algorithms are based on the sample averages calculated from the frequently-monitored RT values; this averaging of successive observations smooths some of the short-term deviations of the RT metric. In the first one, called the static rejuvenation algorithm with averaging (SRAA), the observed RT values are averaged over a fixed sample size. The second one, referred to as the sampling acceleration rejuvenation algorithm with averaging (SARAA), reduces the sample size when a degradation in performance has been detected. Both algorithms employ a bucket metaphor: When the average RT is above (below) a certain threshold a ball is added to (or removed from) the current bucket. If this leads to an overflow (underflow) of the current bucket, we move to the next (previous) bucket. This process is repeated until the last bucket overflows, indicating that the RT metric has degraded sufficiently to warrant the execution of software rejuvenation. In addition to the sample size used for calculating RT averages, the number of buckets and the bucket depth are control parameters of these algorithms. Compared with these algorithms, which use a rather small sample size, the third algorithm, called the central-limit-theorem-based algorithm (CLTA), uses a large sample size to warrant the approximation of the distribution of average RT with the normal distribution, following from the central limit theorem. Avritzer et al. [12] evaluated the performance of the SRAA, the SARAA and the CLTA in simulations, adjusting the control parameters.

The new distribution-based rejuvenation algorithm (DBRA) presented in this paper also makes use of averages of the measured RT values. It employs the bucket metaphor like the SRAA and the SARAA. Similar to the CLTA, it considers the distribution of the average RT; however, it makes use of the exact distribution rather than the asymptotic normal distribution: A ball is added to (removed from) the current bucket when the average RT calculated exceeds (is below) the target quantile of the distribution of average RT. Once all buckets are full, rejuvenation is triggered. To obtain the target quantile, we develop an analytic model of the distribution of average RT.

The Apache HTTP server [13] is the most popular Web server used on the Internet [14], and it is known to suffer from software aging under certain circumstances and configurations [15]. Hence, we select it for our study. While the Apache HTTP server is multi-threaded, a request is handled by its own thread or process throughout the life cycle of the request, a limit is put on the number of processes simultaneously allowed in the server, and some parameters need to be customized [15], [16], [17]. The effect of aging in the Apache HTTP server may result in gradual performance degradation, a gradually decreasing service rate or even unavailability [1]. In practice, these factors need to be considered to build a queuing model of the Apache HTTP server. Based on an accurate analytical model validated by experimental measurements the distribution of average RT can be obtained, which may facilitate a decision on when to trigger rejuvenation using the above-mentioned DBRA.

In summary, our main contributions are as follows:

- We propose and validate the M/E₂/1/K queuing system for modeling the Apache HTTP server architectural behavior. Furthermore, we derive closed-form analytical expressions for the steady-state probabilities of the model, which to the best of our knowledge have not been available in the literature.
- We calculate the RT distribution, and numerically obtain quantiles of the distribution of average RT in order to decide when to rejuvenate the Apache HTTP server.
- We propose the distribution-based rejuvenation algorithm (DBRA) to ensure the performance of the Apache HTTP server even under the effects of software aging. It uses the quantile derived from the exact distribution of average RT for a given confidence level to determine the presence of aging. The simulation results show that the algorithm presented offers advantages over the SRAA proposed by Avritzer et al. [12].

This paper is an extension of a previous conference paper [18]. We make the following new contributions: First, we numerically obtain the quantile of the exact average RT distribution for a given degree of confidence using the SHARPE tool [19]. Second, we propose the DBRA employing this quantile to detect aging and control rejuvenation. Third, we develop a simulation program to validate the DBRA, and to compare the effectiveness of the DBRA and the SRAA under different control parameters.

The rest of the paper is organized as follows: Section 2 presents the queueing model of the Apache HTTP server as a continuous-time Markov chain (CTMC). In Section 3, we first obtain the RT distribution of the M/E₂/1/K model by the tagged-job approach, and then derive its mean and variance; this information is used to validate the CTMC model with experimental results. We then describe a numerical approach to compute the exact distribution of average RT, and we also calculate its mean and variance. In Section 4, the distribution-based rejuvenation algorithm (DBRA) is proposed, and the performance of the DBRA and the SRAA is evaluated by simulation. The effectiveness of these two algorithms is compared by adjusting the control parameters. Finally, Section 5 contains concluding remarks and the discussion of future research.

## 2 Modeling the Apache HTTP Server

To build an analytical model of the Apache HTTP server, we conduct a two-step process. First, we consider its working mechanism as well as some of its configuration parameters. Based on this information, we then propose a queuing model of the Apache HTTP server and derive the steady-state probabilities of the underlying CTMC.
2.1 Apache HTTP server

The Apache HTTP server is structured as a pool of workers (either threads or processes, depending on the specific software release), as shown in Fig. 1.

Requests enter the server at the accept queue, where they wait until a worker is available, i.e., in “idle” state. When a worker starts to process a request, it switches into “busy” state. It then remains busy until the current request has been processed and the response has been sent back to the end-user. The widely-used HTTP 1.1 protocol provides persistent connections. The MaxClients parameter limits the size of the worker pool, thereby imposing a restriction on the processing rate of the server. A parent process is responsible for launching child processes to handle requests, and for adjusting the child processes by killing or spawning them to meet the workload. Besides the capacity of the server, the extent to which it is subjected to the aging phenomenon also depends its configuration. If the MaxRequestsPerChild is set to zero no child processes will ever be killed, speeding up the accumulation of internal error conditions due to aging-related bugs.

Grottke at al. [15] reported the results of a 14-day-period experiment executed to estimate the service rate offered by the Apache HTTP server. With MaxClients set to 250 and MaxRequestsPerChild set to zero, the request rate was varied from 350 requests per second and 390 requests per second, at increments of ten requests per second. Each request rate phase took around three days. The authors concluded that under the settings aforementioned the capacity of the Web server amounted to about 390 requests per second.

2.2 Queuing model of the Apache HTTP server

Making use of the above information on the Apache HTTP server, we build a queuing model for a system that is not subject to aging. The results obtained from this model will serve as a baseline to the SRAA and the DBRA for deciding when to trigger rejuvenation.

We model the service time by a random variable following an r-stage Erlang distribution. The arriving requests are queued, the service discipline is FCFS, and the total number of requests in the system is limited to K; this implies an M/E_r/1/K queuing model [20, p. 537]. In this paper, we specifically assume a 2-stage Erlang distribution as the service time distribution. Thus, the queuing model is M/E_2/1/K; it can be represented by a CTMC as shown in Fig. 2. The state denotes the number of exponential phases to be completed at service rate 2µ. With pi representing the steady-state probability of state i in the CTMC, we write the following system of steady-state balance equations for the M/E_2/1/K model:

\[-λp_0 + 2µp_1 = 0,\]
\[-(λ + 2µ)p_1 + 2µp_2 = 0,\]
\[-(λ + 2µ)p_i + 2µp_{i+1} + λp_{i-2} = 0, \quad 2 ≤ i ≤ 2K - 2,\]
\[-2µp_{2K-1} + 2µp_{2K} + λp_{2K-3} = 0,\]
\[-2µp_{2K} + λp_{2K-2} = 0,\]
\[\sum_{i=0}^{2K} p_i = 1.\]

To derive quantities of interest for our model, we employ the similarities with its unlimited-buffer-space version, i.e., with the M/E_2/1 queuing model. The steady-state balance equations of this latter model can be written as [21, p. 134]

\[-λπ_0 + 2µπ_1 = 0,\]
\[-(λ + 2µ)π_1 + 2µπ_2 = 0,\]
\[-(λ + 2µ)π_i + 2µπ_{i+1} + λπ_{i-2} = 0, \quad i ≥ 2,\]

where π_i denotes the steady-state probability for i phases in the system. Although the two models are not perfectly equivalent, these steady-state probabilities π_i of M/E_2/1 solve the M_2/M/1 constant bulk-input model with service rate 2µ, in which two phases are brought in by each batch arrival [21, p. 134]. The probability π_0 is given by

\[π_0 = 1 - ρ,\]

with ρ ≡ λ/µ, like for all G/G/1 models [21, p. 12]. To derive the steady-state probabilities of the general M^{[X]}/M/1 bulk-input model, where the number of phases per batch is a discrete random variable X, Gross et al. [21, pp. 117-119] use a generating function approach:

\[P(z) = \sum_{i=0}^{∞} π_iz^i, \quad |z| ≤ 1,\]
\[C(z) = \sum_{i=1}^{∞} c_iz^i, \quad |z| ≤ 1,\]
with \( c_i \) denoting the probability that \( X \) equals \( i \). For an \( M^{[X]}/M/1 \) bulk-input model with service rate \( 2\mu \), their result [21, pp. 119] becomes

\[
P(z) = \frac{2\mu \pi_0(1 - z)}{2\mu(1 - z) - \lambda z(1 - C(z))}, \quad |z| \leq 1. \tag{2}
\]

More specifically, for our \( M^{[2]}/M/1 \) model, where each batch arrival consists of exactly two requests, \( c_2 = 1 \) and \( c_i = 0 \) for \( i \neq 2 \); thus, \( C(z) \) is merely equal to \( z^2 \). We therefore obtain the following expression for \( P(z) \) from Equation (2):

\[
P(z) = \frac{2\mu \pi_0(1 - z)}{2\mu(1 - z) - \lambda z(1 + z^2)}, \quad |z| \leq 1,
\]

which can be simplified to

\[
P(z) = \frac{2}{\rho} \cdot \frac{1 - \rho}{z^2 + z - 2/\rho}, \quad |z| \leq 1. \tag{3}
\]

Partial fraction expansion of Equation (3) yields

\[
P(z) = \frac{2}{\rho} \cdot \frac{1 - \rho}{\sqrt{1 + 8/\rho}} \cdot \left( \frac{b_0}{1 - b_0 \cdot z} - \frac{b_1}{1 - b_1 \cdot z} \right),
\]

where

\[
b_0 = \frac{2}{-1 + \sqrt{1 + 8/\rho}} \quad \text{and} \quad b_1 = \frac{2}{-1 - \sqrt{1 + 8/\rho}}.
\]

For \( 0 < \rho < 1 \), both \( |b_0| \) and \( |b_1| \) are less than 1, and we can use infinite geometric sums to write Equation (4) as

\[
P(z) = \frac{2}{\rho} \cdot \frac{1 - \rho}{\sqrt{1 + 8/\rho}} \cdot \sum_{i=0}^{\infty} \left( (b_0^{i+1} - b_1^{i+1}) \cdot z^i \right).
\]

Comparing this with Equation (1) reveals that the steady-state probability of \( i \) phases in the \( M/E_2/1 \) service system is given by

\[
\pi_i = \frac{2}{\rho} \cdot \frac{1 - \rho}{\sqrt{1 + 8/\rho}} \cdot (b_0^{i+1} - b_1^{i+1}).
\]

We obtain the steady-state probabilities of the CTMC related to the \( M/E_2/1/K \) queuing model by normalizing the steady-state probabilities \( \pi_i \):

\[
p_i = \begin{cases} 
\frac{1}{G} \pi_i & \text{if } 0 \leq i \leq 2K - 1, \\
\frac{1}{G} \pi_{2K-2} & \text{if } i = 2K.
\end{cases}
\tag{4}
\]

Summing both sides of Equation (4), we derive

\[
2K - 1 \sum_{i=0}^{2K-1} p_i + p_{2K} = \frac{1}{G} \sum_{i=0}^{2K-1} \pi_i + \frac{\rho}{2G} \pi_{2K-2} = 1. \tag{5}
\]

From Equation (5), the normalizing constant \( G \) is obtained as

\[
G = \frac{2}{\rho} \cdot \frac{1 - \rho}{\sqrt{1 + 8/\rho}} \times \left( \frac{b_0^{2K+1} - b_1^{2K+1}}{1 - b_0} - \frac{b_1^{2K+1} - b_1^{2K+1}}{1 - b_1} + \frac{\rho}{2} (b_0^{2K-1} - b_1^{2K-1}) \right).
\]

With

\[
A = \frac{b_0 - b_0^{2K+1}}{1 - b_0} - \frac{b_1 - b_1^{2K+1}}{1 - b_1} + \frac{\rho}{2} (b_0^{2K-1} - b_1^{2K-1}),
\]

Equation (5) can thus be written as

\[
p_i = \begin{cases} 
\frac{b_0^{i+1} - b_1^{i+1}}{A} & \text{if } 0 \leq i \leq 2K - 1, \\
\frac{\rho}{2A} (b_0^{2K-1} - b_1^{2K-1}) & \text{if } i = 2K.
\end{cases}
\]

An incoming request is accepted unless the queue is full, implying \( 2K \) or \( 2K - 1 \) phases in the system. The probability of acceptance in the \( M/E_2/1/K \) model, \( p_a \), is therefore given by

\[
p_a = 1 - p_{2K} - p_{2K-1}.
\]

### 3 Response time distribution

In this section, we first derive the cumulative distribution function (CDF), the mean and the variance of the random variable \( R \), denoting the RT of an accepted request in the \( M/E_2/1/K \) model. Secondly, we describe an approach to computing the CDF (and its quantiles) of the sample average \( \bar{R}_n \), calculated from \( n \) independently sampled RT values, in addition to its mean and variance.

#### 3.1 CDF, mean and variance of response time

We use the tagged-job approach (see Trivedi [20, p. 418]) to calculate the RT distribution of an accepted request in the \( M/E_2/1/K \) model. The probability that an arriving request sees \( i \leq 2K - 2 \) phases in the system, conditional that it is accepted, is given by \( p_i/p_a \). For such a request, the RT is the sum of the completion times of \( i+2 \) phases. As these completion times are independent and are each exponentially distributed with rate \( 2\mu \), the RT follows an \((i+2)\)-stage Erlang distribution with rate \( 2\mu \). Therefore, the CDF of the response time \( R \) of an accepted request can be written as

\[
F_R(t) = \sum_{i=0}^{2K-2} \frac{p_i}{p_a} \cdot \left( 1 - \sum_{j=0}^{i+1} \frac{(2\mu)^j}{j!} e^{-2\mu} \right).
\tag{6}
\]

Likewise, the probability density function of \( R \), \( f_R(t) \), is a weighted average of Erlang probability density functions, and its Laplace transform is the following weighted average of the Laplace transforms of Erlang probability density functions:

\[
f_R^*(s) = \sum_{i=0}^{2K-2} \frac{p_i}{p_a} \cdot \left( \frac{2\mu}{s+2\mu} \right)^i e^{-2\mu}.
\tag{7}
\]

From Equation (7) we easily derive \( E(R) \), the expected value of \( R \):

\[
E(R) = -\frac{dF_R^*(s)}{ds} \bigg|_{s=0} = \sum_{i=0}^{2K-2} \frac{p_i}{p_a} \cdot \frac{i+2}{2\mu}.
\]
Moreover, we obtain $Var(R)$, the variance of $R$, as

$$Var(R) = E(R^2) - (E(R))^2 = \frac{d^2 f_R(s)}{ds^2} \bigg|_{s=0} - (E(R))^2$$

$$= \sum_{i=0}^{K-2} p_i \frac{(i+2)(i+3)}{4\mu^2} - \left( \sum_{i=0}^{K-2} \frac{p_i}{p_a} \cdot \frac{i+2}{2\mu} \right)^2.$$

We use the expected RT expression for validating the $M/E_2/1/K$ model proposed in Section 2.2. Table 1 also lists the expected RT values for the $M/M/1/250$ and the $M/E_2/1/K$ models, with $r$ equaling 2, 3, 5, and 10. We observed that the $M/E_2/1/K$ model achieves a good approximation of the actual measurements. Under reasonable workloads, it obtains the best fit. However, as the request arrival rate approaches the saturation point (i.e., 390 requests per second) the model fits less accurately.

### 3.2 CDF, mean and variance of average response time

Woolet [22] presented a computation technique for RT distributions that are phase-type, i.e., corresponding to the absorption time distribution in a CTMC. This idea has been adapted for calculating the distribution of the average RT [12], [23], and it is this approach that we employ in the current paper.

To this end, we require a CTMC whose absorption time represents the RT in the $M/E_2/1/K$ queuing model discussed here. Indeed, such a CTMC, similar to the one for the $M/M/c/K$ model, [23], can be found; it is depicted in Fig. 3, with

$$q_i \equiv \frac{p_i}{p_i + p_{i+1} + \ldots + p_{2K-2}} = \frac{q_i^{i+1}}{1 + \mu r} = \frac{k_i^{i+1} - k_i^{i+1} - k_i^{i+2}}{1 + \mu r}.$$ 

Starting in the initial state $T_1$ at time $t = 0$, the transient probability of having reached the absorbing state $R$ by time $t$, $\pi_R(t)$, is equivalent to the CDF of the RT in the $M/E_2/1/K$ model.

#### Differential equations for this CTMC yield

$$\frac{d\pi_{T_1}(t)}{dt} = -2\mu \cdot \pi_{T_1}(t),$$

$$\frac{d\pi_{T_2}(t)}{dt} = -2\mu \cdot \pi_{T_2}(t) + 2\mu \cdot \pi_{T_1}(t),$$

$$\frac{d\pi_{T_i}(t)}{dt} = -2\mu \cdot \pi_{T_i}(t) + 2\mu \cdot (1 - q_{i-3}) \cdot \pi_{T_{i-1}}(t),$$

for $3 \leq i \leq 2K$.

$\pi_{T_1}(0) = 1$ and $\pi_{T_2}(0) = \pi_R(0) = 0$ for $2 \leq i \leq 2K$. Using the Laplace transform, we derive

$$\pi_R^R(s) = \frac{(2\mu)^{i-1}(1-q_0)(1-q_1)\ldots(1-q_{i-3})}{(s+2\mu)^i},$$

and

$$s\pi_R^R(s) = \sum_{i=0}^{2K-2} q_i \cdot \frac{(2\mu)^{i+2}}{p_i + p_{i+1} + \ldots + p_{2K-2}}.$$ 

$$= \frac{2K-2}{p_a} \cdot \frac{(2\mu)^{i+2}}{p_a}.$$ 

Thus, the Laplace transform of the derivative of $\pi_R(t)$ is identical to $f_R^*(s)$, given in Equation (7), confirming that the RT distribution has indeed been equivalently represented.

Let $\overline{R}_n = \frac{1}{n} \sum_{m=1}^{n} R_m = \sum_{m=1}^{n} R_m/n$ denote the average of $n$ independent and identically distributed random variables $R_m$, where each $R_m$ has the CDF of the RT in the $M/E_2/1/K$ queuing system shown in Equation (6). To find the distribution of this sample average $\overline{R}_n$, following the approach in [12] and [23], we make use of the well-known fact that multiplying an exponentially distributed random variable with rate $z$ by some constant $r$ yields an exponential random variable with rate $z/r$ [20, pp. 151–152]. Since all transition times in a Markov chain follow an exponential distribution, multiplying all transition rates in Fig. 3 by $n$ results in a Markov chain whose time to absorption has the same distribution as $R/n$. The random variable $\overline{R}_n$ is the sum of the independent and identically distributed terms $R_1/n, R_2/n, \ldots, R_n/n$. Therefore, we can represent the distribution of $\overline{R}_n$ by concatenating $n$ such Markov chains, fusing the initial state $T_1$ of the $m^{th}$ chain and

### Table 1

Average RT measured and expected RT for $M/M/1/250$ and $M/E_2/1/250$ when $r$ equals 2, 3, 4, 5, and 10

<table>
<thead>
<tr>
<th>$\lambda$ (reqs/s)</th>
<th>$\mu$ (reqs/s)</th>
<th>Average RT measured (ms)</th>
<th>$E(R)$ for $M/M/1/250$ (ms)</th>
<th>$E(R)$ for $M/E_2/1/250$ (ms)</th>
<th>$E(R)$ for $M/MaxClients/1/250$ (ms)</th>
<th>$E(R)$ for $M/E_2/1/250$ (ms)</th>
<th>$E(R)$ for $M/E_4/1/250$ (ms)</th>
<th>$E(R)$ for $M/E_6/1/250$ (ms)</th>
<th>$E(R)$ for $M/E_8/1/250$ (ms)</th>
<th>$E(R)$ for $M/E_{10}/1/250$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>390</td>
<td>20.24</td>
<td>25.00</td>
<td>19.39</td>
<td>17.52</td>
<td>16.58</td>
<td>16.02</td>
<td>14.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>360</td>
<td>390</td>
<td>29.04</td>
<td>33.34</td>
<td>25.64</td>
<td>23.08</td>
<td>21.81</td>
<td>21.02</td>
<td>19.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>370</td>
<td>390</td>
<td>38.85</td>
<td>50.01</td>
<td>38.14</td>
<td>34.02</td>
<td>32.22</td>
<td>31.03</td>
<td>28.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>380</td>
<td>390</td>
<td>103.29</td>
<td>99.01</td>
<td>75.53</td>
<td>70.73</td>
<td>65.88</td>
<td>62.83</td>
<td>56.14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
the absorbing state $R$ of the $(m-1)^{st}$ chain into one state, say $F_{m-1}$, $m = 2, \ldots , n$, as shown in Fig. 4. Using SHARPE, we can numerically calculate the absorption time distribution in the resulting Markov chain, which is equivalent to the CDF $F_{\overline{R}_n}(t)$ of the sample average $\overline{R}_n$. As an example, Fig. 5 depicts the results obtained for the averages of RT values from the M/E$_2$/1/250 queuing model with $\lambda = 350$ and $\mu = 390$ when the sample size $n$ equals 1, 5, 10, and 15, respectively.

Our DBRA, presented in Section 4.3, employs $F_{\overline{R}_n}^{-1}(p)$, the 100$p\%$ quantile of the distribution of $\overline{R}_n$ with confidence level $p \in (0, 1)$, to control rejuvenation. In contrast, the SRAA proposed by Avritzer et al. [12] makes use of the first two moments of the sample average. It is well known that the expected value of $\overline{R}_n$ is identical to the expected value of $R$:

$$E(\overline{R}_n) = \frac{1}{n} \sum_{m=1}^{n} E(R_m) = E(R) = \sum_{i=0}^{2K-2} \frac{p_i}{p_a} i + \frac{2}{2\mu}.$$

Similarly, the variance of $\overline{R}_n$ is related to the variance of $R$:

$$Var(\overline{R}_n) = \frac{1}{n^2} \sum_{m=1}^{n} Var(R_m) = \frac{Var(R)}{n} = \sum_{i=0}^{2K-2} \frac{p_i}{p_a} \frac{(i + 2)(i + 3)}{4\mu^2 n} - \left( \sum_{i=0}^{2K-2} \frac{p_i}{p_a} \frac{i + 2}{2\mu \sqrt{n}} \right)^2,$$

and it can thus be calculated using our previous results.

**4 Rejuvenation Algorithms and Their Simulation Results**

In this section, we briefly review the SRAA due to Avritzer et al. [12] and propose the new DBRA. Moreover, we present simulation results for the performance of an Apache HTTP server in which rejuvenation is
controlled by these two algorithms. Our simulation program implements the M/E2/1/K queuing model of the Apache HTTP server. The aging phenomenon is simulated by reducing the service rate (measured in requests per second) whenever a request is being served; each decrease is calculated by multiplying the time interval (measured in seconds) since the end of the most recent rejuvenation (or since the time simulation started if rejuvenation has not yet been triggered) with the constant 1e-9. As simulation progresses, the performance degradation thus tends to accelerate. We decided on this approach to simulate aging based on the results of experiments we conducted with the Apache HTTP server, adjusting the constant to guarantee a high probability for at least two failure events during the simulation run for a system without rejuvenation. A failure event occurs when the service rate is less than 50% of the initial maximum service rate. The duration of the reactive recovery after a failure has been set to four seconds. During a failure, the incoming requests are discarded (i.e., blocked). The already accepted requests waiting to be served are also discarded (i.e., dropped). The duration of rejuvenation has been set to two seconds. During rejuvenation the incoming requests are discarded (i.e., blocked), while those requests that have already been accepted and that are waiting to be processed are preserved. We run each simulation for a simulation time of 100,000 seconds. To obtain more accurate results by reducing the simulation error, the behavior under each scenario is simulated 15 times, and the average of the results obtained is computed.

Besides studying a Web server employing the SRAA or the DBRA to decide when to rejuvenate, we also simulate the system behavior without any rejuvenation. These no-rejuvenation results serve as a baseline against which to judge the effects of the two algorithms.

Under each scenario, the performance of the Web server is evaluated according to three metrics: average RT, request rejection probability, and request blocking probability. The request rejection probability is estimated by the fraction of requests rejected by the system during simulation because the maximum system capacity has been reached. Similarly, we estimate the request blocking probability by the fraction of simulated requests that are discarded due to rejuvenation; naturally, this probability is zero in the no-rejuvenation cases.

In the following, average RT as well as the request rejection and blocking probabilities are shown as functions of the offered load \( \rho \). We simulate results for the offered loads 350/390, 360/390, 370/390, and 380/390.

### 4.1 SRAA

Slightly adapting the notation, we list the pseudo-code for the SRAA proposed by Avritzer et al. [12] as Algorithm 1. It can be seen that this algorithm places a ball into the current bucket \( b \) if the average calculated from the last \( n \) RT values exceeds the expected value of \( \bar{R}_n \) by \( b - 1 \) standard deviations of \( \bar{R}_n \), where \( \bar{R}_n \) is the sample average of the response time for an M/E2/1/K queuing system that is not subject to aging. If performance degradation due to aging is experienced, the SRAA therefore tends to fill the buckets with balls. Once the \( B^{th} \) bucket overflows, rejuvenation is triggered.

The parameters of the SRAA consist of the sample size \( n \), the number of buckets \( B \) and the bucket depth \( D \), i.e., the number of balls that fit into each bucket. Note that the moments of the sample average \( \bar{R}_n \) exactly apply to the average of observed RT values only if these values have been independently sampled from the RT distribution of an M/E2/1/K queuing system. However, at high loads the RT values of subsequent requests can be substantially correlated.

#### Algorithm 1 Static rejuvenation algorithm with averaging

1: function SRAA(\( n, B, D \))
2: \( u \leftarrow 0 \)
3: \( b \leftarrow 0 \)
4: \( d \leftarrow 0 \)
5: while \( n \) additional observations available do
6: \( u \leftarrow u + 1 \)
7: \( \bar{R}_n \leftarrow \frac{1}{n} \sum_{m=n}^{n+1} R_m \)
8: if \( \bar{R}_n > E(\bar{R}_n) + (b - 1) \cdot \sqrt{Var(\bar{R}_n)} \) then
9: \( d \leftarrow d + 1 \)
10: else
11: \( d \leftarrow d - 1 \)
12: end if
13: if \( d > D \) then
14: \( d \leftarrow 0 \)
15: \( b \leftarrow b + 1 \)
16: end if
17: if \( d < 0 \) and \( b > 1 \) then
18: \( d \leftarrow 0 \)
19: \( b \leftarrow b - 1 \)
20: end if
21: if \( d < 0 \) and \( b == 1 \) then
22: \( d \leftarrow 0 \)
23: end if
24: if \( b > B \) then
25: rejuvenation_route()
26: end if
27: end while
28: end function

### 4.2 SRAA results

We now present the simulation results obtained when employing the SRAA under different settings. We use the no-rejuvenation system simulation results as a baseline. Fig. 6(a) shows the average RT results without rejuvenation as well as using the SRAA with the following parameter values: \((n, B, D) = (1, 10, 30), (5, 10, 30), (10, 10, 30), (15, 10, 30)\). Fig. 6(b) and
Fig. 6. Average RT, request blocking and rejection probability without rejuvenation and with SRAA

Table 2: Results of SRAA simulations

<table>
<thead>
<tr>
<th>Load</th>
<th>Average RT (ms)</th>
<th>Rejection probability</th>
<th>Blocking probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAA(1,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>26.36</td>
<td>0</td>
<td>0.0041</td>
</tr>
<tr>
<td>360/390</td>
<td>37.03</td>
<td>0</td>
<td>0.0054</td>
</tr>
<tr>
<td>370/390</td>
<td>63.43</td>
<td>1.87·10^{-7}</td>
<td>0.0091</td>
</tr>
<tr>
<td>380/390</td>
<td>270.15</td>
<td>0.0066</td>
<td>0.0034</td>
</tr>
<tr>
<td>SRAA(5,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>26.27</td>
<td>0</td>
<td>0.0034</td>
</tr>
<tr>
<td>360/390</td>
<td>33.23</td>
<td>0</td>
<td>0.0046</td>
</tr>
<tr>
<td>370/390</td>
<td>48.51</td>
<td>5.40·10^{-8}</td>
<td>0.0075</td>
</tr>
<tr>
<td>380/390</td>
<td>106.51</td>
<td>1.51·10^{-5}</td>
<td>0.0141</td>
</tr>
<tr>
<td>SRAA(10,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>28.71</td>
<td>0.0091</td>
<td>0.0029</td>
</tr>
<tr>
<td>360/390</td>
<td>36.43</td>
<td>9.91·10^{-8}</td>
<td>0.0035</td>
</tr>
<tr>
<td>370/390</td>
<td>51.03</td>
<td>1.13·10^{-6}</td>
<td>0.0048</td>
</tr>
<tr>
<td>380/390</td>
<td>101.28</td>
<td>1.77·10^{-5}</td>
<td>0.0122</td>
</tr>
<tr>
<td>SRAA(15,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>30.12</td>
<td>2.09·10^{-8}</td>
<td>0.0029</td>
</tr>
<tr>
<td>360/390</td>
<td>38.49</td>
<td>9.25·10^{-8}</td>
<td>0.0035</td>
</tr>
<tr>
<td>370/390</td>
<td>54.71</td>
<td>1.13·10^{-6}</td>
<td>0.0048</td>
</tr>
<tr>
<td>380/390</td>
<td>102.31</td>
<td>2.25·10^{-5}</td>
<td>0.0099</td>
</tr>
</tbody>
</table>

(replacement text: rejuvenation less frequently, leading to a smaller request blocking probability. On the other hand, increasing the sample size used by the SRAA from 5 to 10 and from 10 to 15 worsens the average RT attained by the system, except in the high-load case ρ = 380/390. For this high-load case, an interesting behavior of the SRAA is observed: With n equal one, the SRAA causes a larger average RT, a higher request rejection probability and a significantly lower request blocking probability than for the other sample sizes. This is due to the high autocorrelation in the sequence of RT values at this load. Calculating averages from subsequently observed RT values therefore tends to smooth short-term fluctuation less than assumed in the derivation of the moments of the sample average R̄n. The SRAA thus triggers rejuvenations more frequently, leading to a higher request rejection probability. For n = 1, where this effect does not play any role, the SRAA rejuvenates the system less often, which allows the system queue to fill up, implying both a higher request rejection probability and a larger average RT of those requests that do get served.

To study the influence of the number of buckets B on the SRAA behavior, we run a set of simulations where the sample size n and the bucket depth D were fixed, while the number of buckets varied. Fig. 7 summarizes the results. As expected, a lower number of buckets means a higher frequency with which rejuvenation is triggered. Hence, the request blocking probability increases as the number of buckets decreases.
We develop the distribution-based rejuvenation algorithm (DBRA) as shown in Algorithm 2. It is very similar to the SRAA, but it uses the $100p\%$ quantile of the distribution of the sample average $\overline{R}_n$ to control software rejuvenation, instead of the first two moments of this distribution. In our simulation using the DBRA, we set the confidence degree $p$ equal to 95%, 97.5%, and 99%. As before, it should be noted that a high correlation between subsequent RT values may cause the distribution of their average to deviate substantially from the distribution of $\overline{R}_n$.

### 4.4 DBRA results

In this section, we analyze the results obtained when using the DBRA to control the rejuvenation cycles, employing the settings for $n$, $B$ and $D$ already studied during our SRAA simulations.

Table 3 lists the simulation results obtained for the DBRA with confidence level parameter $p$ equal to 95%. Fig. 8 summarizes the average RT, request rejection probability and request blocking probability attained.

The RT results in Fig. 8(a) indicate that the DBRA is able to reduce the expected RT as compared with the no-rejuvenation case. Similar to the SRAA, increasing the sample size usually means a higher average RT, because rejuvenation is triggered less often as it takes longer to collect a larger sample. Fig. 8(b) shows that the rejection probability in the no-rejuvenation case is greatly larger than that in any of the DBRA scenarios, although it is non-zero in many of these scenarios. As expected, Fig. 8(c) confirms that the request blocking probability gets lower as the sample size increases.

The number of buckets has a small effect on the average RT offered by the system when the DBRA is used to trigger rejuvenation, as can be seen from Fig. 9(a). However, Fig. 9(b) shows there is a clear impact on the request blocking probability: As the number of buckets increases, the blocking probability decreases, because rejuvenation tends to be triggered less frequently.

Furthermore, we also study the effects of the confidence level $p$ on the average RT as well as on the request blocking probability. For the parameters $n = 15, B = 10, D = 30$, Fig. 10(a) depicts the average RT when $p$ is set to 95%, 97.5%, and 99%, while Fig. 10(b) shows the related request blocking probabilities. It is observed that the DBRA guarantees a similar RT under all confidence levels chosen, whereas a lower blocking probability is

```plaintext
Algorithm 2 Distribution-based rejuvenation algorithm
1: function DBRA($n, B, D, p$)
2:   $u \leftarrow 0$
3:   $b \leftarrow 0$
4:   $d \leftarrow 0$
5:   while $n$ additional observations available do
6:       $u \leftarrow u + 1$
7:       $\overline{R}_n \leftarrow \frac{1}{u} \sum_{m=(u-1)n+1}^{un} R_m$
8:       if $\overline{R}_n > F_{\overline{R}_n}^{-1}(p)$ then
9:           $d \leftarrow d + 1$
10:      else
11:          $d \leftarrow d - 1$
12:       end if
13:       if $d > D$ then
14:           $d \leftarrow 0$
15:           $b \leftarrow b + 1$
16:       end if
17:       if $d < 0$ and $b > 1$ then
18:           $d \leftarrow 0$
19:           $b \leftarrow b - 1$
20:       end if
21:       if $d < 0$ and $b == 1$ then
22:           $d \leftarrow 0$
23:       end if
24:       if $b > B$ then
25:           rejuvenation_route()
26:       end if
27:   end while
28: end function
```
TABLE 3
Results of DBRA (95%) simulations

<table>
<thead>
<tr>
<th>Load</th>
<th>Average RT (ms)</th>
<th>Rejection probability</th>
<th>Blocking probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBRA(1,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>21.96</td>
<td>0</td>
<td>0.0216</td>
</tr>
<tr>
<td>360/390</td>
<td>30.73</td>
<td>0</td>
<td>0.0379</td>
</tr>
<tr>
<td>370/390</td>
<td>50.03</td>
<td>0</td>
<td>0.0611</td>
</tr>
<tr>
<td>380/390</td>
<td>115.95</td>
<td>6.19·10⁻⁶</td>
<td>0.1001</td>
</tr>
<tr>
<td>DBRA(5,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>24.22</td>
<td>0</td>
<td>0.0039</td>
</tr>
<tr>
<td>360/390</td>
<td>30.16</td>
<td>0</td>
<td>0.0061</td>
</tr>
<tr>
<td>370/390</td>
<td>43.33</td>
<td>0</td>
<td>0.0148</td>
</tr>
<tr>
<td>380/390</td>
<td>90.35</td>
<td>3.8·10⁻⁷</td>
<td>0.0402</td>
</tr>
<tr>
<td>DBRA(10,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>26.28</td>
<td>0</td>
<td>0.0033</td>
</tr>
<tr>
<td>360/390</td>
<td>34.67</td>
<td>0</td>
<td>0.0040</td>
</tr>
<tr>
<td>370/390</td>
<td>46.67</td>
<td>0</td>
<td>0.0071</td>
</tr>
<tr>
<td>380/390</td>
<td>89.15</td>
<td>0.0207</td>
<td></td>
</tr>
<tr>
<td>DBRA(15,10,30)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>350/390</td>
<td>27.14</td>
<td>0</td>
<td>0.0031</td>
</tr>
<tr>
<td>360/390</td>
<td>35.37</td>
<td>9.81·10⁻⁴</td>
<td>0.0038</td>
</tr>
<tr>
<td>370/390</td>
<td>50.16</td>
<td>3.69·10⁻⁷</td>
<td>0.0055</td>
</tr>
<tr>
<td>380/390</td>
<td>92.59</td>
<td>1.38·10⁻⁵</td>
<td>0.0140</td>
</tr>
</tbody>
</table>

achieved when \( p = 99\% \).

4.5 Comparing the DBRA with the SRAA

Finally, we compare both algorithms under the same parameter settings to evaluate their performance. Fig. 11 summarizes the performance of the SRAA and the DBRA. For clarity, we only show two cases for each algorithm. We observe that the system using the DBRA offers better average RT results in both cases studied. Since the SRAA is more conservative, triggering rejuvenation less frequently than the DBRA, the request rejection probability under the SRAA is larger. However, the blocking probability of the DBRA is higher than the one attained by the SRAA. This is especially critical when a small sample size \( n \) is used. As the sample size increases, the blocking probabilities under the DBRA and the SRAA become similar.

5 CONCLUSION

In this paper, we proposed to describe the behavior of the Apache HTTP server using an M/E₂/1/K queuing model. After deriving closed-form expressions for the steady-state probabilities as well as the related response time distribution and its moments, we validated this model by actual data measured during experiments. We then obtained the cumulative distribution function, the mean and the variance of the response time sample average \( R_n \). The SHARPE tool was used to numerically calculate the response time distribution and its quantiles. Based on these results, we introduced the distribution-based rejuvenation algorithm (DBRA) to control rejuvenation and used simulation to compared it with the static rejuvenation algorithm with averaging (SRAA) presented by Avritzer et al. We also studied the influence of the sample size and the number of buckets on the average response time and the blocking probability.
Finally, we analyzed the performance of the DBRA using different values of the confidence level parameter.

The results evidenced that the performance of the DBRA is advantageous over the SRAA to maintain a shorter average response time offered by the system. However, the DBRA achieves this by triggering rejuvenation more frequently, which causes the request blocking probability to be larger than for a system employing the SRAA.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under Grant No. 60873036, the National Research Foundation for the Doctoral Program of Higher Education of China No.20070217051, the fundamental research funds for the Central Universities (award number HEUCF100601, HEUCFT1007). This work was also supported in part by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) under a JPL subcontract # 1440119, as well as by the Dr. Theo and Friedl Schoeller Research Center for Business and Society.

REFERENCES

Fig. 11. Average RT, request rejection and blocking probability with SRAA and with DBRA


Jing Zhao
Jing Zhao received Ph.D. (2006) degree in computer science and Technology in Harbin institute of Technology of China. In 2010 he was with Department of Electrical and Computer Engineering at Duke University, Durham, NC, working as a Postdoc under supervision of Dr. Kishor Trivedi. She is currently a Professor in the School of Computer Science at Harbin Engineering University of China. Her research interests include reliability engineering, software aging theory, and dependability modeling.

Kishor S. Trivedi
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, Queueing and Computer Science Applications, published by Prentice-Hall; a thoroughly revised second edition (including its Indian edition) of this book has been published by John Wiley. He has also published two other books entitled, Performance and Reliability Analysis of Computer Systems, published by Kluwer Academic Publishers and Queueing Networks and Markov Chains, John Wiley. He is a Fellow of the Institute of Electrical and Electronics Engineers. He is a Golden Core Member of IEEE Computer Society. He has published over 490 articles and has supervised 44 Ph.D. dissertations. He is the recipient of IEEE Computer Society Technical Achievement Award for his research on Software Aging and Rejuvenation. His research interests in are in reliability, availability, performance and survivability of computer and communication systems and in software dependability. He works closely with industry in carrying our reliability/availability analysis, providing short courses on reliability, availability, and in the development and dissemination of software packages such as SHARPE, SREPT and SPNP.
Michael Grottke received an MA in economics from Wayne State University, USA, and a Diploma degree in business administration as well as a Ph.D. from the Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany. From 2004 to 2007, he was a Research Associate and Assistant Research Professor in the Department of Electrical and Computer Engineering at Duke University, USA. In 2010, he received the Habilitation degree from the Friedrich-Alexander-Universität Erlangen-Nürnberg. His research interests include intermediate fields between statistics, computer science, and business administration, such as software aging and rejuvenation, software performance, software engineering economics, and stochastic modeling. He has published papers on these topics at international conferences as well as in international journals, including IEEE Computer, IEEE Transactions on Reliability, Journal of Systems and Software, and Performance Evaluation. He is a member of the IEEE and the German Statistical Society.

Javier Alonso received the masters degree in Computer Science in 2004 and the Ph.D. degree from the Technical University of Catalonia (Universitat Politècnica de Catalunya, UPC) in 2011. From 2006 to 2011, he held an assistant lecturer position in the Computer Architecture Department of UPC. Since 2011, he has a Postdoctoral Associate position under the supervision of Professor K.S. Trivedi, in the Electrical and Computer Engineering Department, Duke University, Durham, NC. Dr. Alonso has published more than 20 papers about different aspects of dependability, availability, reliability and software aging in premier conferences and journals. He has also served as a reviewer for IEEE Transactions on Computers, IEEE Transactions on Dependability and Security Computing, Performance Evaluation, and Cluster Computing, and several international conferences. His research interests are in dependability, reliability, availability, and performance of computer and communication systems. He has special interest in software dependability and software aging and rejuvenation topics. He is or has been involved in JPL/NASA, NEC, NATO, Huawei, WiPro funded projects.

YanBin Wang received Ph.D. (2006) degree in Industrial Engineering department in Harbin institute of Technology of China. In 2010 he was with Department of Electrical and Computer Engineering at Duke University, Durham, NC, working as a research associate. He is currently an associate Professor in Industrial Engineering department in Harbin institute of Technology of China. His research interests include quality management, scheduling and optimization.