An Overhead and Resource Contention Aware Analytical Model for Overloaded Web Servers

Vipul Mathur*
vipulmathur@iitb.ac.in

Varsha Apte
varsha@cse.iitb.ac.in

Department of Computer Science and Engineering
Indian Institute of Technology - Bombay
Powai, Mumbai, Maharashtra 400076, India

ABSTRACT

Increased response time during periods of overload on a Web server may cause impatient users to time-out, causing the server to do unproductive work in processing these abandoned requests. Overhead time spent in preprocessing each request adds to the unproductive work even for requests that are not taken up for service. This causes the usable throughput, i.e. *goodput*, of the overloaded Web server to drop drastically, while resource utilization remains at 100%. Although this behaviour can be easily reproduced experimentally, existing analytical models of queues with abandonments are not able to do so.

We present an analytical model that captures characteristics specific to networked software servers, namely, overhead processing and contention for shared hardware resources, that is able to explain the goodput degradation typically observed in overloaded servers. We use this model to compare the performance of the LIFO and FIFO queueing disciplines during overload and show that LIFO goodput and response time are better than those of FIFO.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]—*Modeling techniques*; I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*; G.3 [**Probability and Statistics**]—*Queueing theory*

General Terms: Performance

Keywords: Web server, overload, analytical, model, LIFO, resource sharing, Markov chain, queueing theory

1. INTRODUCTION

A software server, such as a Web server, is said to be overloaded if the load offered to it exceeds its maximum capacity to serve requests. Unpredictable events such as breaking news or failure of some servers in a cluster can trigger unexpected levels of service demand. Server infrastructure, however, is usually designed for a maximum capacity that is lower than the peak demand, to avoid wasted capacity.

*Corresponding author

WOSP'07, February 5–8, 2007, Buenos Aires, Argentina.

The ideal behavior of an overloaded server is to keep serving requests at its maximum capacity, even when offered load increases beyond its capacity. However as offered load increases, response times increase, resulting in impatient users abandoning their requests. This leads to processing of requests for which the user timed-out before the reply was ready. Additionally there is overhead work such as TCP connection processing, parsing and preprocessing of the request by the Web server software, which must be done even for requests that eventually time-out while waiting to be served. This unproductive work causes the usable throughput, i.e. *goodput*, of the server to reduce drastically in an overloaded state, even though resource utilization remains at 100%. For E-commerce Web-sites, such degradation is especially harmful, as it may lead to significant loss in revenue, and loss of repeat customer visits.

To prevent such behavior, servers are often designed with an *overload control* mechanism [20]. An overload control mechanism aims to avoid the degradation of goodput of the server during periods of overload, and aims to maintain it at its maximum capacity value. Several different approaches and solutions for overload control have been proposed in the literature. Schemes such as admission control [6,23] or specialized scheduling policies [4], or a combination of both [2,10] have been proposed. A control-theory based approach to overload control is described by Abdelzaher et al. [1].

The LIFO scheduling discipline has been well known to improve throughput in overloaded telecommunication switches [9, 11, 20]. This idea of using LIFO during periods of overload has been invoked [8,21] in the context of Web server overload control as well. A solution based on the use of LIFO for overload control, tailored towards E-commerce Web servers, was proposed by Singhmar et al. (2004) [21]. The authors implemented this mechanism and experimentally showed that the use of LIFO discipline during overload results in higher overall throughput and lower mean response time in an overloaded state. However, their results also show that with either of the scheduling disciplines there is a significant degradation of goodput as the server becomes overloaded.

Although these experiments offered some insight into the behavior of overloaded software servers, we believe there is a need to study the behavior of these servers under overload, using *analytical* models. Several existing models [15, 17] were studied for their applicability to our scenario (i.e. overloaded Web server, with user abandonments and LIFO scheduling discipline). Although several existing models compare the LIFO and FIFO scheduling disciplines, none of them are able to reproduce the observed effect of drastic goodput degradation at overload, with the resource utilization remaining at 100%.

Our key observation is that this drastic degradation happens due to two reasons: contention of the software server threads for the hardware resources, and secondly, the time wasted in processing

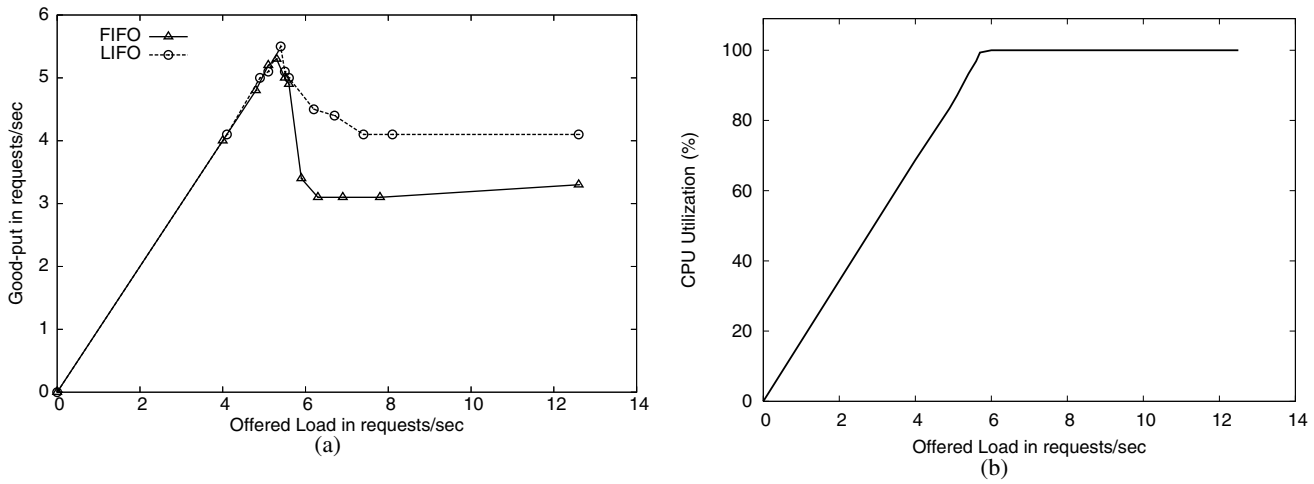


Figure 1: Goodput degrades after reaching a peak value, as the Web server becomes overloaded. The bottleneck resource utilization reaches and remains at 100%. Use of LIFO scheduling during overload results in higher average goodput. Experimental observations from [21].

overhead related to the requests. Thus, in an overloaded state, response times shoot up drastically due to software contention for hardware resources, leading to abandonments, while the same resources also spend unreasonable amount of time processing overhead, thus significantly reducing the amount of useful work that they can do. These are characteristics unique to *software* servers that the traditional models of queues with abandonments fail to represent.

The present work develops a model of a Web server at overload that is built on the premise that these characteristics, namely, the overhead processing and contention for resources must be accounted for, together with other factors such as abandonments, for proper comparison of the LIFO and FIFO scheduling disciplines. We do this by building a *layered queueing network* model [19], where two servers, one representing overhead and one representing actual processing, use a common hardware resource. Numerical results show that our model is able to reproduce the goodput degradation behavior at overload, in addition to showing that LIFO goodput is better than FIFO at overload.

In addition to the layered model for overloaded Web servers, we also present a model for calculating the response time distribution of the $M/M/C/K/LIFO$ queue with abandonments. Numerical results from the model show that in an overloaded state, the mean response time is much higher for FIFO as compared to LIFO. The variance of response time however, is higher for LIFO after overload sets in.

The rest of the paper is organized as follows. In Section 2 we briefly motivate our work and review existing literature on the topic of overload control. We present extended queueing models with LIFO scheduling and abandonments, including derivation of the response time distribution, in Section 3. Section 4 describes and solves our layered queueing model for overloaded servers with resource sharing and overhead processing, and presents numerical results. We conclude the paper in Section 5.

2. MOTIVATION AND BACKGROUND

Singhmar et al [21] had used the idea of switching to the LIFO scheduling discipline during overload to increase the throughput and improve the response time of Web servers at overload. They validated this idea experimentally, by implementing this mechanism in a Web server. Fig. 1 shows the results of an experiment where load was generated on a Web server that implements this mechanism. The figure shows that server goodput degrades drastically during overload. Nonetheless, by switching to LIFO at overload, the server is able to sustain a better goodput than FIFO. Additionally, the CPU utilization graph shows that in both cases, the CPU continues to operate at 100%, *even after the goodput degrades to nearly 60% of its peak*. Our goal is to develop an analytical model that captures the system characteristics that result in this behavior.

The system characteristics that we need to represent are impatient users with generally distributed timeouts, multi-threaded servers with limited buffers, whose threads contend for a shared hardware resource (e.g. the CPU) and spend some of the request service time in overhead processing.

Additionally, several types of server responses to user abandonments must be modeled—specifically, removal of abandoned requests from the buffer or server, as appropriate, must be represented. Most importantly, we would also like to analyze such a system with the FIFO as well as LIFO scheduling disciplines, on the basis of performance measures such as goodput and response time.

Keeping these characteristics in mind, we introduce an augmented Kendall notation in order to simplify the description of the queueing models that we study or develop in this paper. A queueing system in this extended notation is represented as:

$$A/S/C/K/P/D + T/R$$

where,

A, S represent the arrival process and the service time distribution respectively,

C is the number of servers,

K is the maximum number of requests, that can either be waiting or getting served, in the system—thus K is the sum of buffer size and number of servers,

P is the total population of the system (defaulting to infinity),

Table 1: Comparison of related work

Work Reference	Type	User Timeout Distribution	Removal of Timed-out Requests	No. of Server Processes	Buffer Capacity	Resource Sharing	Scheduling Policy
Reeser and Hariharan [18]	ANA	×	NONE	C	Fixed K	Time sharing	FIFO
Chen and Mohapatra [5]	ANA, EXP	×	NONE	C	Hybrid	×	FIFO
Dalal and Jordan [8]	ANA, SIM	D, EXP	NONE	1	Infinite	×	PS, FIFO, Greedy, LCFS-PR
Doshi and Heffes [9]	ANA	D, EXP	NONE	1	Fixed K, Infinite	×	FIFO, LIFO
Carter and Cherkasova [3]	SIM	D	BUF	?	Fixed 1024	×	FIFO
Movaghar [15]	ANA	G	BUF	C	Fixed K, Infinite	×	FIFO
Movaghar [16]	ANA	G	ANY	1	Fixed K, Infinite	×	FIFO
Pla et al. [17]	ANA	PH	BUF	C	Fixed K	×	FIFO, LIFO, SIRO
Our Model	ANA	EXP, ERL	NONE, BUF, ANY	C	Fixed K	Time sharing	FIFO, LIFO

D is the scheduling discipline (default being FIFO),

T is the distribution of user timeout values, which may be one of exponential(EXP), Erlang(ERL), general(G), phase-type(PH), deterministic(D), and

R describes removal policy of timed-out requests. It may be one of NONE, BUF, SRV, ANY; respectively representing no removal, removal from buffer only, server only or anywhere.

For example, M/M/1/10/LIFO + EXP/NONE would represent a single server queue with Poisson arrivals and exponentially distributed service time, with a buffer size of nine and exponentially distributed user timeouts with no removal of timed-out requests. The queueing discipline would be LIFO with total population defaulting to infinity. A dot (•) in the place of a parameter indicates that one of several choices may be used.

2.1 Existing Work

There is significant amount of literature that studies the behavior of queueing systems along the various aspects that we outlined earlier [5,8,9,15–18]. In order to compare the existing work in a structured manner, we characterize the work according to the following features: whether the model is simulation based or analytical, the timeout distribution assumed, the removal policy on timeout, the number of servers, the buffer size, whether resource sharing has been considered and the scheduling policies considered. Table 1 summarizes our comparative analysis of the existing work.

Reeser and Hariharan proposed a model [18] that takes into account blocking and queueing at various stages of processing for a typical Web server and models the contention for the CPU. However, they do not consider the case of impatient users, or the LIFO discipline.

A look at the scheduling policy column in Table 1 tells us that, from the listed work, only Dalal and Jordan [8], Doshi and Heffes [9] and Pla et al. [17] deal with some form of LIFO queueing discipline. Dalal and Jordan have used a LIFO policy with preemption of jobs (LCFS-PR). However, preemption of requests in service at a Web server is complex to implement and hence not typically done.

Doshi and Heffes provide a detailed comparison [9] of various queueing disciplines in a single server case and show that LIFO based schemes perform well under certain conditions when customers randomly abandon their requests while waiting for service. They however, do not consider multiple servers, resource sharing, or request removal on abandonment.

More recently, Pla et al. [17] and Movaghar [15, 16] have studied and solved models that involve queueing with impatient customers. As can be seen from Table 1, these models are quite general, and capture most of the characteristics that we require. Although they do not consider resource sharing, or overhead processing, they serve as building blocks for the different parts of the queueing network model that we propose in Section 4. The characteristics of our model are summarized in the last row of Table 1. In the next section, we describe these models and our extensions to them, in detail.

3. THE *M/M/C/K* QUEUE WITH LIFO AND ABANDONMENTS

Movaghar's work provides us with analysis for queues with impatient customers with deadlines until beginning of service [15] and till the end of service [16], with FIFO queueing discipline. Thus, using our notation, Movaghar gives us results for the $M/M/C/K/FIFO+G/BUF$ and the $M/M/C/K/FIFO+G/ANY$ queueing models. However, it is assumed in the $M/M/C/K/FIFO+G/BUF$ model that customers stop being impatient when they start receiving service. We use Movaghar's results for the $M/M/C/K/FIFO+G/ANY$ for modeling one of the queues, in the layered queueing network model presented in Section 4.

The model described by Pla et al., considers phase-type distribution for timeouts, the LIFO scheduling discipline and removal of abandoned requests from the buffer. Thus, in our notation, they give us results for the $M/M/C/K/FIFO+PH/BUF$ and

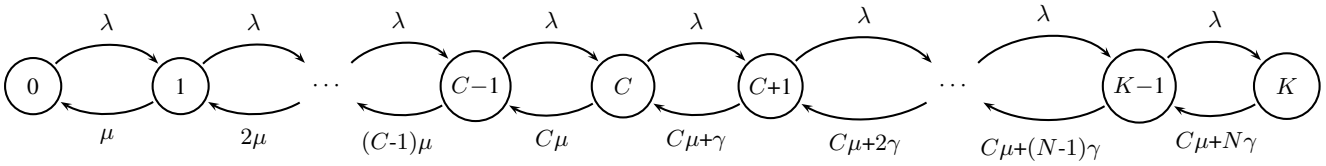


Figure 2: Markov chain for $M/M/C/K/\bullet + EXP/BUF$ model.

$M/M/C/K/LIFO + PH/BUF$ models. Here too, however, customers stop being impatient when they start receiving service.

We adapt Pla's model to our requirement as follows: first, we allow users to abandon even while their requests are in service and continue to occupy the server. Such requests, on completion do not count towards the goodput of the server.

Second, we use the tagged customer approach [14], to numerically calculate the response time distribution for the $M/M/C/K/\bullet + EXP/BUF$ model.

Third, we consider two special cases of the phase-type distribution for timeouts, namely the exponential and the Erlang distribution. This allows us to create and solve a simpler Markov chain model which represents the queueing system. Thus, we solve the $M/M/C/K/\bullet + EXP/BUF$, $M/M/C/K/\bullet + ERL/BUF$ and the $M/M/C/K/\bullet + ERL/NONE$ models. These are the basic models of an overloaded Web server, used later as building blocks for our overhead-aware layered model with resource sharing.

In the following subsections, we first introduce our simplest model, i.e., the $M/M/C/K/\bullet + EXP/BUF$ queue, and analyze it for its goodput and response time distribution. We then derive the goodput for the $M/M/C/K/\bullet + ERL/BUF$ queueing model.

3.1 The *M/M/C/K/• + EXP/BUF* model

Figure 2 depicts the Markov chain for a $M/M/C/K/\bullet + EXP/BUF$ system. User timeouts have a $EXP(\gamma)$ distribution. A state of this Markov chain consists of the total number of requests, i, in the buffer and servers combined. Thus $0 \leq i \leq K$. Also let λ be the mean arrival rate and μ be the mean service rate. Thus if π_i is the steady-state probability of being in state i, then the throughput of this system is given by [22]:

$$\Lambda = \sum_{i=1}^{C} i\mu\pi_i + \sum_{i=C+1}^{K} C\mu\pi_i \tag{1}$$

The probability P_{BAD}, that a request that is being serviced has been abandoned by the user before completion of service, is given by:

$$P_{BAD} = \left(\frac{\gamma}{\mu+\gamma}\right) \tag{2}$$

Therefore, the goodput is:

$$\Lambda_{GOOD} = \Lambda\left(1 - P_{BAD}\right) \tag{3}$$

$$= \frac{\mu^2}{\mu+\gamma}\left(\sum_{i=1}^{C} i\pi_i + C\sum_{i=C+1}^{K}\pi_i\right) \tag{4}$$

Clearly, this model is the same for both LIFO and FIFO scheduling disciplines. Thus, for exponentially distributed user timeouts, the goodput of LIFO and FIFO is the same. However, the response time characteristics—specifically, the *distribution* of response time—are known to differ. In the following, we derive the response time distribution of this queueing model, using the well-known tagged customer approach [14].

3.1.1 Response Time Distribution

The Markov chain described above was an ergodic Markov chain, solved at steady state, which gave us the means to obtain the goodput of the queueing system. Now we use the tagged-customer approach [14] to build a Markov chain that can be solved to give us the repose time distribution of a customer. This Markov chain represents the evolution of a tagged customer from the time that it arrives in the queue till it exits the system in one of three possible states:

Buffer Timeout: Timeout occurs while request is in buffer and request is removed from the system.

Unsuccessful Completion: Service finishes, but does not contribute to goodput. This will happen when user times-out while request is being served.

Successful Completion: Service finishes before user times-out. Adds to the goodput of the system.

Three absorbing states exist in this Markov chain, one corresponding to each of the exit states described above. The response time for the tagged customer is modelled as the time to absorption in this chain, given some initial starting state.

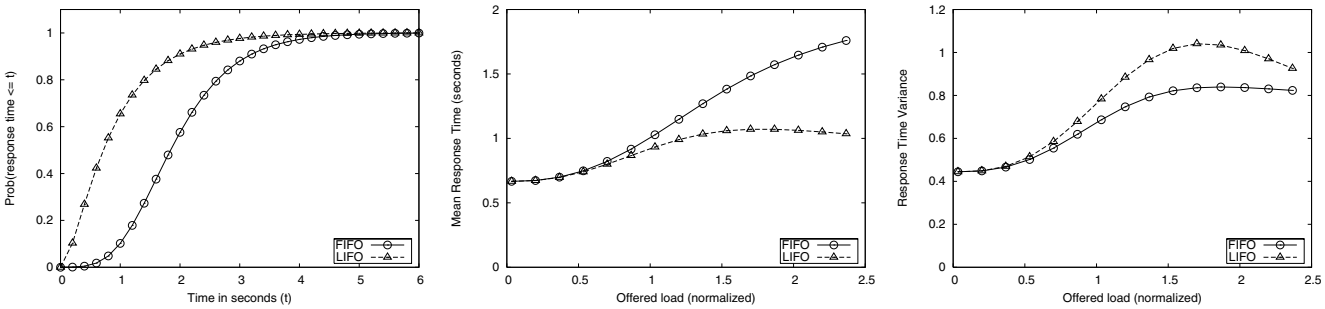
A non-absorbing state (l, m) of such a tagged customer Markov chain for the $M/M/C/K/\bullet + EXP/BUF$ system consists of the current number of requests l, and the position m of the tagged customer in the system. For this Markov chain, all C servers are assigned a position number 0 and buffer positions are numbered 1 to N, where $N = K - C$, with 1 being the head of the queue. Thus $0 \leq l \leq K$ and $0 \leq m \leq l - C$.

Arrivals, service completions and timeouts from the buffer will be similar to the ergodic Markov chain described earlier. To find the time-dependent probabilities $P_{(l,m)}(t)$ for the tagged customer chain, we need the initial state probabilities. These probabilities can be obtained from the steady-state probabilities π_i from the ergodic chain. Then for LIFO and FIFO we have the following non-zero starting state probabilities:

$$P^{LIFO}_{(l,m)}(0) = \begin{cases} \frac{\pi_{l-1}}{1-\pi_K}, & \text{for } 1 \leq l \leq C, m = 0 \\ \frac{\pi_{l-1}}{1-\pi_K}, & \text{for } C < l \leq K, m = 1 \end{cases}$$

$$P^{FIFO}_{(l,m)}(0) = \begin{cases} \frac{\pi_{l-1}}{1-\pi_K}, & \text{for } 1 \leq l \leq C, m = 0 \\ \frac{\pi_{l-1}}{1-\pi_K}, & \text{for } C < l \leq K, m = l - C \end{cases}$$

We define $P_{GOOD}(t)$ as the probability of being in the successful completion absorbing state at time t; this is then the cumulative distribution function (CDF) of the response time of a successfully completed request. Similarly, we define $P_{BAD}(t)$ and $P_T(t)$ as the CDFs of time to unsuccessful completion and buffer timeout respectively. The probability of a given request eventually ending



(a) CDF of response time for requests that finish successfully. λ = 12 requests/s.

(b) Mean response time vs. offered load.

(c) Response time variance vs. offered load.

Figure 3: Response time CDF, mean and variance for a $M/M/3/10/\bullet + EXP/BUF$ queueing model. $1/\gamma$ = 2.0s, μ = 1.0 request/s.

up in each of the absorbing states is given by:

$$P_{GOOD} = \lim_{t\to\infty} P_{GOOD}(t), \quad P_{BAD} = \lim_{t\to\infty} P_{BAD}(t), \quad P_T = \lim_{t\to\infty} P_T(t). \qquad (5)$$

One can evaluate the tagged-customer Markov chain with absorbing states described above using a tool like SHARPE [7] to obtain the CDF of response time of requests that finish in each of the absorbing states. Results depicted in Fig. 3(a) show the conditional CDF of time to successful completion of a request (which counts towards goodput of the system). From the figure, it can be seen that about 65% of successfully completed requests have a response time less than or equal to 1.0s for LIFO, whereas this value is only about 10% for FIFO. Additionally, the 90^{th} percentile values for the conditional response time of successful requests for LIFO and FIFO are 2.0s and 3.2s respectively.

Figures 3(b) and 3(c) show the mean and the variance, respectively, of the response time distribution of successful requests for increasing offered load. We can see that both LIFO and FIFO have nearly the same means up to the point at which offered load reaches 1, after which, the mean response time of FIFO continues to increase, while that of LIFO flattens out earlier, at a lesser value. The variance of the LIFO response time is, however, much more than that of FIFO, after overload. Due to the higher variance in LIFO, out of the requests that are taken up for service, a larger fraction avoids getting abandoned and hence P_{BAD} is lower.

Thus we see that with exponential timeouts, even though the goodput is the same in both LIFO and FIFO, the response time distribution of requests for LIFO is significantly better than that for FIFO.

3.2 The *M/M/C/K/• + ERL/•* model

An important aspect of the model of an overloaded server is the distribution of timeouts. Here we refer to the variable *user* timeouts rather than machine/browser timeouts that are usually fixed at a large value. From Table 1, we observe that authors have assumed fixed [3], exponential [9], phase-type [17] or general distribution [15, 16] for user timeouts.

Experience suggests that most users will wait for some minimum amount of time before abandoning their requests leading to a non-zero value for the most probable timeout duration. This is contrary to the well-known fact that the mode of an exponential distribution is at zero. In order to model user timeouts more realistically, we choose an Erlang distribution for user timeouts. The mode for an $Erlang(R, \lambda)$ distribution is at $\frac{R-1}{\lambda}$, which is a non-zero value for $R \geq 2$. This is in agreement with the reasoning given above. For the sake of simplicity, we evaluate our model for the case of $R = 2$.

We solve this $M/M/C/K/\bullet + ERL/\bullet$ model for BUF and $NONE$ removal policies. BUF represents removal of timed-out requests only from the buffer, while $NONE$ implies that timed-out requests leave the system only after receiving full service irrespective of where the request is when the user times out. Thus $NONE$ represents an extreme case where there is no possibility of unproductive work being removed from the system.

For this system, the Markov chains for LIFO and FIFO processing will be different since the customers may be in different stages of impatience and hence their position is a part of the system state. The mean delay of each exponential stage is set to $1/R\gamma$ such that the overall mean user timeout of R impatience stages is

$$R\left(\frac{1}{R\gamma}\right) = \frac{1}{\gamma}.$$

In order to completely describe the state of the system with this $Erlang(R, R\gamma)$ timeout distribution, we need to keep track of the impatience stage r, that each request is currently in, where $1 \leq r \leq R$.

A state of the Markov chain for this $M/M/C/K/\bullet + ERL/\bullet$ system is denoted as $s = (i_{K-1}, \ldots, i_0)$, $s \in S$ where S is the state space of the CTMC. Each i_k, where $0 \leq k < K$, represents the state of a position in the servers or queue. Here subscripts 0 to $(C-1)$ represent servers and C to $(K-1)$ are queue positions. The possible values of the state, i_k, of the position k are

$$i_k = \begin{cases} 0, & \text{position } k \text{ is empty,} \\ r, & \text{impatience stage of request, } 1 \leq r \leq R, \\ -1, & \text{user timed out, but request still in system.} \end{cases}$$

For example, a state $(0, 0, 0, 1, 2, -1)$ for C=3, K=6 signifies an empty queue, one server with a request in first stage of impatience, one in the second stage and the third serving a request from a timed-out user.

Rules for construction of the Markov chain are as follows. Transitions in the internal state i_k of a position in the system are only possible in the order: $0 \to 1 \to 2 \to \ldots \to (R-1) \to R \to -1$. The difference in the LIFO and FIFO scheduling is captured by the way arrivals are enqueued. If no server is free, and there is at least one position empty in the queue, an arrival occurs as follows. In case of LIFO queueing discipline, the arriving request will join at the head of the queue, with all existing requests in the queue being pushed back one position. For FIFO, it will join at the back of the queue.

The infinitesimal generator matrix $Q = [q_{I,J}]$ of this CTMC can be constructed from the following rules. Let $I = (i_{K-1}, \ldots, i_0)$ be the current state and $J = (j_{K-1}, \ldots, j_0)$ be the resultant state after the transition. Also let p be the position under consideration, where $0 \leq p < K$. The head of the queue is at position C and let the first empty position at the back of the queue be denoted by B. Then $\forall i_k$ such that $I \in S$, we have the following.

Arrival in server: $q_{I,J} = \lambda$ for $p = \min\{x \mid 0 \leq x < C, i_x = 0\}$ where

$$j_k = \begin{cases} i_k, & \text{for } k \neq p \\ 1, & \text{for } k = p \end{cases}$$

Arrival in queue (LIFO): $q_{I,J} = \lambda$ for $p = C$ and LIFO scheduling where

$$j_k = \begin{cases} i_k, & \text{for } 0 \leq k < C \\ i_{k-1}, & \text{for } C < k < K \\ 1, & \text{for } k = p \end{cases}$$

Arrival in queue (FIFO): $q_{I,J} = \lambda$ for $p = B$ and FIFO scheduling where

$$j_k = \begin{cases} i_k, & \text{for } k \neq p \\ 1, & \text{for } k = p \end{cases}$$

Service completion: $q_{I,J} = \mu$ for $0 \leq p < C$ where

$$j_k = \begin{cases} i_C, & \text{for } k = p \\ i_k, & \text{for } k \neq p,\ 0 \leq k < C \\ i_{k+1}, & \text{for } C < k \leq (K-2) \\ 0, & \text{for } k = (K-1) \end{cases}$$

Change in impatience stage: $q_{I,J} = \gamma_r$ for $0 \leq p < K$, $1 \leq r < R$ where

$$j_k = \begin{cases} i_k, & \text{for } k \neq p \\ i_p + 1, & \text{for } k = p \end{cases}$$

User timeout: $q_{I,J} = \gamma_R$ for $0 \leq p < K$ and $ERL/NONE$ abandonments where

$$j_k = \begin{cases} i_k, & \text{for } k \neq p \\ -1, & \text{for } k = p \end{cases}$$

$q_{I,J} = \gamma_R$ for $0 \leq p < C$ and ERL/BUF abandonments where

$$j_k = \begin{cases} i_k, & \text{for } k \neq p \\ -1, & \text{for } k = p \end{cases}$$

$q_{I,J} = \gamma_R$ for $C \leq p < K$, ERL/BUF abandonments where

$$j_k = \begin{cases} i_k, & \text{for } 0 \leq k < p \\ i_{k+1}, & \text{for } p \leq k \leq (K-2) \\ 0, & \text{for } k = (K-1) \end{cases}$$

Figure 4 shows the Markov chain for a $M/M/1/2 \bullet + ERL/BUF$ system. Arrivals are $EXP(\lambda)$, service time distribution is $EXP(\mu)$ and timeout distribution is $Erlang(2, 2\gamma)$. In the figure, γ_1 signifies transitions from I stage of impatience to II stage and γ_2 refers to transitions from II stage of impatience to a timed-out state. The values of γ_1 and γ_2 are equal to twice the mean rate of timeout γ. Note that the buffer size of 1, used in Fig. 4 for simplicity of chain, is a special case where LIFO and FIFO chains turn out to be identical as there is only one buffer location.

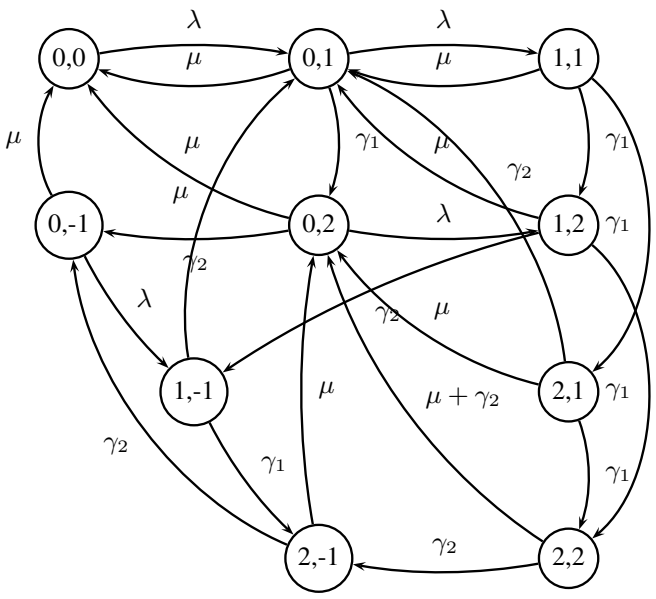


Figure 4: Markov chain for a $M/M/1/2/\bullet + ERL/BUF$ system. User timeouts have $Erlang(2, 2\gamma)$ distribution. $\gamma_1 = \gamma_2 = 2\gamma$

The Markov chain can be solved using a solver like PRISM [13] or SHARPE [7] to obtain the steady-state probabilities π_s of being in a state $s \in S$. The probability fractions of an incoming request getting blocked (P_B), timed-out and removed (P_T), successful service completion (P_{GOOD}), and unsuccessful service completion (P_{BAD}) can then be derived as follows.

For any given state $s \in S$, let $R_T(s)$ be the rate at which requests time-out and are removed from the system, in state s. Let $R_C(s)$ be the total completion rate of requests, of which $R_{BAD}(s)$ is the rate of completion of abandoned requests. To write the expressions for these rates, we first define a function Φ_r as follows: for any position $i_k \in s$, where $0 \leq k < K$,

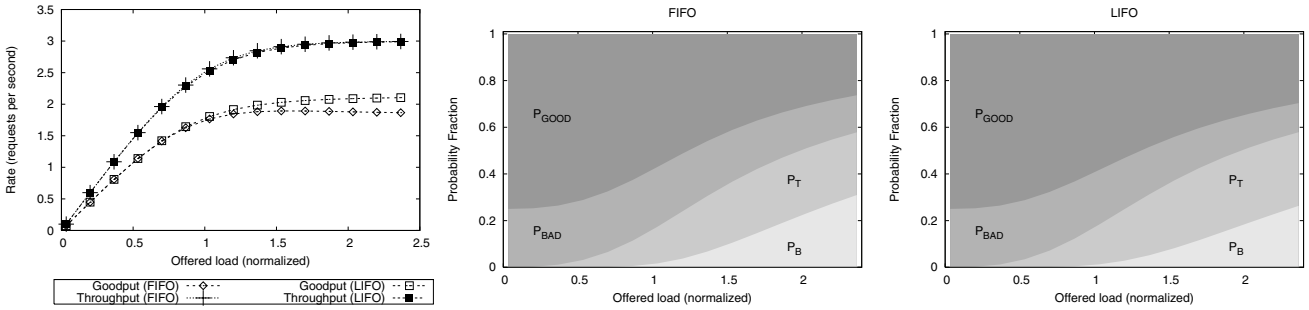
$$\Phi_r(i_k) = \begin{cases} 1, & \text{for } i_k = r \\ 0, & \text{for } i_k \neq r \end{cases} \tag{6}$$

Also, let $S_n \subset S$ be defined as:

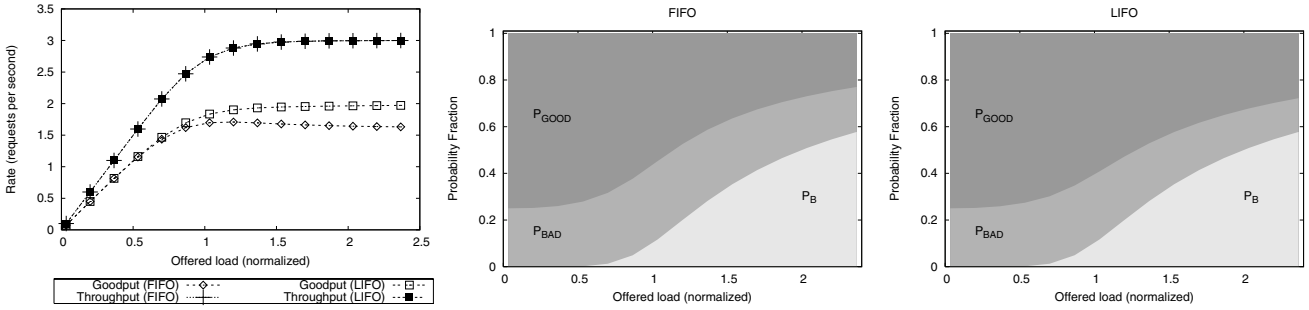
$$S_n = \left\{ s \mid s \in S, \left(K - \sum_{k=0}^{K-1} \Phi_0(i_k) \right) = n \right\} \tag{7}$$

Now, we can express the rates $R_T(s)$, $R_C(s)$ and $R_{BAD}(s)$ for $s \in S$ and $i_k \in s$ as follows:

$$\begin{aligned} R_T(s) &= \gamma_R \sum_{k=C}^{K-1} \Phi_R(i_k) \\ R_C(s) &= \mu \left(C - \sum_{k=0}^{C-1} \Phi_0(i_k) \right) \\ R_{BAD}(s) &= \mu \sum_{k=0}^{C-1} \Phi_{-1}(i_k) \end{aligned} \tag{8}$$



(a) $M/M/3/10/\bullet + ERL/BUF$ System



(b) $M/M/3/10/\bullet + ERL/NONE$ System

Figure 5: Comparison of throughput, goodput and probabilities of a request getting blocked (P_B), timed-out & removed (P_T), completed but user has timed-out (P_{BAD}) or completed successfully (P_{GOOD}) for two removal policies. $1/\gamma$ = 2.0s, μ = 1.0 request/s.

Using Equations (6)–(8), we define

$$\Lambda_B = \lambda \sum_{s \in S_K} \pi_s, \qquad \Lambda_{BAD} = \sum_{s \in S} R_{BAD}(s)\pi_s,$$
$$\Lambda_T = \sum_{s \in S} R_T(s)\pi_s, \qquad \Lambda_C = \sum_{s \in S} R_C(s)\pi_s. \quad (9)$$

Then, if λ is the rate of incoming requests, we have:

$$\begin{aligned} \lambda &= \Lambda_B + \Lambda_T + \Lambda_C \\ &= \Lambda_B + \Lambda_T + \Lambda_{BAD} + \Lambda_{GOOD} \end{aligned} \quad (10)$$

From Eq. (10) and (9) we can get the required probabilities, throughput and goodput as

$$P_{BAD} = \frac{\Lambda_{BAD}}{\lambda}, \qquad P_{GOOD} = \frac{\Lambda_C - \Lambda_{BAD}}{\lambda},$$
$$P_B = \frac{\Lambda_B}{\lambda}, \quad P_T = \frac{\Lambda_T}{\lambda}, \quad \Lambda_{GOOD} = \Lambda_C - \Lambda_{BAD}. \quad (11)$$

We use the above equations to derive numerical results for a $M/M/3/10/\bullet + ERL/\bullet$ system, for the FIFO and LIFO scheduling disciplines and for the BUF and $NONE$ cases for request removals (Fig. 5). Results for both models (BUF as well as $NONE$) show that, though the raw throughput (Λ_C) does not show much difference between FIFO and LIFO, the goodput Λ_{GOOD} is better for LIFO when the server is overloaded. Furthermore, for the $NONE$ case the goodput for both LIFO and FIFO is worse than that of BUF, and the difference between throughput and goodput is also more significant.

The better goodput for LIFO can be explained by observing the probability fraction charts in Fig. 5(a). Recall that the probability of a request completing successfully is given by $P_{GOOD} = 1 - P_B - P_T - P_{BAD}$. Goodput for LIFO turns out to be better than FIFO primarily because P_{BAD} (i.e. the probability of unsuccessful completion) is lower for LIFO, while $P_B + P_T$ is the same for both LIFO and FIFO.

Results for the $NONE$ case (Fig. 5(b)) are similar to the BUF case; they show that the blocking probability P_B is almost the same for both LIFO and FIFO for all levels of offered load (and P_T is 0 as their are no removals). The goodput, however, is better for LIFO as P_{BAD} is lower for LIFO during overload.

3.3 Model Summary

From the analysis of the basic $M/M/C/K/\bullet+\bullet/\bullet$ model of an overloaded server, we can conclude that although non-exponential user timeout distribution is necessary to explain the better goodput performance of LIFO as compared with FIFO at overload, LIFO shows better response time characteristics than FIFO at overload, even with exponential timeouts.

We also see that though the basic model predicts goodput reducing to a fraction of raw throughput as the system becomes overloaded, it is still unable to explain the observed drastic drop (Fig. 1) and the subsequent flattening-out of goodput during overload. In the next section we present our layered queueing network model based on the models developed in this section, which captures the specific behavior of *software* servers, namely overhead processing, and hardware resource sharing.

4. LAYERED MODEL WITH OVERHEAD AND RESOURCE SHARING

In a Web server, there are many overheads that must be incurred even before actual processing can start. This includes factors such as the TCP connection setup time and the request preprocessing that happens at the root *listener* thread in a Web server [21]. This

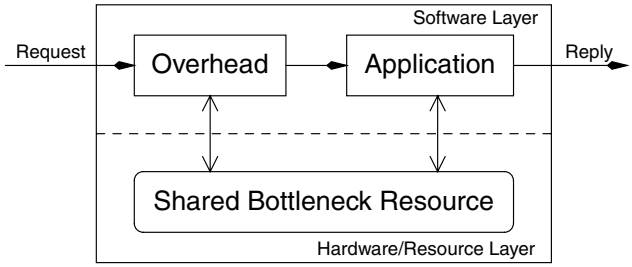


Figure 6: High-level model of the processing of a request at a server. Overhead (OHD) and application (APP) processing stages contend for the shared bottleneck resource.

thread is typically responsible for receiving the HTTP request from the client, parsing, possibly carrying out security checks, and finally enqueueing it to wait for a *worker* thread to become available. Many Web servers (e.g. Sun Java Web System, IBM WebSphere) are based on a variant of this design. This overhead adds to the unproductive work being done by the CPU, which is the bottleneck resource for dynamic requests.

In our proposed model, the *listener* and *worker* threads are modelled as separate queueing systems or *stages*. The processing by the listener thread represents the overhead processing, while the processing by the worker threads represents the "useful" processing (henceforth termed as the *application* processing). While this abstraction is grounded in reality, it is true that part of the processing at the application stage may also be classified as "overhead" (e.g. paging, swapping, context switching overheads incurred at this stage). However, we make a simplifying assumption of clubbing together all the overhead in a single initial overhead stage. Both these servers share the underlying bottleneck resource—thus it is a layered queueing network model (Fig. 6).

The overhead and application stages are represented using the queueing system models developed in the previous sections. The threads of these stages are in turn customers for the CPU queueing system. In the rest of this section, we provide details of this model and present numerical results.

4.1 Model Description

Figure 6 shows a high-level representation of our model of a networked software server. The three main components of this picture are the overhead (OHD) processing stage, the application (APP) processing stage in the software layer, and the shared resource in the hardware layer.

We consider the CPU to be the bottleneck resource, even though similar mechanisms may be applied to any known bottleneck resource. We thus have a two-layer model with the OHD and APP processing on the software layer and the CPU at the device layer [19]. The choice of the specific model for each block of the picture is made on the basis of the characteristics of that block. For example, if CPU and network I/O are significant in a system, then only the device model inside the shared resource block (Fig. 6) need be changed. Since in this paper we are taking the example of a Web server with CPU as the bottleneck resource, our choice of specific models is as follows.

4.1.1 Overhead and Application Stages

The overhead (OHD) processing block represents overheads that need to be incurred for a request to be serviced by the Web server. The server in the OHD block represents the single listener thread. The queue in front of this server is essentially the TCP listen-queue. This queue is at the operating system level and is processed with the FIFO queueing discipline. Also, the timeout distribution here is assumed to be exponential to simplify analysis since we do not use LIFO at this queue. If, at any time, a timeout occurs in the OHD stage and the client closes the connection, the processing is aborted and the request is removed from the system. This is feasible since most of the system state is limited to the listener thread itself. A $M/M/1/K/FIFO + EXP/ANY$ queueing system is thus appropriate for this block of processing. We use the model described by Movaghar [16] to represent this queue. In case a system has multiple listener threads, a $M/M/C/K/FIFO + EXP/ANY$ queueing system may be used, the rest of the analysis remains the same. However, we have not considered such a case in this paper.

The processing done by the worker thread is denoted as the application (APP) processing. In case of a request for static content, this is equivalent to reading a file from storage and sending it to the user. Serving of the file can be aborted at any stage, should a client timeout occur. Strategies for doing this have been suggested by Carter and Cherkasova [3]. However, in this paper, we are concentrating on the case of requests for dynamically generated content, where the CPU is the bottleneck resource. To serve such a request, a scripting engine[1] is invoked to parse the input data and carry out the actual processing (database lookups, calls to other subsystems, calculations, etc.) required to generate the reply. A request for dynamic content is usually not aborted and removed from the system once it enters service at the APP processing block due to the way a typical Web server handles requests for dynamic content. Once the scripting engine has been invoked, it is not aborted until the processing is complete to avoid loss of system state possibly created in different sub-systems. Thus, while timed-out requests may be removed from buffer, removal from server is not possible. Hence the possible request removal policies for this stage are BUF and $NONE$. Since there are multiple worker threads, a $M/M/C/K$ model with (BUF) or without ($NONE$) removal of timed-out requests is appropriate for the APP part. The models $M/M/C/K/\bullet + ERL/\bullet$ discussed in Section 3.2 can be used for the APP block.

A detailed view of the model, using the $M/M/C/K/FIFO + EXP/ANY$ queue for the OHD stage and a $M/M/C/K/\bullet + ERL/\bullet$ queue for the APP stage, is shown in Fig. 7. A request entering the system may leave from various points in the system. These possible points of exit and their respective probabilities are depicted in Fig. 7.

The notation used in the figure, and the various parameters of the model are explained in detail below.

4.1.2 Notation and Model Parameters

The parameters of the model depicted in Fig. 7 and related performance metrics are described below.

λ_{IN} is the mean request arrival rate at the server.

λ_{OHD} is the mean request rate offered to the overhead part. $\lambda_{OHD} = \lambda_{IN}$.

Λ_{OHD}, Λ_{APP} are the mean throughputs of OHD and APP stages respectively.

λ_{APP} is the mean request rate offered to the APP part. This is the same as Λ_{OHD}.

[1]A scripting engine (for instance an interpreter for scripting languages like Perl, PHP, Python, shell) is typically an entity external to the core of the Web server.

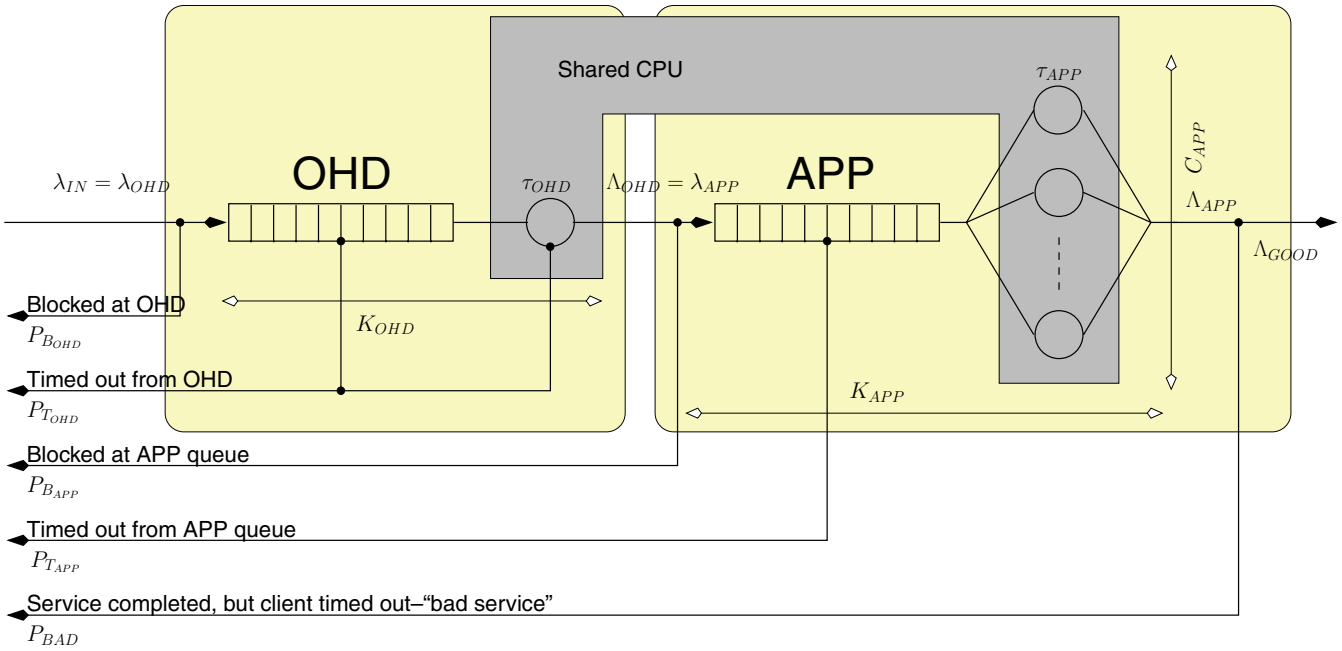


Figure 7: Two-stage model of processing a request at a Web server serving dynamic content. OHD is the overhead, APP is the actual request processing. Probabilities of the request leaving the system due to blocking or customer timing out have been indicated.

K_{OHD}, K_{APP} are the maximum number of requests in the OHD and APP parts, including the ones in service.

C_{APP} is the number of worker (APP) threads/processes in the server.

τ_{OHD}, τ_{APP} are the mean service time requirements of a request at the OHD and APP parts, given exclusive access to the CPU.

τ'_{OHD}, τ'_{APP} are the mean service time requirements of a request at the OHD and APP parts, after compensating for CPU sharing.

$P_{B_{OHD}}$, $P_{B_{APP}}$ are the probabilities of a request being blocked at the OHD and APP stages.

For users who become impatient and abandon a request, we have the following parameters and metrics.

T_0 is the mean user timeout duration, and

γ is the average impatience rate defined as $1/T_0$.

$P_{T_{OHD}}$, $P_{T_{APP}}$ are the probabilities of a request being timed-out and removed from the system for the OHD and APP stages.

P_{BAD} is the probability that a request was successfully completed, but the customer had abandoned the request while it was receiving service at the APP stage. This adds to 'bad processing' and does not count towards the usable throughput (i.e. goodput).

Λ_{GOOD} is the goodput of the system, and is expressed as $\Lambda_{APP}\,(1 - P_{BAD})$.

U_{CPU}, $U_{CPU_{GOOD}}$ are the total utilization level of the CPU, and the "good" utilization of the CPU, respectively.

4.1.3 Shared CPU: Layered Model

The CPU is the bottleneck resource and all active threads/processes in the system have to contend for the CPU. To model this contention, we use a *time sharing* model for the shared CPU. According to this model, the service time requirements of each active thread, namely τ_{OHD} and τ_{APP} are linearly scaled up by a factor (n_{CPU}) equal to the average number of active threads in the system. It has been shown [18] that such a linear scaling model is more accurate as compared to a standard Processor Sharing (PS) model with non-linear scaling [12]. Context-switching time is assumed to be negligible.

This is a layered model with the service times at the OHD and APP queues being the response time of the respective thread from the shared CPU [19]. The linear scaling is applied to obtain the inflated service time requirements τ'_{OHD} and τ'_{APP} as follows:

$$\tau'_{OHD} = n_{CPU}\,\tau_{OHD} \quad (12)$$

$$\tau'_{APP} = n_{CPU}\,\tau_{APP} \quad (13)$$

$$n_{CPU} = \max(1, n_{OHD} + n_{APP}) \quad (14)$$

and by applying Little's Law to *just the servers* (not including queues) in both OHD and APP processing stages, we get

$$\begin{aligned} n_{OHD} &= \Lambda_{OHD}\,\tau'_{OHD} \\ &= \lambda_{OHD}\,(1 - P_{B_{OHD}} - P_{T_{OHD}})\,\tau'_{OHD} \quad (15) \\ n_{APP} &= \Lambda_{APP}\,\tau'_{APP} \\ &= \lambda_{APP}\,(1 - P_{B_{APP}} - P_{T_{APP}})\,\tau'_{APP}. \quad (16) \end{aligned}$$

The required probabilities are obtained from a solution of the respective stage models. In the next subsection, we describe the procedure of solving this layered queueing model.

4.2 Solving the Model

We are interested in the throughput performance metrics of the system to be able to explain the observed goodput drop and bet-

ter performance of LIFO during periods of overload. The inter-dependence of τ'_{OHD}, n_{CPU}, and n_{OHD} (and similarly for APP) is evident from Eq. (12)-(16). Hence, this model is solved *iteratively* starting with the supplied initial values of τ_{OHD}, τ_{APP} and $n_{CPU} = 1$.

Algorithm 1 *Iterative Solution of Two-Stage Shared-CPU Model*

Inputs: λ_{IN}, τ_{OHD}, $C_{OHD} = 1$, K_{OHD}, τ_{APP}, C_{APP}, K_{APP}, γ

$\lambda_{OHD} \leftarrow \lambda_{IN}$
$\tau'_{OHD} \leftarrow \tau_{OHD}$
$\tau'_{APP} \leftarrow \tau_{APP}$

repeat
From Eq. (17) and (18)

$(P_{B_{OHD}}, P_{T_{OHD}}) \leftarrow$ **OHD_MODEL**$(\lambda_{OHD}, \tau'_{OHD}, C_{OHD}, K_{OHD}, \gamma)$

$\Lambda_{OHD} \leftarrow \lambda_{OHD}\,(1 - P_{B_{OHD}} - P_{T_{OHD}})$

$\lambda_{APP} \leftarrow \Lambda_{OHD}$

From Eq. (11),

$(P_{B_{APP}}, P_{T_{APP}}, P_{BAD}) \leftarrow$ **APP_MODEL**$(\lambda_{APP}, \tau'_{APP}, C_{APP}, K_{APP}, \gamma)$

$\Lambda_{APP} \leftarrow \lambda_{APP}\,(1 - P_{B_{APP}} - P_{T_{APP}})$

CPU_LINEAR_SCALING :
$n_{OHD} \leftarrow \Lambda_{OHD}\,\tau'_{OHD}$
$n_{APP} \leftarrow \Lambda_{APP}\,\tau'_{APP}$
$n_{CPU} \leftarrow \max(1, n_{OHD} + n_{APP})$

$U_{CPU} \leftarrow \Lambda_{OHD}\,\tau_{OHD} + \Lambda_{APP}\,\tau_{APP}$

$\tau''_{OHD} \leftarrow n_{CPU}\,\tau_{OHD}$
$\tau''_{APP} \leftarrow n_{CPU}\,\tau_{APP}$
$\Delta \leftarrow |\tau''_{APP} - \tau'_{APP}|$

$\tau'_{OHD} \leftarrow \tau''_{OHD}$
$\tau'_{APP} \leftarrow \tau''_{APP}$
until $\Delta < \epsilon$

$\Lambda_{GOOD} \leftarrow \Lambda_{APP}\,(1 - P_{BAD})$
$U_{CPU_{GOOD}} \leftarrow \Lambda_{GOOD}\,\tau_{APP}$

Outputs: Λ_{GOOD}, τ'_{OHD}, τ'_{APP}, n_{CPU}, $P_{B_{OHD}}$, $P_{T_{OHD}}$, $P_{A_{OHD}}$, $P_{B_{APP}}$, $P_{T_{APP}}$, P_{BAD}, Λ_{OHD}, Λ_{APP}, U_{CPU}

Algorithm 1 outlines the iterative solution process of our two-stage, shared-resource model. In the iterative solution there are two steps **OHD_MODEL** and **APP_MODEL** that solve the individual models for the OHD and APP parts. Using each of the sub-models, we need to determine the probabilities of an incoming request blocking, getting timed out and removed, or completing unsuccessfully. These probabilities are then used to determine the throughput and goodput at each stage. In each successive iteration, the linear scaling model for CPU sharing is used to update the scaled-up service time requirement (τ'_{OHD}, τ'_{APP}) of each stage. The **OHD_MODEL** and **APP_MODEL** steps are discussed below. The algorithm terminates when the error value, Δ, between successive iterations falls below a predefined limit ϵ.

4.2.1 OHD Stage Model Solution

We use the $M/M/1/K/FIFO{+}EXP/ANY$ model (removal of timed-out customers till end of service) to represent the OHD stage. The solution to this queueing model is available from results published by Movaghar [16] as follows.

Let p_n be the probability of having n requests at the OHD stage where $0 \le n \le K_{OHD}$. Also let $\mu_{OHD} = 1/\tau'_{OHD}$. Then p_n is derived from a special case of [16] as:

$$p_n = \frac{\prod_{i=1}^{n} \frac{\lambda_{OHD}}{\mu_{OHD}+i\gamma}}{1 + \sum_{j=1}^{K} \prod_{i=1}^{j} \frac{\lambda_{OHD}}{\mu_{OHD}+i\gamma}}.$$

Thus the probabilities of blocking and timing out at the OHD stage, respectively, are:

$$P_{B_{OHD}} = p_{K_{OHD}} \tag{17}$$

$$P_{T_{OHD}} = 1 - p_{K_{OHD}} - \sum_{n=0}^{K_{OHD}-1} p_n \frac{\mu_{OHD}}{\mu_{OHD} + (n{+}1)\gamma} \tag{18}$$

4.2.2 APP Stage Model Solution

The APP stage is modelled as a $M/M/C/K/\bullet + ERL/\bullet$ queueing system, with removal of abandoned requests from the buffer (BUF) or without any removal ($NONE$). The input to the APP stage is assumed to be a Poisson stream with rate λ_{APP}. The distribution of service times at the CPU is $EXP(\mu_{APP})$ where $\mu_{APP} = 1/\tau'_{APP}$. The Markov chain for this model has already been discussed in Section 3.2. Thus the probabilities $P_{B_{APP}}$, $P_{T_{APP}}$ and P_{BAD} are obtained from a solution of the Markov chain as described earlier (Eq. (11)).

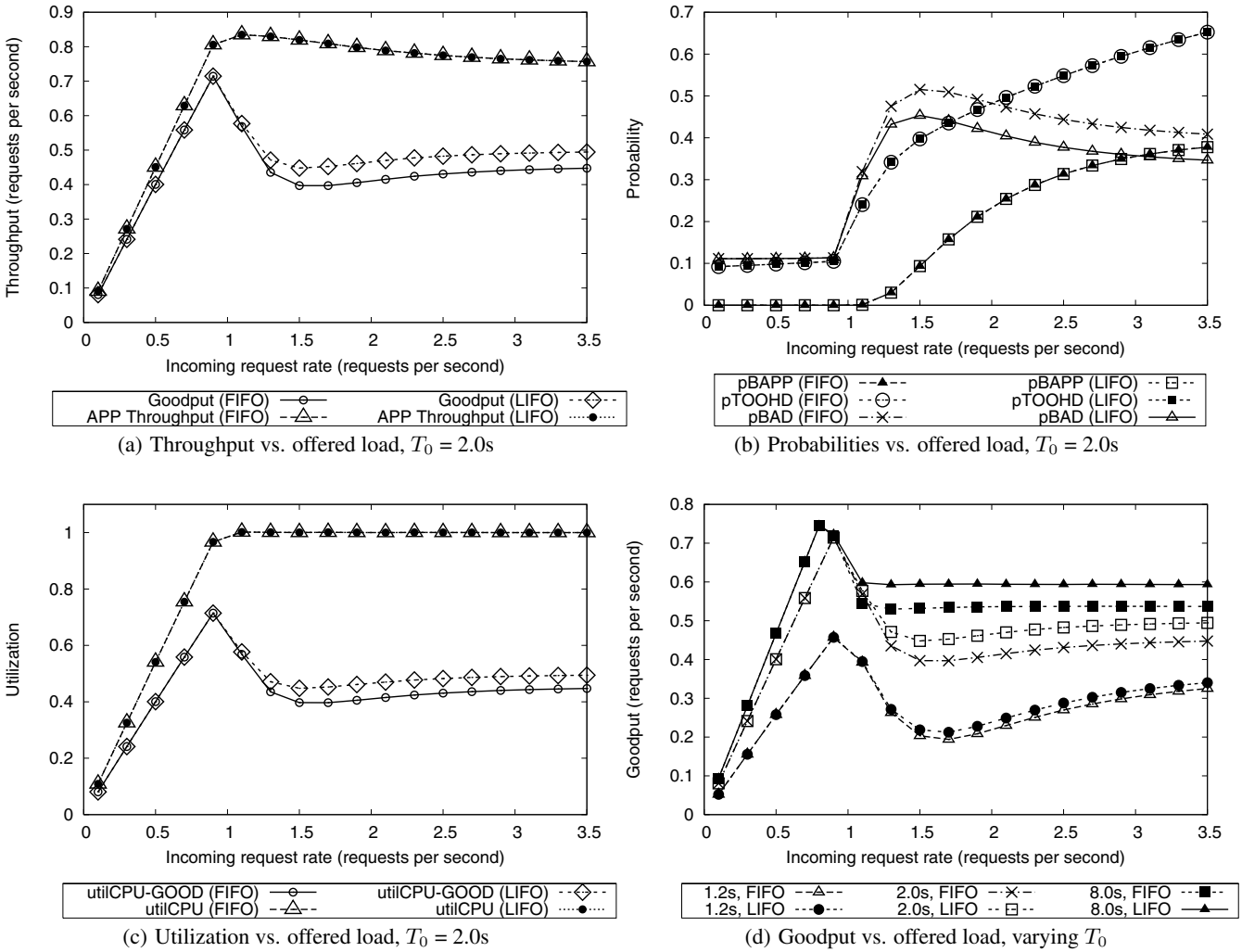
4.3 Results and Discussion for Complete Model

We now present results for the complete layered model with shared CPU and overhead processing, while using LIFO and FIFO at the APP stage, from Algorithm 1.

Fig. 8 shows results for a case where the OHD stage model is $M/M/1/101/FIFO{+}EXP/ANY$ and τ_{OHD} = 0.2s. The APP stage model is $M/M/3/13/\bullet{+}ERL/NONE$ with τ_{APP} = 1.0s. User timeouts have a $Erlang(2, 2\gamma)$ distribution where T_0=2.0s.

The throughput vs. offered load graph in Fig. 8(a) shows that in the region where the server is overloaded, i.e. normalized offered load (ρ) > 1, LIFO shows a better goodput as compared to FIFO discipline. This graph also clearly depicts that there is a sudden drop and subsequent flattening-out of the goodput, to about 60% of its peak value for FIFO, as ρ increases beyond 1. Figure 8(c) also shows the fraction of time that the CPU spends in useful work ($U_{CPU_{GOOD}}$). As can be seen clearly, initially (when $\rho < 1$), the CPU spends most of its time doing useful work while when $\rho \gg 1$, even though the utilization of the CPU is at 100%, the fraction of time spent in productive work is only about 40–50%. Thus the sharp degradation of goodput while the bottleneck resource remains fully utilized, as was experimentally observed in Fig. 1, is reproduced by this model.

The Web server becomes overloaded when the CPU becomes saturated with work and the processing time of each stage gets inflated (Eq. (12) and (13)) as offered load increases further. This increase in service time requirement causes delays in the processing of requests and the impatient users start abandoning requests. These abandoned requests may be processed by the server and contribute to the unproductive work. Fig. 8(b) shows the increase



(a) Throughput vs. offered load, T_0 = 2.0s

(b) Probabilities vs. offered load, T_0 = 2.0s

(c) Utilization vs. offered load, T_0 = 2.0s

(d) Goodput vs. offered load, varying T_0

Figure 8: Overall throughput, service time, probabilities and utilization vs. offered load for proposed layered model with overhead. The APP model is $M/M/3/13/\bullet + ERL/NONE$ while the OHD model is $M/M/1/101/FIFO + EXP/ANY$. τ_{OHD} = 0.2s and τ_{APP} = 1.0s. The degradation & flattening of goodput, and better performance of LIFO are clearly observed.

in probabilities of blocking at APP queue, timeout in the OHD queue and unproductive service. Thus when overloaded, goodput degrades since the CPU is busy doing unproductive work including the overhead processing and users are abandoning requests (Fig. 8(b), P_{BAD} & $P_{T_{APP}}$ respectively).

The degradation of goodput 'flattens out', as offered load increases, when the queue of one of the stages fills up and it starts blocking. This is observed at around the $\rho = 1.5$ point in Fig. 8 where $P_{B_{APP}}$ becomes significant. Figure 8(d) depicts the goodput curves for LIFO and FIFO disciplines with varying values of mean user timeout duration. The peak goodput, as well as the value at which it flattens out, increases with increasing patience in users.

Note that in the results presented here, the values of some input parameters of the model (e.g. K_{OHD}=101, C_{APP}=3, and K_{APP}=13) are chosen to illustrate the behavior of interest, and are scaled down from values typically found in servers (e.g. K_{OHD}=1024, C_{APP}=50, and K_{APP}=200). Use of such large buffer sizes and number of servers leads to state space explosion of the Markov chain models. Our focus in this paper has been to *qualitatively* reproduce the experimentally observed behaviour with a tractable model, hence the scaled-down values.

5. CONCLUSION

We presented an analytical model of a software server that explains the drastic drop in goodput as offered load exceeds system capacity. The model captures the 'flattening-out' of goodput as load is increased further. Our model specifically addresses the fact that various threads of the server compete with each other for access to the shared resource, leading to non-linear increase in response time in an overloaded state. A novel feature of this model is that the overhead involved in processing of a request is explicitly accounted for separately from the actual application processing. These factors, coupled with impatience in users, give results that produce the experimentally observed trend.

We also analyzed, in detail, the impact of using LIFO queueing discipline when the server is overloaded. We saw that the use of LIFO does, in fact, provide a higher goodput when user timeouts are not exponential. We further analyzed the difference in perfor-

mance between FIFO and LIFO disciplines by obtaining response time distributions. This analysis showed that even for exponential distributions, the response time characteristics of LIFO are better than those of FIFO.

In the present work, we concentrated on Web servers serving requests for dynamically generated content where the CPU becomes the bottleneck resource. However, this may be extended to other bottleneck resources and software servers with an appropriate choice of OHD, APP and resource sharing models.

The solution of our model, being based on repeatedly solving a multi-dimensional Markov chain, suffers from the state-space explosion problem that may limit its direct application as an online prediction tool. Nevertheless the model provides insights into the dynamics of an overloaded server and will aid in the analysis and development of overload control strategies.

Future work involves integration of more sub-models such as SIRO, priority-based preemptive scheduling for the OHD and APP parts and round robin, context-switching time sensitive models for resource sharing. We are also working on an application of such a model in self-tuned overload control mechanisms that suggest changes in configuration parameters to maintain high goodput during overload.

6. REFERENCES

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, January 2002.

[2] J. Carlstrom and R. Rom. Application-aware admission control and scheduling in web servers. In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, volume 2, pages 506–515, New York City, June 2002. IEEE.

[3] R. Carter and L. Cherkasova. Detecting timed-out client requests for avoiding livelock and improving web server performance. In *Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*, pages 2 – 7, Antibes-Juan les Pins, July 2000.

[4] H. Chen and P. Mohapatra. Session-based overload control in QoS-aware Web servers. In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 516–524, New York City, June 2002. IEEE.

[5] H. Chen and P. Mohapatra. Overload control in QoS-aware Web servers. *Computer Networks*, 42(1):119–133, 2003.

[6] L. Cherkasova and P. Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers*, 51(6):669–685, June 2002.

[7] C.Hirel, R. Sahner, X. Zang, and K. Trivedi. Reliability and performability modeling using SHARPE 2000. In *11th International Conference, TOOLS 2000*, Schaumburg, Illinois USA, March 2000.

[8] A. C. Dalal and S. Jordan. Improving user-perceived performance at a World Wide Web server. In *Proceedings of Global Telecommunications Conference (GLOBECOM)*, pages 2465–2469, San Antonio, Texas, September 2001.

[9] B. T. Doshi and H. Heffes. Overload performance of several processor queuing disciplines for the M/M/1 queue. *IEEE Trans. on Communications*, 34(6):538–546, June 1986.

[10] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. A method for transparent admission control and request scheduling in E-commerce Web sites. In *Proceedings of the 13th international conference on World Wide Web*, pages 276–286. ACM Press, 2004.

[11] J. S. Kaufman and A. Kumar. Traffic overload control in a fully distributed switching environment. In *Teletraffic Sciences for New Cost Effective Systems*, North Holland, 1989.

[12] L. Kleinrock. *Queueing Systems*, volume I: Theory. Wiley-Interscience, New York, 1975.

[13] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proceedings of the 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, pages 200–204. Springer, April 2002.

[14] B. Melamed and M. Yadin. Numerical computation of sojourn-time distributions in queuing networks. *Journal of the ACM*, 31(4):839–854, 1984.

[15] A. Movaghar. On queueing with customer impatience until the beginning of service. In *Proceedings of the 2nd International Computer Performance and Dependability Symposium*, page 150, Washington, DC, USA, 1996. IEEE Computer Society.

[16] A. Movaghar. On queuing with customer impatience until the end of service. In *Proceedings of the 4th International Computer Performance and Dependability Symposium*, page 167, Washington, DC, USA, 2000. IEEE Computer Society.

[17] V. Pla, V. Casares-Giner, and J. Martínez. On a multiserver finite buffer queue with impatient customers. In *Proceedings of the 16th ITC Specialist Seminar on Performance Evaluation of Wireless and Mobile Systems*, Antwerp, Belgium, August-September 2004.

[18] P. Reeser and R. Hariharan. An analytic model of web servers in distributed computing environments. *Telecommunication Systems*, 21(2):283–299, 2002.

[19] J. A. Rolia and K. C. Sevcik. The method of layers. *IEEE Trans. on Soft. Engg.*, 21(8):689–700, Aug 1995.

[20] M. Schwartz. *Telecommunication Networks, Protocols, Modeling and Analysis*, chapter 11, pages 606–608. Pearson Education, 1987.

[21] N. Singhmar, V. Mathur, V. Apte, and D. Manjunath. A combined LIFO-priority scheme for overload control of E-commerce Web servers. In *the International Infrastructure Survivability Workshop (affiliated with the 25th IEEE International Real-Time Systems Symposium)*, Lisbon, Portugal, December 2004. arXiv:cs.PF/0611087.

[22] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons Inc., New York, second edition, 2002.

[23] T. Voigt and P. Gunningberg. Adaptive resource-based web server admission control. In *Proceedings of the Seventh International Symposium on Computers and Communications (ISCC 2002)*, page 219, Washington, DC, USA, 2002. IEEE Computer Society.