Ensuring the Performance of Apache HTTP Server Affected by Aging

Jing Zhao, Kishor S. Trivedi, *Fellow*, *IEEE*,
Michael Grottke, *Member*, *IEEE*, Javier Alonso, and 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-affecting 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 $\mathrm{M/E_2/1/K}$ 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—Queuing 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, nonaging-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, nonterminated 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 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, and so on, 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.

Avritzer and Weyuker [11] witnessed the aging phenomenon in telecommunications software, where the service rate of the software decreases with time, increasing the queue lengths and eventually causing the loss of packets. Avritzer et al. [12] built an M/M/c queuing model and proposed a set of three online algorithms which are able to distinguish between system performance degradations caused by software aging and those that are due to bursts in the arrival process. 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

- *J. Zhao is with the School of Computer Science and Technology, Harbin Engineering University, 133 Room, 21 Building, No. 145 Nantong Street, Nangang District, Harbin 150001, China. E-mail: jingzhao.duke@gmail.com.*
- *K.S. Trivedi and J. Alonso are with the Department of Electrical and Computer Engineering, Duke University, 206 Hudson Hall, Durham, NC 27708-0291.*
- *M. Grottke is with the Department of Statistics and Econometrics, Friedrich-Alexander-Universität Erlangen-Nürnberg, Lange Gasse 20, Nürnberg, Bayern 90403, Germany.*
- *Y. Wang is with the Industrial Engineering Department, Harbin Institute of Technology of China, Room 224, Yi-Fu Building, No. 92 XiDaZhi Street, Nangang District, Harbin 150001, China. E-mail: wangyb@hit.edu.cn.*

Manuscript received 13 June 2012; revised 28 Aug. 2013; accepted 2 Sept. 2013; published online 6 Sept. 2013.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2012-06-0123.
Digital Object Identifier no. 10.1109/TDSC.2013.38.

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 multithreaded, 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 $M/E_2/1/K$ 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.

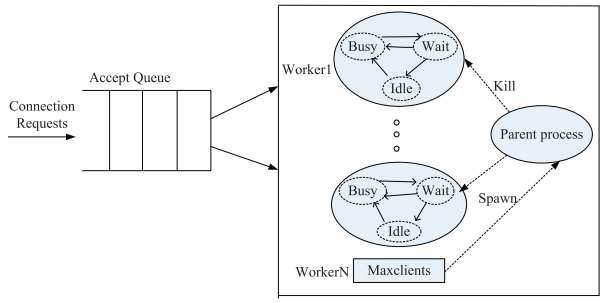


Fig. 1. Apache architecture.

- We calculate the RT distribution, and numerically obtain quantiles of the distribution of average RT to decide when to rejuvenate the Apache HTTP server.
- We propose the 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 queuing model of the Apache HTTP server as a continuous-time Markov chain (CTMC). In Section 3, we first obtain the RT distribution of the $M/E_2/1/K$ 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 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

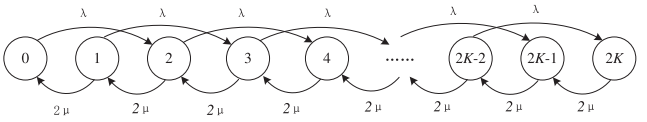


Fig. 2. CTMC of the $\mathrm{M/E_2/1/K}$ queuing model.

"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 on 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 et 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 10 requests per second. Each request rate phase took around three days. The authors concluded that under the aforementioned settings, 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 $\mathrm{M/E}_r\mathrm{/1/K}$ queuing model [20, p. 537]. In this paper, we specifically assume a two-stage Erlang distribution as the service time distribution. Thus, the queuing model is $\mathrm{M/E_2/1/K}$; 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 p_i representing the steady-state probability of state i in the CTMC, we write the following system of steady-state balance equations for the $\mathrm{M/E_2/1/K}$ model:

$$\begin{aligned}
-\lambda p_0 + 2\mu p_1 &= 0,\\
-(\lambda + 2\mu)p_1 + 2\mu p_2 &= 0,\\
-(\lambda + 2\mu)p_i + 2\mu p_{i+1} + \lambda p_{i-2} &= 0, \quad 2 \le i \le 2K-2,\\
-2\mu p_{2K-1} + 2\mu p_{2K} + \lambda p_{2K-3} &= 0,\\
-2\mu p_{2K} + \lambda p_{2K-2} &= 0,\\
\sum_{i=0}^{2K} p_i &= 1.
\end{aligned}$$

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

$$\begin{aligned}
-\lambda\pi_0 + 2\mu\pi_1 &= 0,\\
-(\lambda + 2\mu)\pi_1 + 2\mu\pi_2 &= 0,\\
-(\lambda + 2\mu)\pi_i + 2\mu\pi_{i+1} + \lambda\pi_{i-2} &= 0, \quad i \ge 2,
\end{aligned}$$

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 $\mathrm{M/E_2/1}$ solve the $\mathrm{M^{[2]}/M/1}$ 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

$$\pi_0 = 1 - \rho,$$

with $\rho \equiv \lambda/\mu$, like for all $\mathrm{G/G/1}$ models [21, p. 12]. To derive the steady-state probabilities of the general $\mathrm{M^{[X]}/M/1}$ 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:

$$\begin{aligned}
P(z) &= \sum_{i=0}^{\infty} \pi_i z^i, \quad |z| \le 1,\\
C(z) &= \sum_{i=1}^{\infty} c_i z^i, \quad |z| \le 1,
\end{aligned} \tag{1}$$

with c_i denoting the probability that X equals i. For an $\mathrm{M^{[X]}/M/1}$ bulk-input model with service rate 2μ, their result [21, p. 119] becomes

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

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

$$P(z) = \frac{2\mu\pi_0(1-z)}{2\mu(1-z) - \lambda z(1 - z^2)}, \quad |z| \le 1,$$

which can be simplified to

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

Partial fraction expansion of (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 (4) as

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

TABLE 1
Average RT Measured and Expected RT for M/M/1/250 and M/E$_r$/1/250 When r Equals 2, 3, 4, 5, and 10

λ (reqs/s)	μ (reqs/s)	Average RT measured (ms)	$E(R)$ for M/M/1/250 (ms)	$E(R)$ for M/E$_2$/1/250 (ms)	$E(R)$ for M/E$_3$/1/250 (ms)	$E(R)$ for M/E$_4$/1/250 (ms)	$E(R)$ for M/E$_5$/1/250 (ms)	$E(R)$ for M/E$_{10}$/1/250 (ms)
350	390	20.24	25.00	19.39	17.52	16.58	16.02	14.90
360	390	29.04	33.34	25.64	23.08	21.81	21.02	19.48
370	390	38.85	50.01	38.14	34.02	32.22	31.03	28.65
380	390	103.29	99.01	75.53	70.73	65.88	62.83	56.14

Comparing this with (1) reveals that the steady-state probability of i phases in the M/E$_2$/1 service system is given by

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

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 π_i:

$$p_i = \begin{cases} \frac{1}{G}\pi_i, & \text{if } 0 \le i \le 2K-1, \\ \frac{\rho}{2G}\pi_{2K-2}, & \text{if } i = 2K. \end{cases} \quad (4)$$

Summing both sides of (4), we derive

$$\sum_{i=0}^{2K-1} p_i + p_{2K} = \frac{1}{G}\sum_{i=0}^{2K-1}\pi_i + \frac{\rho}{2G}\pi_{2K-2} = 1. \quad (5)$$

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

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

With

$$A = \frac{b_0 - b_0^{2K+1}}{1-b_0} - \frac{b_1 - b_1^{2K+1}}{1-b_1} + \frac{\rho}{2}\left(b_0^{2K-1} - b_1^{2K-1}\right),$$

(5) can thus be written as

$$p_i = \begin{cases} \frac{b_0^{i+1} - b_1^{i+1}}{A}, & \text{if } 0 \le i \le 2K-1, \\ \frac{\rho}{2A}\cdot\left(b_0^{2K-1} - b_1^{2K-1}\right), & \text{if } i = 2K. \end{cases}$$

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. Second, we describe an approach to computing the CDF (and its quantiles) of the sample average $\overline{R}_n$, 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 [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 $i \le 2K-2$ 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 2μ, the RT follows an $(i+2)$-stage Erlang distribution with rate 2μ. Therefore, the CDF of the response time R of an accepted request can be written as

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

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:

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

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

$$E(R) = -\left.\frac{\mathrm{d}f_R^*(s)}{\mathrm{d}s}\right|_{s=0} = \sum_{i=0}^{2K-2}\frac{p_i}{p_a}\cdot\frac{i+2}{2\mu}.$$

Moreover, we obtain $Var(R)$, the variance of R, as

$$Var(R) = E(R^2) - (E(R))^2 = \left.\frac{\mathrm{d}^2 f_R^*(s)}{\mathrm{d}s^2}\right|_{s=0} - (E(R))^2 = \sum_{i=0}^{2K-2}\frac{p_i}{p_a}\cdot\frac{(i+2)(i+3)}{4\mu^2} - \left(\sum_{i=0}^{2K-2}\frac{p_i}{p_a}\cdot\frac{i+2}{2\mu}\right)^2.$$

We use the expected RT expression for validating the M/E$_2$/1/K model proposed in Section 2.2. Table 1 shows the average RT values measured during the experiments reported by Grottke et al. [15], which we already introduced in Section 2.1. As the `MaxClients` parameter was set to 250 during these experiments, we compare the actual averages with the expected RT for our M/E$_2$/1/250 queuing model. Table 1 also lists the expected RT values for the M/M/1/250 model and the M/E$_r$/1/250 models with r equaling 2, 3, 5, and 10. We obtained these latter values numerically using the SHARPE tool. The results indicate that the M/E$_2$/1/250 model achieves a good approximation of the actual measurements. Under reasonable workloads,

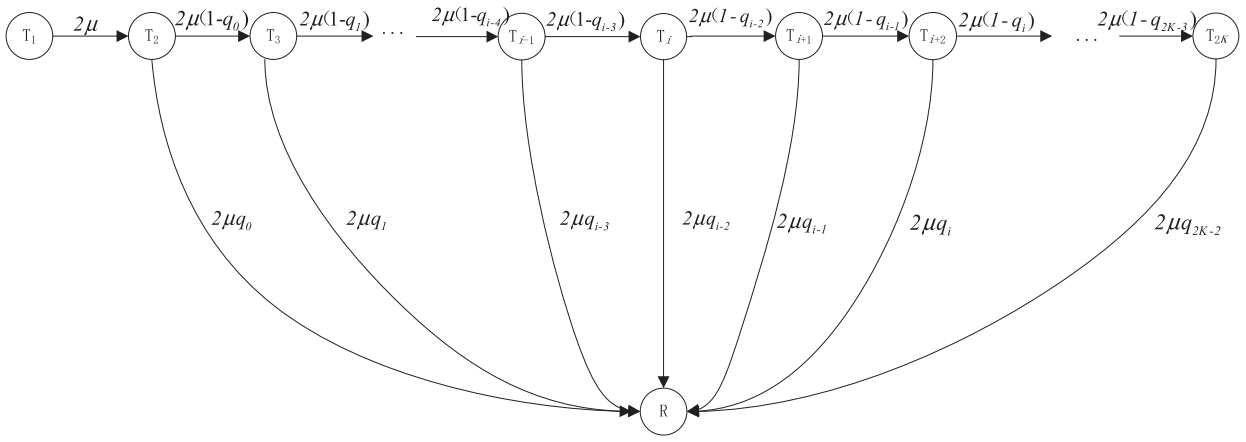


Fig. 3. CTMC whose absorption time represents R in the $\mathrm{M/E_2/1/K}$ model.

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 $\mathrm{M/E_2/1/K}$ 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

$$q_i \equiv \frac{p_i}{p_i + p_{i+1} + \cdots + p_{2K-2}} = \frac{b_0^{i+1} - b_1^{i+1}}{\frac{b_0^{i+1} - b_0^{2K}}{1-b_0} - \frac{b_1^{i+1} - b_1^{2K}}{1-b_1}}.$$

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 $\mathrm{M/E_2/1/K}$ model.

Differential equations for this CTMC yield

$$\begin{aligned}
\frac{\mathrm{d}\pi_{T_1}(t)}{\mathrm{d}t} &= -2\mu \cdot \pi_{T_1}(t),\\
\frac{\mathrm{d}\pi_{T_2}(t)}{\mathrm{d}t} &= -2\mu \cdot \pi_{T_2}(t) + 2\mu \cdot \pi_{T_1}(t),\\
\frac{\mathrm{d}\pi_{T_i}(t)}{\mathrm{d}t} &= -2\mu \cdot \pi_{T_i}(t) + 2\mu \cdot (1 - q_{i-3}) \cdot \pi_{T_{i-1}}(t),\\
&\quad 3 \le i \le 2K,\\
\frac{\mathrm{d}\pi_R(t)}{\mathrm{d}t} &= \sum_{i=2}^{2K} 2\mu \cdot q_{i-2} \cdot \pi_{T_i}(t),
\end{aligned}$$

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

$$\pi^*_{T_i}(s) = \frac{(2\mu)^{i-1}(1-q_0)(1-q_1)\cdot\ldots\cdot(1-q_{i-3})}{(s+2\mu)^i},$$

and

$$\begin{aligned}
s\pi^*_R(s) &= \sum_{i=0}^{2K-2} q_i \cdot \frac{(2\mu)^{i+2}}{(s+2\mu)^{i+2}} \cdot \frac{p_i + p_{i+1} + \cdots + p_{2K-2}}{p_a}\\
&= \sum_{i=0}^{2K-2} \frac{p_i}{p_a} \cdot \frac{(2\mu)^{i+2}}{(s+2\mu)^{i+2}}.
\end{aligned}$$

Thus, the Laplace transform of the derivative of $\pi_R(t)$ is identical to $f^*_R(s)$, given in (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 $\mathrm{M/E_2/1/K}$ queuing system shown in (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 mth chain and 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 $\mathrm{M/E_2/1/250}$ 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 $F^{-1}_{\overline{R}_n}(p)$, the $100p\%$ 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:

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

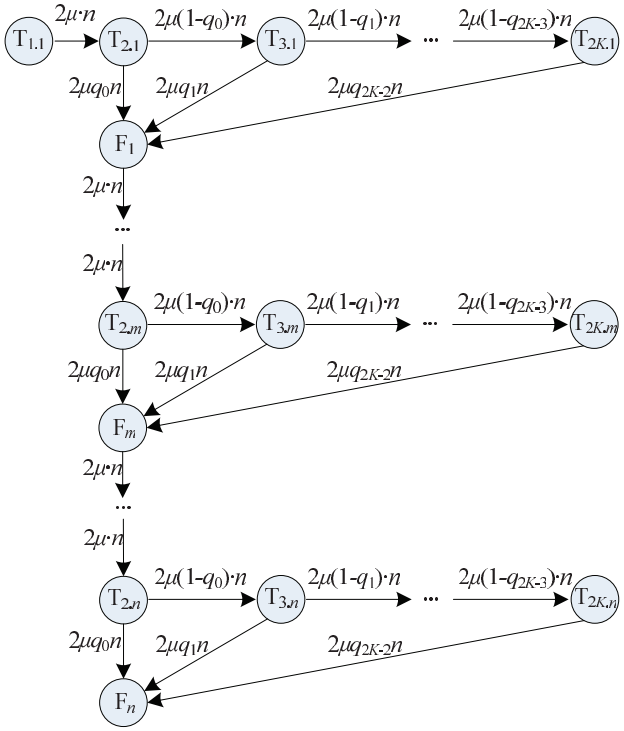


Fig. 4. CTMC whose absorption time represents $\overline{R}_n$.

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

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

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 $M/E_2/1/K$ 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 percent 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 2 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.

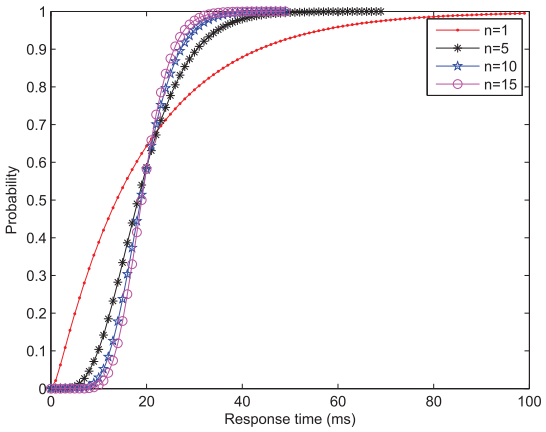


Fig. 5. CDF of $\overline{R}_n$ for different sample sizes n.

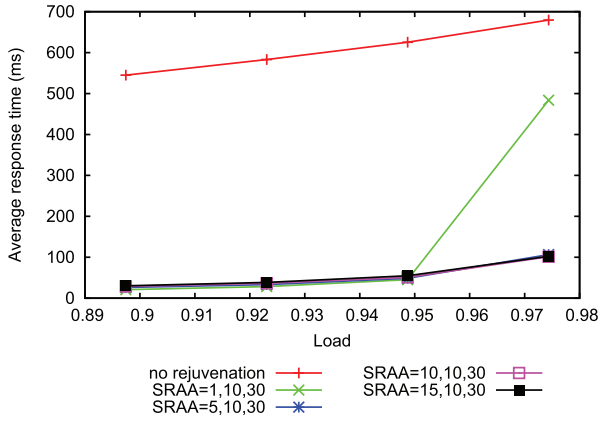
Besides studying a Web server employing the SRAA or the DBRA to decide when to rejuvenate, we also simulate the system behavior without any rejuvenation. These no-rejuvenation results serve as a baseline against which to judge the effects of the two algorithms.

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.

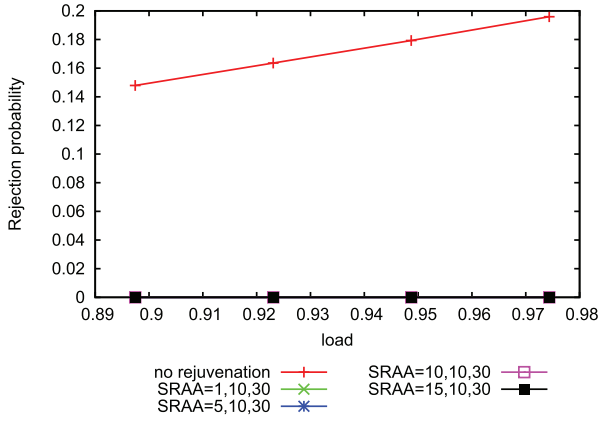
In the following, average RT as well as the request rejection and blocking probabilities are shown as functions of the offered load ρ. We simulate results for the offered loads 350/390, 360/390, 370/390, and 380/390.

4.1 SRAA

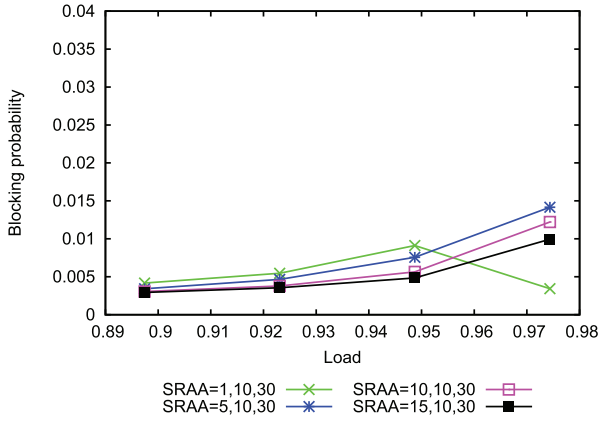
Slightly adapting the notation, we list the pseudocode for the SRAA proposed by Avritzer et al. [12] as Algorithm 1. It can be seen that this algorithm places a ball into the current bucket b if the average calculated from the last n RT values exceeds the expected value of $\overline{R}_n$ by $b-1$ standard deviations of $\overline{R}_n$, where $\overline{R}_n$ is the sample average of the response time for an $M/E_2/1/K$ 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 Bth bucket overflows, rejuvenation is triggered.



(a) Average RT without rejuvenation and with SRAA



(b) Request rejection probability without rejuvenation and with SRAA



(c) Request blocking probability with SRAA

Fig. 6. Average RT, request blocking, and rejection probability without rejuvenation and with SRAA.

Algorithm 1. Static rejuvenation algorithm with averaging.

function SRAA(n, B, D)
  $u \leftarrow 0$
  $b \leftarrow 0$
  $d \leftarrow 0$
  while n additional observations available **do**
    $u \leftarrow u + 1$
    $\overline{R}_n \leftarrow \frac{1}{n}\sum_{m=(u-1)n+1}^{un} R_m$
    if $\overline{R}_n > E(\overline{R}_n) + (b-1)\cdot\sqrt{Var(\overline{R}_n)}$ **then**
      $d \leftarrow d + 1$
    else
      $d \leftarrow d - 1$
    end if
    if $d > D$ **then**
      $d \leftarrow 0$
      $b \leftarrow b + 1$
    end if
    if $d < 0$ and $b > 1$ **then**
      $d \leftarrow 0$
      $b \leftarrow b - 1$
    end if
    if $d < 0$ and $b == 1$ **then**
      $d \leftarrow 0$
    end if
    if $b > B$ **then**
      rejuvenation_route()
    end if
  end while
end function

TABLE 2
Results of SRAA Simulations

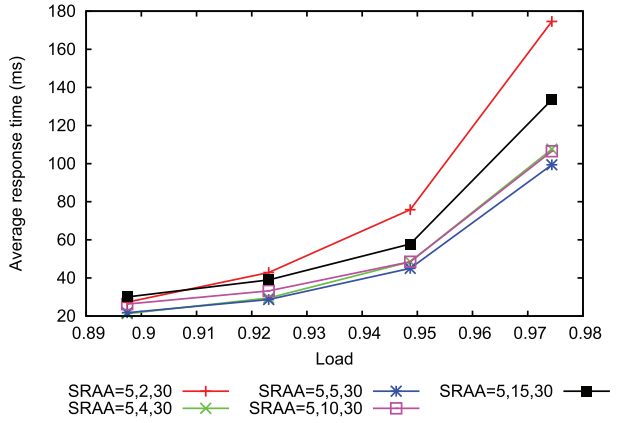
	Load	Average RT (ms)	Rejection probability	Blocking probability
SRAA(1,10,30)				
	350/390	26.36	0	0.0041
	360/390	37.03	0	0.0054
	370/390	63.43	$1.87\cdot10^{-7}$	0.0091
	380/390	270.15	0.0066	0.0034
SRAA(5,10,30)				
	350/390	26.27	0	0.0034
	360/390	33.23	0	0.0046
	370/390	48.51	$5.40\cdot10^{-8}$	0.0075
	380/390	106.51	$1.51\cdot10^{-5}$	0.0141
SRAA(10,10,30)				
	350/390	28.71	0	0.0030
	360/390	36.43	0	0.0037
	370/390	51.03	$9.91\cdot10^{-8}$	0.0056
	380/390	101.28	$1.77\cdot10^{-5}$	0.0122
SRAA(15,10,30)				
	350/390	30.12	$2.09\cdot10^{-8}$	0.0029
	360/390	38.49	$9.25\cdot10^{-8}$	0.0035
	370/390	54.71	$1.13\cdot10^{-6}$	0.0048
	380/390	102.31	$2.25\cdot10^{-5}$	0.0099

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.

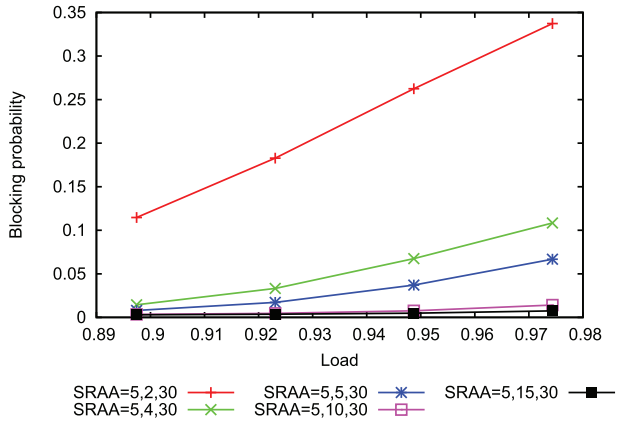
Note that the moments of the sample average $\overline{R}_n$ exactly apply to the average of observed RT values only if these values have been independently sampled from the RT distribution of an $M/E_2/1/K$ queuing system. However, at high loads, the RT values of subsequent requests can be substantially correlated.

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.



(a) Average RT with SRAA (n=5)



(b) Request blocking probability with SRAA (n=5)

Fig. 7. Average RT and request blocking probability with SRAA ($n = 5$).

Fig. 6a shows the average RT results without rejuvenation, as well as using the SRAA with the following parameter values: $(n, B, D) = (1, 10, 30), (5, 10, 30), (10, 10, 30), (15, 10, 30)$. Figs. 6b and 6c present the request rejection probability and request blocking probability, respectively. The exact simulation results for the SRAA are listed in Table 2.

It is clearly seen that the average RT of the system without rejuvenation is larger, due to the performance degradation caused by aging. The results also show that as the sample size n increases, the SRAA triggers 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 $\rho = 380/390$. 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 $\overline{R}_n$. 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.

TABLE 3
Results of DBRA (95 Percent) Simulations

	Load	Average RT (ms)	Rejection probability	Blocking probability
DBRA(1,10,30)				
	350/390	21.96	0	0.0216
	360/390	30.73	0	0.0379
	370/390	50.03	0	0.0611
	380/390	115.95	$6.19{\cdot}10^{-6}$	0.1001
DBRA(5,10,30)				
	350/390	24.22	0	0.0039
	360/390	30.16	0	0.0061
	370/390	43.33	0	0.0148
	380/390	90.35	$3.8{\cdot}10^{-7}$	0.0402
DBRA(10,10,30)				
	350/390	26.28	0	0.0033
	360/390	34.67	0	0.0040
	370/390	46.67	0	0.0071
	380/390	89.15	0	0.0207
DBRA(15,10,30)				
	350/390	27.14	0	0.0031
	360/390	35.37	$9.81{\cdot}10^{-8}$	0.0038
	370/390	50.16	$3.69{\cdot}10^{-7}$	0.0055
	380/390	92.59	$1.38{\cdot}10^{-5}$	0.0140

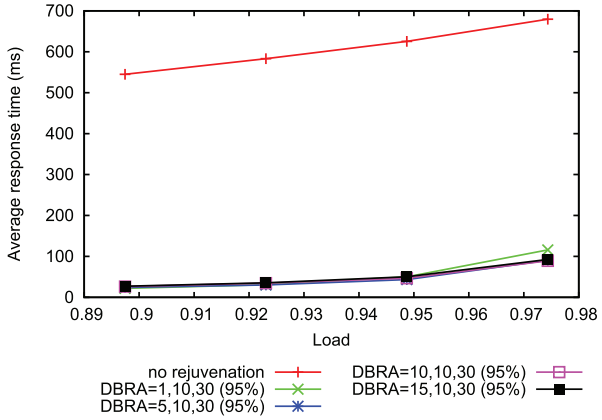
To study the influence of the number of buckets B on the SRAA behavior, we run a set of simulations where the sample size n and the bucket depth D were fixed, while the number of buckets varied. Fig. 7 summarizes the results. As expected, a lower number of buckets means a higher frequency with which rejuvenation is triggered. Hence, the request blocking probability increases as the number of buckets decreases.

4.3 DBRA

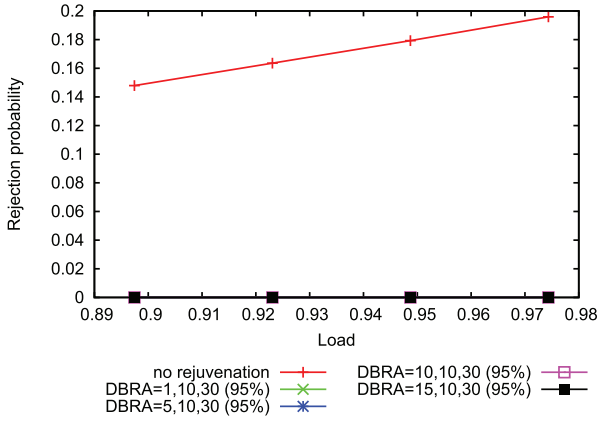
We develop the DBRA as shown in Algorithm 2. It is very similar to the SRAA, but it uses the $100p\%$ quantile of the distribution of the sample average $\overline{R}_n$ to control software rejuvenation, instead of the first two moments of this distribution. In our simulation using the DBRA, we set the confidence degree p equal to 95, 97.5, and 99 percent. 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 $\overline{R}_n$.

Algorithm 2. Distribution-based rejuvenation algorithm.

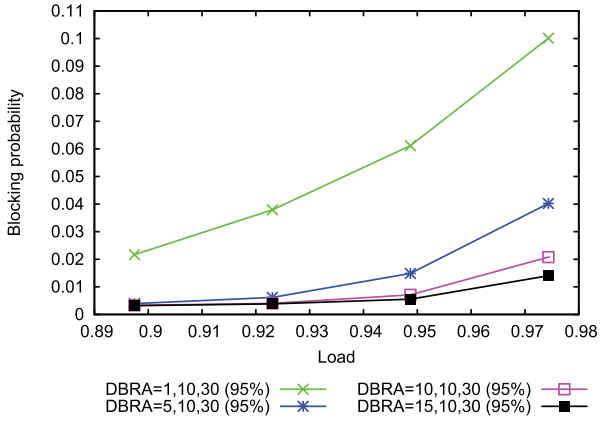
1: **function** DBRA(n, B, D, p)
2: $u \leftarrow 0$
3: $b \leftarrow 0$
4: $d \leftarrow 0$
5: **while** n additional observations available **do**
6: $u \leftarrow u + 1$
7: $\overline{R}_n \leftarrow \frac{1}{n}\sum_{m=(u-1)n+1}^{un} R_m$
8: **if** $\overline{R}_n > F^{-1}_{\overline{R}_n}(p)$ **then**
9: $d \leftarrow d + 1$
10: **else**



(a) Average RT without rejuvenation and with DBRA

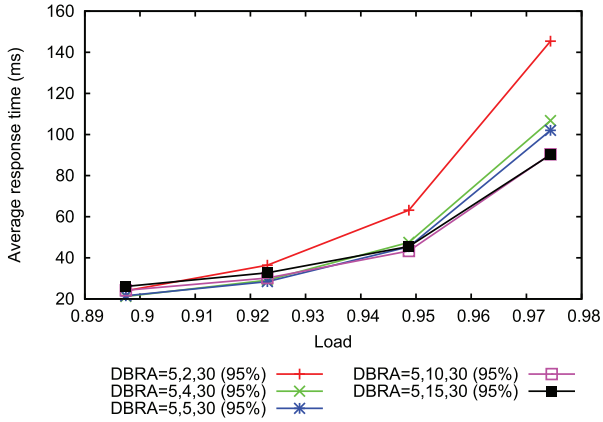


(b) Request rejection probability without rejuvenation and with DBRA

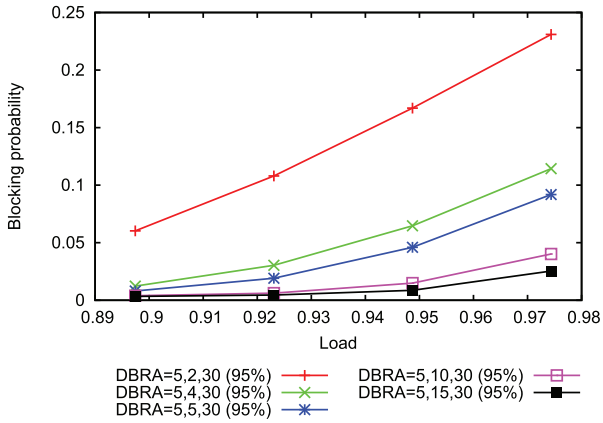


(c) Request blocking probability with DBRA

Fig. 8. Average RT, request rejection, and blocking probability without rejuvenation and with DBRA (CI 95 percent).



(a) Average RT with DBRA (n=5)



(b) Request blocking probability with DBRA (n=5)

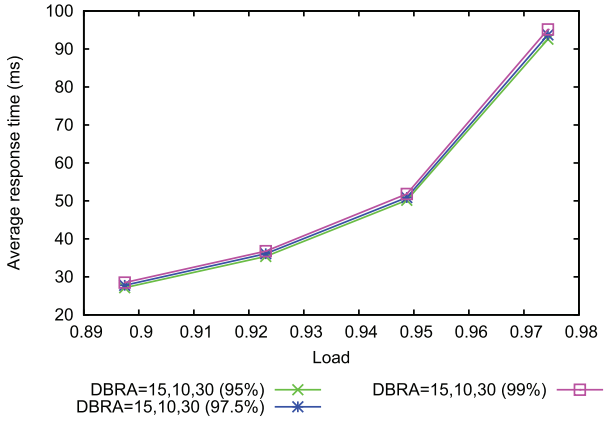
Fig. 9. Average RT and request blocking probability with DBRA ($n = 5$).

```
            end if
            if d < 0 and b == 1 then
                d ← 0
            end if
            if b > B then
                rejuvenation_route()
            end if
        end while
    end function
```

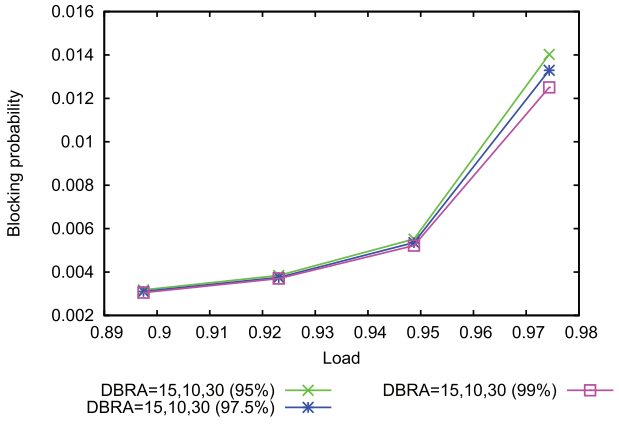
4.4 DBRA Results

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 percent. Fig. 8 summarizes the average RT, request rejection probability, and request blocking probability attained. The RT results in Fig. 8a 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. 8b shows that the rejection probability in the no-rejuvenation case is greatly larger than that in any of the DBRA scenarios,

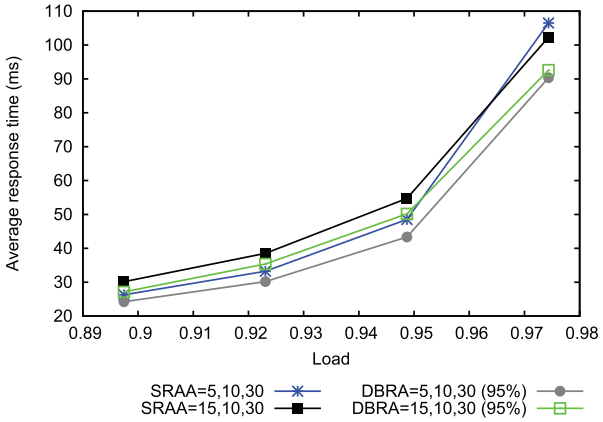


(a) Average RT with DBRA for different confidence levels

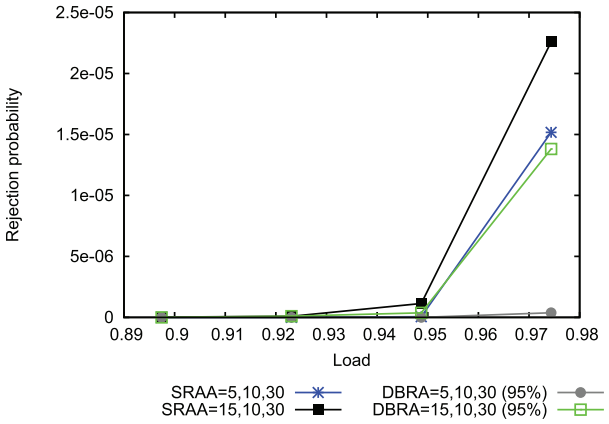


(b) Request blocking probability with DBRA for different confidence levels

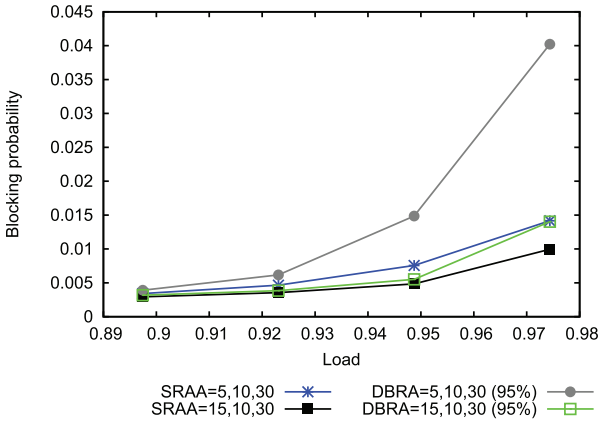
Fig. 10. Average RT and request blocking probability with DBRA for different confidence levels p.



(a) Average RT with SRAA and with DBRA



(b) Request rejection probability with SRAA and with DBRA



(c) Request blocking probability for SRAA and with DBRA

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

although it is nonzero in many of these scenarios. As expected, Fig. 8c 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. 9a. However, Fig. 9b shows that 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. 10a depicts the average RT when p is set to 95, 97.5, and 99 percent, while Fig. 10b 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 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 $M/E_2/1/K$ 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 $\overline{R}_n$. The SHARPE tool was used to numerically calculate the response time distribution and its quantiles. Based on these results, we introduced the DBRA to control rejuvenation and used simulation to compare it with the 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 was 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 numbers 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

[1] S. Garg, A. Puliafito, M. Telek, and K.S. Trivedi, "Analysis of Preventive Maintenance in Transactions Based Software Systems," *IEEE Trans. Computers,* vol. 47, no. 1, pp. 96-107, Jan. 1998.

[2] M. Grottke, A.P. Nikora, and K.S. Trivedi, "An Empirical Investigation of Fault Types in Space Mission System Software," *Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks,* pp. 447-456, 2010.

[3] S. Garg, A. van Moorsel, K. Vaidyanathan, and K.S. Trivedi, "A Methodology for Detection and Estimation of Software Aging," *Proc. Nineth Int'l Symp. Software Reliability Eng.,* pp. 283-292, 1998.

[4] Y. Huang, C. Kintala, N. Kolettis, and N.D. Fulton, "Software Rejuvenation: Analysis, Module and Applications," *Proc. 25th Symp. Fault-Tolerant Computing,* pp. 381-390, 1995.

[5] Y. Jia, L. Zhao, and K.-Y. Cai, "A Nonlinear Approach to Modeling of Software Aging in a Web Server," *Proc. 15th Asia-Pacific Software Eng. Conf.,* pp. 77-84, 2008.

[6] E. Marshall, "Fatal Error: How Patriot Overlooked a Scud," *Science,* vol. 255, no. 5050, article 1347, 1992.

[7] X.M. Zhang and H. Pham, "Predicting Operational Software Availability and Its Applications to Telecommunication Systems," *Int'l J. Systems Science,* vol. 33, no. 11, pp. 923-930, 2002.

[8] K.J. Cassidy, K.C. Gross, and A. Malekpour, "Advanced Pattern Recognition for Detection of Complex Software Aging in Online Transaction Processing Servers," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 478-482, 2002.

[9] K.-Y. Cai, "Software Reliability and Control," *J. Computer Science and Technology,* vol. 21, no. 5, pp. 697-707, 2006.

[10] M. Grottke, R. Matias Jr, and K.S. Trivedi, "The Fundamentals of Software Aging," *Proc. First Int'l Workshop Software Aging and Rejuvenation,* pp. 1-6, 2008.

[11] A. Avritzer and E.J. Weyuker, "Monitoring Smoothly Degrading Systems for Increased Dependability," *Empirical Software Eng.,* vol. 2, no. 1, pp. 59-77, 1997.

[12] A. Avritzer, A. Bondi, M. Grottke, K.S. Trivedi, and E.J. Weyuker, "Performance Assurance via Software Rejuvenation: Monitoring, Statistics and Algorithms," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 435-444, 2006.

[13] Apache Web Server, "Homepage," http://www.apache.org, 2013.

[14] Nectcraft, "August 2013 Web Server Survey," http://news.netcraft.com/archives/2013/08/09/august-2013-web-server-survey.html, Aug. 2013.

[15] M. Grottke, L. Li, K. Vaidyanathan, and K.S. Trivedi, "Analysis of Software Aging in a Web Server," *IEEE Trans. Reliability,* vol. 55, no. 3, pp. 411-420, Sept. 2006.

[16] Y.-F. Jia, Y.S. Jing, and K.-Y. Cai, "A Feedback Control Approach for Software Rejuvenation in a Web Server," *Proc. IEEE Int'l Conf. Software Reliability Eng./First Int'l Workshop Software Aging and Rejuvenation,* pp. 1-6, 2008.

[17] J. Cao, M. Andersson, C. Nyberg, and M. Kihl, "Web Server Performance Modeling Using an M/G/1/k*ps Queue," *Proc. 10th Int'l Conf. Telecomm.,* pp. 1501-1506, 2003.

[18] J. Zhao and K.S. Trivedi, "Performance Modeling of Apache Web Server Affected by Aging," *Proc. Third Int'l Workshop Software Aging and Rejuvenation,* pp. 56-61, 2011.

[19] K.S. Trivedi and R. Sahner, "SHARPE at the Age of Twenty Two," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 36, no. 4, pp. 52-57, 2009.

[20] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications,* second ed. Wiley, 2001.

[21] D. Gross, J.F. Shortle, J.M. Thompson, and C.M. Harris, *Fundamentals of Queueing Theory,* fourth ed. Wiley, 2008.

[22] S.P. Woolet, "Performance Analysis of Computer Networks," PhD dissertation, Dept. Electrical Eng., Duke Univ., 1993.

[23] M. Grottke, V. Apte, K.S. Trivedi, and S. Woolet, "Response Time Distributions in Networks of Queues," *Queueing Networks: A Fundamental Approach,* R. Boucherie and N. Van Dijk, eds., Springer, pp. 587-641, 2011.

Jing Zhao received the PhD degree in computer science and technology from the Harbin Institute of Technology of China in 2006. In 2010, she was with the Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, working as a postdoctoral researcher under supervision of Dr. Kishor Trivedi. She is currently a professor at the School of Computer Science, Harbin Engineering University of China. Her research interests include reliability engineering, software aging theory, and dependability modeling.

Kishor S. Trivedi holds the Hudson chair in the Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina. 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*, published by Prentice-Hall; a thoroughly revised second edition (including an Indian edition) of this book has been published by John Wiley. He has also published two other books, *Performance and Reliability Analysis of Computer Systems*, published by Kluwer Academic, and *Queueing Networks and Markov Chains*, published John Wiley. He has published more than 490 articles and has supervised 44 PhD dissertations. He is the recipient of the IEEE Computer Society Technical Achievement Award for his research on software aging and rejuvenation. His research interests include reliability, availability, performance, and survivability of computer and communication systems and software dependability. He works closely in the 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. He is a fellow of the IEEE and a Golden Core member of the IEEE Computer Society.

Michael Grottke received the MA degree in economics from Wayne State University, and the diploma degree in business administration and the PhD degree 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, Duke University. 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*, the *IEEE Transactions on Reliability*, the *Journal of Systems and Software*, and *Performance Evaluation*. He is a member of the IEEE and the German Statistical Society.

Javier Alonso received the master's degree in computer science in 2004 and the PhD degree from the Technical University of Catalonia (Universitat Politecnica de Catalunya, UPC) in 2011. From 2006 to 2011, he held an assistant lecturer position in the Computer Architecture Department, UPC. Since 2011, he has been a postdoctoral associate under the supervision of Professor K.S. Trivedi in the Electrical and Computer Engineering Department, Duke University, Durham, North Carolina. He 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 the *IEEE Transactions on Computers*, *IEEE Transactions on Dependability and Security Computing*, *Performance Evaluation*, and *Cluster Computing*, and several international conferences. His research interests include 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 projects funded by JPL/NASA, NEC, NATO, Huawei, and WiPro.

Yanbin Wang received the PhD degree from the Industrial Engineering Department, Harbin Institute of Technology of China, in 2006. In 2010, he was a research associate with the Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina. He is currently an associate professor in the Industrial Engineering Department, Harbin Institute of Technology of China. His research interests include quality management, scheduling, and optimization.