A new blocking problem from Java-based schedulers

Peter G. Harrison and Catalina M. Lladó

Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ, {pgh,cl16}@doc.ic.ac.uk

Abstract

We consider a form of blocking, which is typical in client-server systems including those implemented under the Enterprise JavaBean (EJB) specification. The novel feature is that tasks must wait for one of a number of parallel queues to clear its outstanding work. Thus, blocking time is the minimum of sojourn times at the parallel queues. Under certain simplifying assumptions, we solve this model for the probability distribution of blocking time and obtain a simple formula for its mean value. We then use this result in an aggregate server model of a larger queueing network in which further non-standard techniques are included to represent this form of blocking. We compare our approximate results against simulation data, obtaining good agreement for both system throughput and queue length probability distributions at equilibrium.

Keywords: Markov chains; Queueing networks; Blocking; Enterprise JavaBeans

1 Introduction

Distributed client-server systems are now so complex that they cannot be designed efficiently without the use of some form of quantitative (performance) model; see [10] for example. Simulation can sometimes be used for this purpose but analytical models are preferred where possible in view of their greater efficiency and flexibility for experimentation. We develop an approximate analytical model for the central scheduler in a distributed, three-tier, client-server architecture, typical of large, Java-supported, Internet applications. In particular, we focus on a new form of blocking in which tasks must wait for one of a number of parallel queues to clear its outstanding work. Thus, blocking time is the minimum of sojourn times at the parallel queues. We solve this model for the equilibrium probability distribution of blocking time and obtain a simple formula for its mean value. We then use this result in an aggregate server model of a larger queueing network wherein further non-standard techniques

are applied. Validation is with respect to simulation, comparing both system throughput and queue length probability distributions at equilibrium.

The rest of this paper is organised as follows. Section 2 summarises the real system under study – the Enterprise JavaBeans (EJB) architecture – while section 3 describes the analytical methods used and specifies the resulting models utilised in the case study: an EJB server implementation used as a scheduler for distributed object oriented systems. Section 4 validates the analytical results against simulation for this scheduling system. The paper concludes in section 5, where future research directions are also discussed.

2 Enterprise JavaBeans Architecture

EJB technology [8] defines a model for the development and deployment of reusable Java server components. A bean instance (or component) is created and managed at runtime by a container instance (from now on simply referred to as a container), so that a task can only access a bean instance through its container.

EJB defines two kinds of components: *session* beans and *entity* beans. While *session* beans are lightweight, relatively short lived, do not survive server crashes and execute on behalf of a single client, *entity* beans are robust, expected to exist for a considerable amount of time, do survive server crashes and are shared between different users.

A *bean implementation* and two interfaces, the *bean interface* and the *home interface*, define a bean class. A client accesses a bean through the bean interface, which defines the business methods that are callable by clients. The home interface allows the client to create and, in case of entity beans, look up a bean. From the bean implementation and the interfaces, two new classes are automatically generated using the container facilities: a bean class that will wrap the actual bean (*Container Generated Bean* or CGBean class) and a home class that allows a user to create a bean (*Container Generated Home* or CGHome class). Fig. 1 shows the class structure.

In the EJB implementation under study, each container is responsible for managing one EJB. Thus, there is one container for each bean class. Fig. 2 shows how a client uses different bean instances of the same bean class.

Multiple clients can access an entity instance concurrently. In this case, the container synchronises its access. Each client uses a different instance of the CGBean class that interfaces the client with the container but all the clients share the same bean instance. An entity container has a limited number of

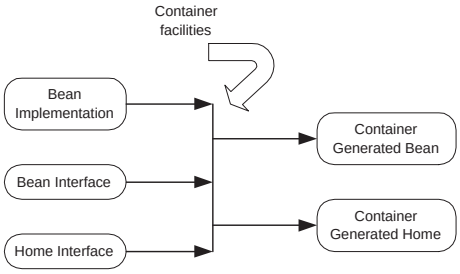

Fig. 1. Class generation using container facilities.

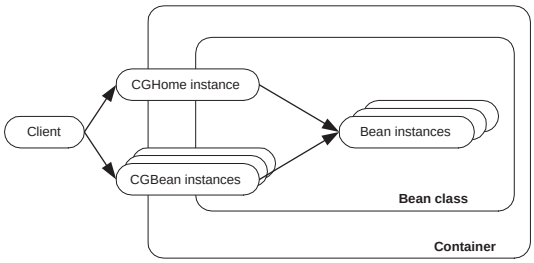

Fig. 2. Client use of several bean instances.

bean instances that can exist concurrently, i.e. instances in main memory ready to be accessed or pool of active instances. The execution of a method in a bean instance requires this instance to be active. Consequently blocking can arise when this particular instance is not active and the current number of active instances is equal to the maximum.

Since the most common operation over a bean instance is a method execution, we consider its behaviour for the case of entity beans. A detailed description of the execution characteristics and the corresponding queueing network abstraction can be found in [5]. As far as this paper is concerned, we concentrate on the blocking subsystem, which consists of an access to the container server followed by an access to the instance server required. The modelling of this subsystem is described in the following section.

3 A Queueing Network Model with Blocking

The queueing network model that represents the blocking subsytem is shown in Fig. 3. It consists of $M + 1$ stations where 1 corresponds to the outer (container) server and M is the number of parallel (instance) servers, i.e. the maximum number of bean instances that can be active at the same time. A closed, 'short-circuited' network is considered with a view to employing a Flow Equivalent Server (FES) in a larger model, using this network's throughput function at each population size [1].

Blocking is the critical non-standard characteristic in this network; a client

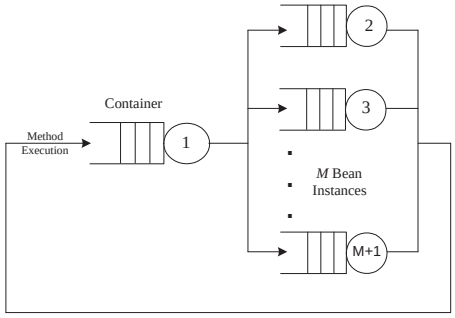

Fig. 3. Queueing network with blocking.

that has completed service in the container station is blocked if the required bean instance is not active and there is no free instance to passivate [1] – i.e. no idle server in the model. The client will be blocked at the container station until one of the M parallel servers becomes idle. As a consequence, the blocking time needs to be calculated. In conventional blocking models, see for example [9], blocking time is normally equal to the residual service time at some downstream server. Here, however, it is the time required for the first of the M parallel servers to clear its queue in a blocking-after-service discipline.

Table 1 summarises the notation used for the different parameters of the model.

3.1 Blocking Time

Given that a customer is blocked at the container server, so that all the instance servers have non-empty queues, the blocking time B is the minimum of the times for each of the M queues to become empty, with no further arrivals during the time B. Let the random variable T_i denote the time for queue i to become empty, given that initially (i.e. when the customer is first blocked) there are $n_i \geq 1$ other customers in queue i, including the one in service ($1 \leq i \leq M$). Then the probability distribution function, $F_{n_i}(t)$, of T_i is Erlang-n_i with parameter μ_i, i.e.

$$F_{n_i}(t) = P_{n_i}(T_i \leq t) = 1 - \sum_{k=0}^{n_i-1} \frac{(\mu_i t)^k}{k!} e^{-\mu_i t} \qquad (n_i > 0) \ . \tag{1}$$

Now, the conditional (on queue lengths $\mathbf{n} = (n_1, \ldots, n_M)$) blocking time $T_{\mathbf{n}} = \min_{1 \leq i \leq M} T_i$ has complementary cumulative probability distribution function

[1] The term *passivate* is used according to the terminology in the EJB-1.1 specification [8].

Table 1
Notation for the model parameters.

C	Containers or Bean classes
M	Parallel servers
I	Different instances
N	Population
m_1	Mean service time for outer server
μ	Service rate for each of the parallel servers
j	Clients at the outer server
k	Clients at the aggregated server
$\mu_1(j)$	Service rate for outer server with queue length j
$\mu_2(k)$	Service rate for aggregated server with queue length k
$b(k)$	Mean blocking time when there are k clients at the parallel servers
β_k	Blocking probability when there are k clients at the parallel servers
π_{kn}	Probability that n out of M parallel servers are busy, when there are k clients at the parallel servers
α	Probability that a task completing service at the outer server is not blocked, given that all the parallel servers are busy
z_k	Probability that when a task completes service at the outer server with k tasks at the parallel servers, at least one of these servers is idle
$p(j)$	Steady state probability distribution of queue length at the outer server

$$P_{\mathbf{n}}(T_{\mathbf{n}} > t) = \prod_{i=1}^{M} P(T_i > t) = \prod_{i=1}^{M} \sum_{k=0}^{n_i-1} \frac{(\mu_i t)^k}{k!} e^{-\mu_i t} \tag{2}$$

since the T_i are independent. Hence the unconditional blocking time has distribution function $B(t)$ given by

$$\overline{B}(t) \equiv 1 - B(t) = \sum_{\mathbf{n}:\forall i.n_i>0} \pi(\mathbf{n}) \prod_{i=1}^{M} \sum_{k=0}^{n_i-1} \frac{(\mu_i t)^k}{k!} e^{-\mu_i t} \tag{3}$$

where $\pi(\mathbf{n})$ is the equilibrium probability that there are n_i customers in queue i at the instant a customer becomes blocked at the container server. We make the approximating assumption that all valid states $\mathbf{n}$ (i.e. with $n_i > 0$ for all i) have the same equilibrium probability. This property would hold were we to have the job observer property in a decomposable, Markovian network with equally utilised servers $1, \ldots, M$, e.g. in the case that there was no blocking

and we were considering departure instants from the container server. With this assumption, when the population of the servers $1, \ldots, M$ is K, i.e. when $\sum n_i = K$, for all $\mathbf{n}$, $\pi(\mathbf{n})$ is the reciprocal of the number of ways of putting $K - M$ balls into M bags — at least one must be in each bag. In other words, for all $\mathbf{n}$,

$$\pi(\mathbf{n}) = \pi = \frac{(K-M)!(M-1)!}{(K-1)!} \ . \tag{4}$$

We now have the following result for the probability distribution of blocking time.

Theorem 1 *With the assumptions stated above, the probability distribution of the blocking delay suffered by a blocked customer at equilibrium is*

$$B(t) = 1 - \frac{(K-M)!}{(K-1)!} e^{-M\overline{\mu}t} \sum_{k=0}^{K-M} \frac{(K-k-1)!}{(K-M-k)!k!} \left(M\overline{\mu}t\right)^k \tag{5}$$

when there are K customers at the M parallel servers, where $\overline{\mu}$ is the average service rate of the M parallel servers, i.e. $M\overline{\mu} = \sum \mu_i$.

Proof. When there are K customers in queues $1, \ldots, M$,

$$\begin{aligned} \overline{B}(t) &= \sum_{\substack{\mathbf{n}: \\ \forall i. n_i > 0, \sum n_i = K}} \pi \sum_{\substack{\mathbf{k}: \\ \forall i. 0 \le k_i \le n_i - 1}} \prod_{i=1}^{M} \frac{(\mu_i t)^{k_i}}{k_i!} e^{-\mu_i t} \\ &= \pi \sum_{\substack{\mathbf{k}: \\ \forall i. k_i \ge 0, \sum k_i \le K-M}} \sum_{\substack{\mathbf{n}: \\ \forall i. n_i \ge k_i + 1, \sum n_i = K}} \prod_{i=1}^{M} \frac{(\mu_i t)^{k_i}}{k_i!} e^{-\mu_i t} \end{aligned} \tag{6}$$

by changing the order of summation. The inner summand is independent of $\mathbf{n}$ and the number of terms is equal to the number of ways of putting $K - M - k$ balls into M bags, when $k = \sum_{i=1}^{M} k_i$. (The reasoning here is that $k_i + 1$ at least must be in each bag, using up $k + M$ of the K balls in total.)

Consequently, we may write:

$$\begin{aligned} \overline{B}(t) &= \frac{(K-M)!(M-1)!}{(K-1)!} \times \\ &\qquad \sum_{\substack{\mathbf{k}: \\ \forall i. k_i \ge 0, \sum k_i \le K-M}} \frac{(K - \sum k_i - 1)!}{(M-1)!(K - M - \sum k_i)!} \prod_{i=1}^{M} \frac{(\mu_i t)^{k_i}}{k_i!} e^{-\mu_i t} \\ &= \frac{(K-M)!}{(K-1)!} e^{-M\overline{\mu}t} \sum_{k=0}^{K-M} \frac{(K-k-1)!}{(K-M-k)!k!} \sum_{\substack{\mathbf{k}: \\ \forall i. k_i \ge 0, \sum k_i = k}} k! \prod_{i=1}^{M} \frac{(\mu_i t)^{k_i}}{k_i!} \end{aligned}$$

$$= \frac{(K-M)!}{(K-1)!} e^{-M\overline{\mu}t} \sum_{k=0}^{K-M} \frac{(K-k-1)!}{(K-M-k)!k!} (M\overline{\mu}t)^k \tag{7}$$

and the result follows. □

We assume that all the M servers are equally likely to be chosen by each arrival and that their service rates are all equal to μ — this guarantees equal utilisations. We then have

$$\overline{B}(t) = \frac{(K-M)!}{(K-1)!} e^{-M\mu t} \sum_{k=0}^{K-M} \frac{(K-k-1)!}{(K-M-k)!k!} (M\mu t)^k \quad . \tag{8}$$

Mean blocking time, $b(K)$ say, which we use in the equilibrium model for the queue lengths for the whole system below, now follows straightforwardly.

Corollary 2 *Mean blocking time for blocked customers at equilibrium is $K/(M^2\overline{\mu})$ when there are K customers at the M parallel servers.*

Proof.

$$b(K) = \int_0^\infty \overline{B}(t)\mathrm{d}t \tag{9}$$
$$= \frac{(K-M)!}{(K-1)!} \sum_{k=0}^{K-M} \frac{(K-k-1)!}{(K-M-k)!k!} \int_0^\infty (M\overline{\mu}t)^k \, e^{-M\overline{\mu}t} \mathrm{d}t$$

Changing the integration variable from t to $s/M\overline{\mu}$ then yields

$$b(K) = \frac{(K-M)!}{M\overline{\mu}(K-1)!} \sum_{k=0}^{K-M} \frac{(K-k-1)!}{(K-M-k)!k!} \int_0^\infty s^k e^{-s} \mathrm{d}s$$
$$= \frac{(K-M)!}{M\overline{\mu}(K-1)!} \sum_{k=0}^{K-M} \frac{(K-k-1)!}{(K-M-k)!}$$
$$= \frac{(K-M)!}{M\overline{\mu}(K-1)!} \sum_{k=0}^{K-M} \frac{(k+M-1)!}{k!} \tag{10}$$

(by changing the summation variable to $K-M-k$.)

We complete the proof by showing by induction on N that

$$\sum_{k=0}^{N} \frac{(k+m)!}{k!} = \frac{(N+m+1)!}{(m+1)N!} \tag{11}$$

for $N, m \geq 0$. For $N = 0$, both sides are $m!$. Now assume that

$$\sum_{k=0}^{n} \frac{(k+m)!}{k!} = \frac{(n+m+1)!}{(m+1)n!} \tag{12}$$

for $n \geq 0$. Then we have

$$\begin{aligned} \sum_{k=0}^{n+1} \frac{(k+m)!}{k!} &= \frac{(n+m+1)!}{(m+1)n!} + \frac{(n+m+1)!}{(n+1)!} \\ &= \frac{(n+m+1)!}{(m+1)(n+1)!}(n+m+2) \end{aligned} \tag{13}$$

as required. Substituting $N = K - M$ and $m = M - 1$ now yields

$$b(K) = \frac{(K-M)!}{M\overline{\mu}(K-1)!}\frac{K!}{M(K-M)!} = \frac{K}{M^2\overline{\mu}} \tag{14}$$

which completes the proof. □

3.2 The Aggregated Server

The model of a subsystem comprising a container and its M instance servers, which are aggregated into a single node, at constant population N is shown in Fig. 4. The throughput function (as a function of N), once validated, can be used to parameterise the service rate of a FES node for the subsystem in an aggregated model of a larger system [5].

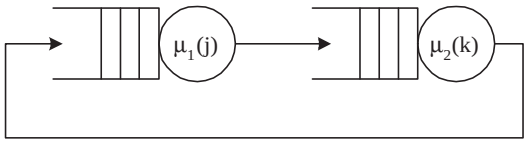

Fig. 4. Aggregated Model.

When there are j clients at the outer (container) server and $k = N - j$ at the M parallel (instance) servers, the service rate functions $\mu_1(j)$ (with blocking) and $\mu_2(k)$, are estimated as follows:

$$\mu_1(j) = \begin{cases} 1/(m_1 + \beta_{N-j} b(N-j)) & \text{if } (N-j) \geq M \\ 1/m_1 & \text{otherwise} \end{cases} \tag{15}$$

where m_1 is the mean service time of the outer server when there is no blocking, $b(N-j)$ is the mean blocking time when there are j customers at the outer server ($N - j$ customers at the parallel servers) and β_{N-j} is the equilibrium blocking probability, which we derive in the next subsection. The intuition

behind this formula is that the mean service time is m_1 when there is no blocking and $m_1 + b(N-j)$ when the queue length is j and the front task becomes blocked. Hence the overall average service time is (for $M \leq N-j$) $m_1 + \beta_{N-j} b(N-j)$.

For the second server we have

$$\mu_2(k) = \sum_{n=1}^{M} \pi_{kn} n\mu \tag{16}$$

where μ is the service rate of each of the parallel servers. The parameter π_{kn} is the probability that n out of the M servers are busy, given that there are k customers at the parallel servers altogether; it is also derived in the next subsection.

Clearly the visitation rate is the same for both servers. The steady state probability distribution for this network – $p(j)$ for the state with j tasks at the outer server and $N-j$ at the parallel servers – can then be approximated as a product form in standard fashion (see, for example, [1]). However, we improve considerably on this approximation in section 3.4 by using a simple, explicit Markov model. Throughput $T(N)$ is now given by:

$$T(N) = \sum_{j=1}^{N} p(j)\mu_1(j) \ . \tag{17}$$

3.3 Instance-Active and Client-Blocking Probabilities

Let z_k denote the equilibrium probability that when there are k tasks at the M parallel servers, at least one of these servers is idle. Let α denote the probability that a task completing service at the container server is not blocked, given that all the parallel servers are busy. This parameter can be estimated from the method-calling profile of tasks, for example $\alpha = M/I$ in the simplest, uniform case where $I \geq M$ is the number of different instances supported in the container. Then the dynamic blocking probability for a task completing service at the outer (container) server when there are k tasks at the parallel servers is given by:

$$\beta_k = \frac{(1-z_k)(1-\alpha)\frac{m_1}{m_1+\beta_k b(k)}}{z_k + (1-z_k)\frac{m_1}{m_1+\beta_k b(k)}} \tag{18}$$

This quantity is the ratio of the equilibrium probability flux from unblocked states with k tasks at the parallel servers to a blocked state, over the total flux from unblocked states (see [3], for example). The fraction $\frac{m_1}{m_1+\beta_k b(k)}$ is

the long-term proportion of time that tasks at the container server undergo normal service, rather than are blocked. From Eq. (18), a quadratic equation in β_k, β_k can be expressed as follows:

$$\beta_k = \frac{\sqrt{m_1^2 + 4b(k)z_k(1-z_k)(1-\alpha)m_1} - m_1}{2b(k)z_k} \tag{19}$$

The parameter z_k can be estimated by considering a simple two dimensional Markov chain (K_t, Ξ_t) where, at time t, K_t represents the number of tasks in total at the M parallel servers and Ξ_t the number of non-empty queues, i.e. busy servers, out of the M. However, we use an even simpler submodel to estimate z_k: an M-state Markov chain Ξ_t^k for each population size $k \geq M$ at the parallel servers, where each state corresponds to the number of busy queues in the system (see Fig 5). Clearly this is an approximate decomposition since K_t and Ξ_t are not independent, but it does capture the essence of the blocking that is present.

This predicts a relatively low probability of empty queues (when $k \geq M$ and $I \geq M$) since it will often be the case that a task is blocked when a server with only one task completes a service period, resulting in an empty queue which is instantaneously occupied by the unblocked task; the empty state is therefore not seen.

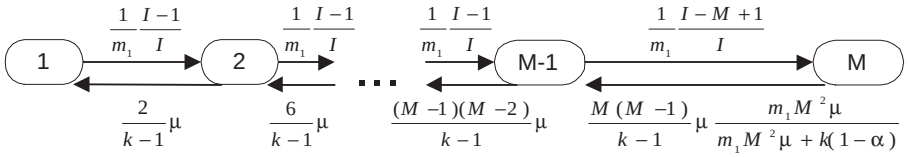

Fig. 5. 1-D Markov chain for each population k.

For population size $k \geq M$ at the parallel servers, let the equilibrium probability that $\Xi_\infty^k = l$ $(l = 1...M)$ be denoted by $\pi_k(l)$.

The submodel then has balance equations:

$$\pi_k(l)(m_1)^{-1}\frac{I-l}{I} = \pi_k(l+1)(l+1)\mu\frac{l}{k-1} \tag{20}$$

for $1 \leq l \leq M-2$. The fraction on the left hand side represents the probability of a task not choosing an instance at one of the l busy servers. The fraction on the right hand side is the probability that any given non-empty queue has length exactly one, when there are $l+1$ busy servers with k tasks in total. We estimate it as the ratio of the number of arrangements of $k-l-1$ tasks in l queues, to the number of arrangements of $k-l-1$ tasks in $l+1$ queues; after assigning one task to each queue, there are only $k-l-1$ left. Since the number of arrangements of n tasks in m queues is $\frac{n+m-1)!}{n!(m-1)!}$, the required ratio is $\frac{l}{k-1}$.

The final balance equation, for $l = M - 1$, is

$$\pi_k(M-1)(m_1)^{-1}\frac{I-M+1}{I} = \pi_k(M)v_k M\mu\frac{M-1}{k-1} \tag{21}$$

where v_k is the probability of observing, just before a departure instant from the all-busy parallel servers when there are k tasks there in total, a task at the front of the container queue in a non-blocked state – i.e. receiving service with mean time m_1 (when all of the parallel servers are busy). Hence we estimate v_k by the ratio of mean container service time to the sum of this time and the mean time blocked when all instance servers are busy, appealing to the law of large numbers for Markov chains – cf. the estimation of β_k above. The time spent blocked is 0 if the required instance is active (with probability α) and $b(k)$ otherwise. Thus we define

$$v_k = \frac{m_1 M^2\mu}{m_1 M^2\mu + (1-\alpha)k} \tag{22}$$

The recursive function for the *unnormalised* probabilities $\hat{\pi}_k(n)$ derived from the balance equations (20) can be written,

$$\hat{\pi}_k(n) = \begin{cases} 1 & \text{if } n = 1 \\ \frac{(I-n+1)(k-1)}{n(n-1)m_1\mu I}\hat{\pi}_k(n-1) & \text{if } 2 \le n < M \\ \frac{(I-n+1)(k-1)(m_1M^2\mu+(1-\alpha)k)}{M^2m_1^2\mu^2 In(n-1)}\hat{\pi}_k(n-1) & \text{if } n = M \end{cases} \tag{23}$$

Normalising the $\hat{\pi}_k(n)$ to give the probabilities $\pi_{kn} = \hat{\pi}_k(n)/\sum_{l=1}^{M}\hat{\pi}_k(l)$, we now estimate z_k by $1 - \pi_{kM}$ and β_k follows for $M \le k < N$. The same applies to $\mu_2(k)$, but when there is no blocking, i.e. when $k \le M$ or $k \le I$, $\mu_2(k) = \frac{kI\mu}{k+I-1}$ by usual multiple parallel server result [1].

This approximation is validated against the 2-Dimensional Markov chain referred to above; see Fig. 8.

3.4 Container-server sub-model

Representation of the container server as a single Markovian queue, specified solely by its service rate function, is inadequate. This would require negative exponential service times, which is a poor approximation in view of the high variability in service times; tasks either have short service times (mean value m_1) if they are not blocked or very long ones otherwise. Therefore, we consider the model shown in Fig. 6 where the outer server is split into two phases. Phase 1 comprises the normal service time at the container server (mean value

m_1) and Phase 2 comprises the blocking time (with state dependent mean as derived in section 3.1). Consequently, when a customer finishes Phase 1 it goes to Phase 2 with probability β_k and goes to the aggregated node for the parallel servers with probability $1 - \beta_k$, when there are k tasks at the parallel servers. Clearly only one of the phases is busy at a time.

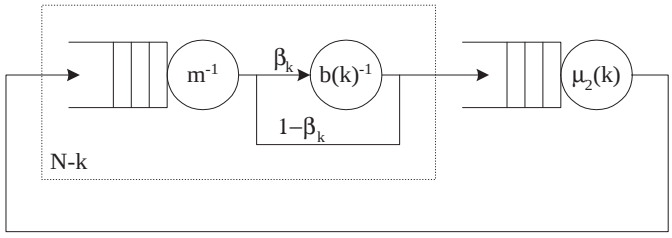

Fig. 6. Model with a 2-Phase outer server.

The Markov chain (K_t, B_t) derived from this model can be seen in Fig. 7, where K_t is the number of tasks at the parallel servers and $B_t = 0$ (respectively, 1) if the task at the front of the container queue is blocked (respectively, not blocked), at time t. For $K_t = N$, $B_t = 1$ with probability 1. The parameter $b(k) = \frac{k}{M^2\mu}$ (see section 3.1), μ_2 is as in Eq. 16 and β_k is as defined in section 3.3. This chain is finite and irreducible and so has a steady state from which the equilibrium queue length probability p_j is computed as $P(K_\infty = N - j, B_\infty = 0) + P(K_\infty = N - j, B_\infty = 1)$.

4 Numerical Validation

Since it is only possible to analytically model complex distributed systems using several simplifications and assumptions, approximation techniques have to be – and have been – used. Consequently, theoretical results need to be validated by comparing them with those obtained from simulation. A simulation program was written using QNAP2 V. 9.3, a modelling language developed by INRIA [2] . Amongst other facilities, this language provides a discrete-event simulator. The simulations were run for 500,000 time units, using the spectral method, and the resulting confidence intervals had an error of less than 3% at a 95% level of confidence. In addition to simulation, validation against a 2-dimensional Markov chain that represents the behaviour of the whole system more faithfully has been carried out. Subsection 4.1 describes this model and how it has been solved while subsection 4.2 discusses the validation of the analytical model's predictions against simulation.

[2] *Institut Nacional de Reserche en Informatique et Automatique*

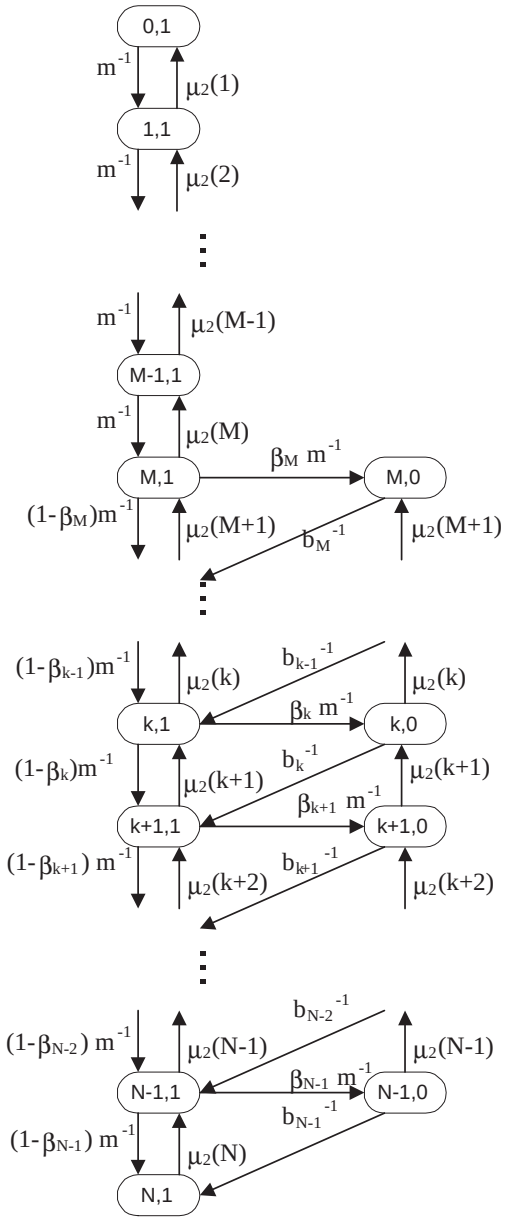

Fig. 7. Markov chain for the container server sub-model.

4.1 Explicit 2-D Markov Chain

The Markov chain depicted by Fig. 8 represents in finer detail the behaviour of the queueing network under study and has been solved numerically using Knottenbelt's Markov chain analyser [4]. It describes the joint evolution of the number of busy parallel servers and the number of tasks at those parallel servers. The solution of this Markov chain is very expensive when N and/or M are reasonably big; therefore it is only used for validation – see subsection 4.2. As stated in subsection 3.3 this solution is used to validate the approximation for the value of the parameter z_k. Unless this Markov chain model agrees well

with the simulation, the approximate model of section 3 cannot be adequate.

In the Markov chain, each row (apart from the top one) contains all the states with the same number of busy servers, while each column contains all the states with the same number of clients at the parallel servers. The top row represents all states with blocking, i.e. those in which there is a client blocked in the outer server waiting for one of the parallel servers to become empty. In this Marcov chain $\lambda = 1/m_1$ and all the other parameters are as defined in section 3.

Since the state space is finite and the chain is irreducible (as can be seen by inspection) a steady state solution for the state probabilities always exists.

4.2 Numerical Results

Numerical results of the analytical model's predicted throughput and queue length probability distribution are validated against simulation in subsections 4.2.1 and 4.2.2. Moreover, graphs for the validation of the specific parameter values for blocking time and z_k (probability of having at least one idle server) are shown and discussed in subsections 4.2.3 and 4.2.4.

4.2.1 Throughput

Graphs of the throughput predicted by the analytical and simulation models for different values of N and M are shown in Fig. 9. Table 2 tabulates the corresponding numerical values and the difference between simulated and analytical predictions. The parameters $m_1 = 0.4$ and $\mu = 1/4.1$. These graphs show excellent agreement between the analytical model and the simulation, the error being less than 1% at all points. Given the approach taken in constructing the analytical model, based on aggregated service rates, this is not surprising.

Table 2
Throughput comparison between simulation and analytical models.

M=6, I=20				N=30, I=40			
N	*Simulation*	*Analytical*	*Error*	*M*	*Simulation*	*Analytical*	*Error*
5	0.9283	0.9263	-0.0002	3	0.7112	0.7190	-0.0078
10	1.3530	1.3538	0.0008	6	1.3710	1.3785	-0.0075
15	1.3760	1.3721	-0.0039	9	1.9370	1.9116	0.0254
20	1.3760	1.3734	-0.0026	12	2.3230	2.2695	0.0535
25	1.3770	1.3735	-0.0035	15	2.4750	2.4476	0.0274
30	1.3770	1.3735	-0.0035	18	2.4960	2.4901	0.0058

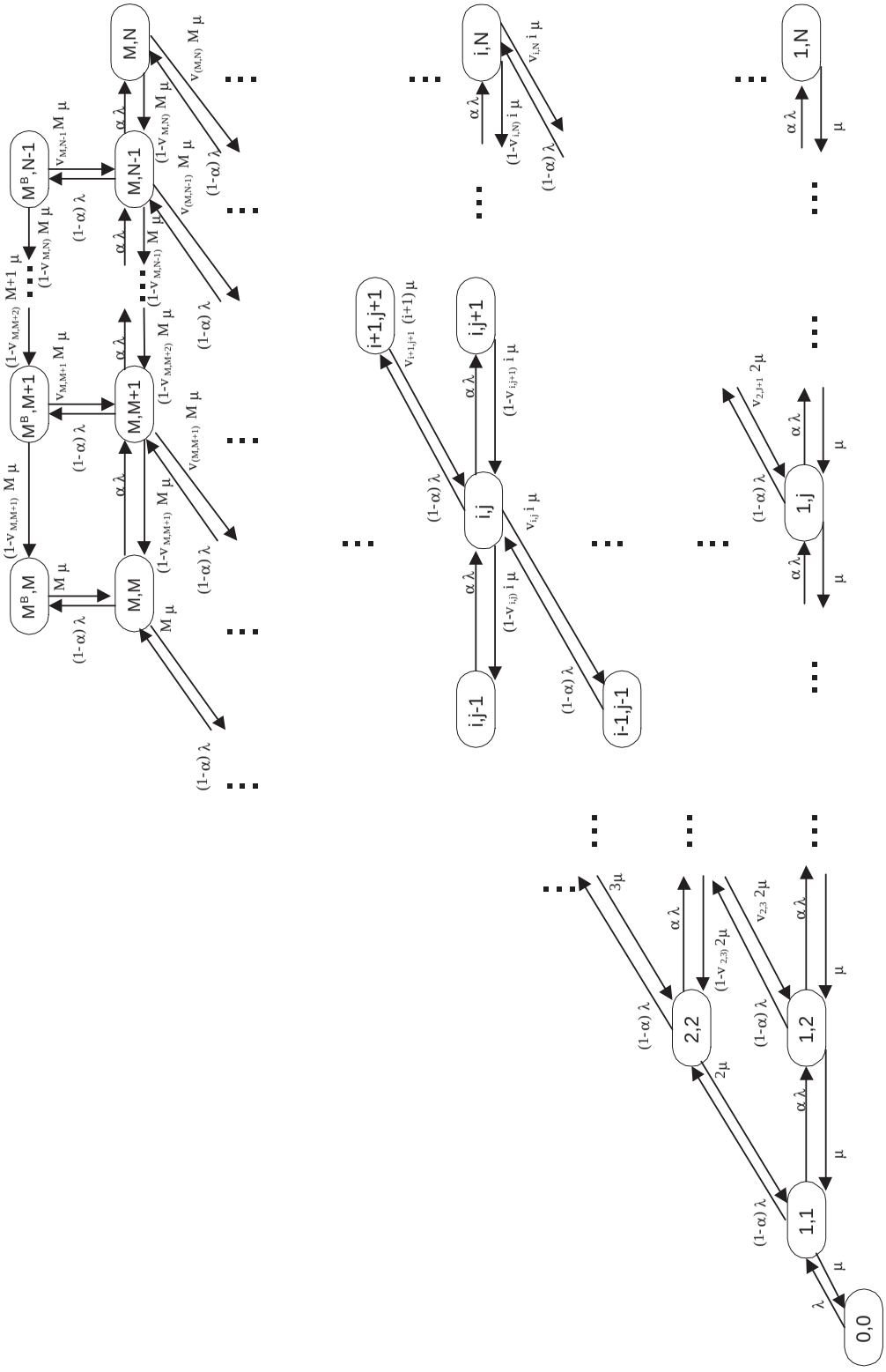

Fig. 8. 2-dimensional detailed Markov chain.

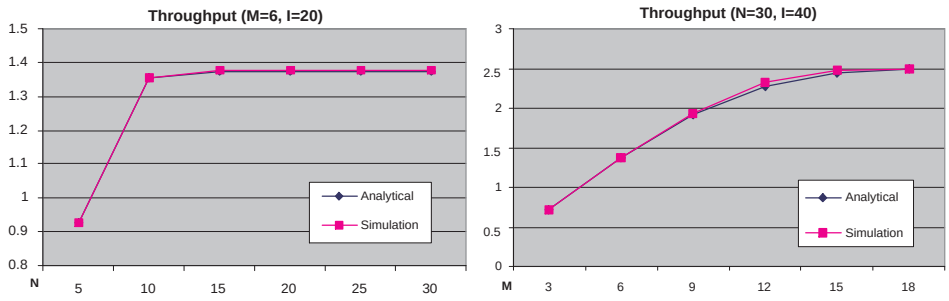

Fig. 9. Throughput comparison for simulated and analytical results for $M = 6$, $I = 20$ and $N : 5 - 30$ (left) and $N = 30$, $I = 40$ and $M : 3 - 18$ (right).

4.2.2 Queue Length Probability Distribution

Graphs of the container server's queue length probability distribution for different combinations of N and M are shown in Fig. 10. The simulation reveals a fairly sharp peak at values of k just over M (j just below $N - M$). This is due to the onset of blocking. When $k < M$, there is no blocking and the outer server is very fast relative to the aggregated server; hence the queue length probability is small but increasing (as the aggregated server gets faster). As soon as blocking occurs, increasingly as $k = M, M + 1, \ldots$, the outer server becomes the slower and so its queue length increases. This build-up is rapid since, when $k \geq M$, the probability of empty queues at the parallel server is reduced compared to a FES submodel in which all state vectors for the M queues have the same equilibrium probability. This is a consequence of the blocking mechanism; whenever all the instance servers are busy, there is a significant probability (viz. $1 - \alpha$) that the next arrival will be blocked and so immediately replace a departing task that leaves an empty queue behind.

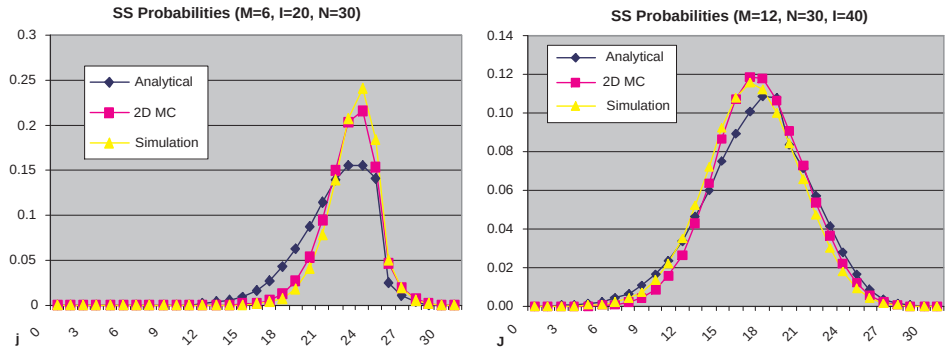

Fig. 10. Steady State Probabilities comparison for simulated, analytical and 2-D Markov chain results for $M = 6$, $I = 20$ and $N = 30$ (left) and $M = 12$, $I = 40$ and $N = 30$ (right).

A previous analytical model, which assumed exponential service times in a product form solution, predicts this peak poorly, even though the blocking mechanism was represented in detail through estimates for α, z_k, β_k and hence $\mu_2(k)$. However, the variability of service times at the container server, when

blocking is possible, was poorly modelled since the server was characterised solely by its rate, i.e. its *mean* service time. The present model captures this variability explicitly and produces dramatically better predictions.

As expected, the values obtained in the numerical solution of the 2-dimensional Markov chain stay in between the simulated values and the analytical values. Our analytical model uses the same assumption as the Markov chain model (same probability distribution for the different client-configurations in the non-idle parallel servers) as well as some other approximations. Consequently, its predictions should not be any better than the ones obtained through the Markov chain.

4.2.3 Blocking Time

Fig. 11 shows two graphs comparing the mean blocking time values predicted by the formula (Corollary 2) with those obtained through simulation, for different values of M, I and N. These graphs show very good agreement between the analytical and simulation models, the error being less than 3% at all points. This supports the assumptions made to obtain the blocking time probability distribution function and consequently the mean blocking time formula.

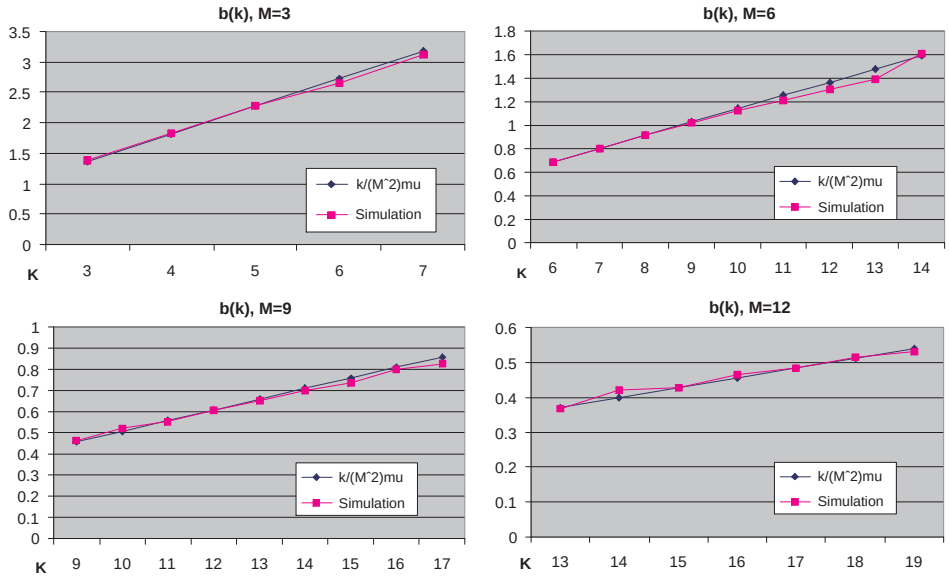

Fig. 11. Blocking time comparison for simulated and analytical results for $M = 3, 6, 9$ and 12.

4.2.4 Probability of having at least one idle server, z_k

Fig. 12 shows graphs comparing the results for z_k obtained through simulation with those obtained using the analytical models. Concretely, two different

Markov chains have been used to obtain z_k: the 2-D Markov chain that represents the behaviour of the system in detail (number of busy instance servers and number of clients there) and the simpler 1-D Markov chain which is the one that is actually used in the analytical model.

Again, the estimate of z_k validates well, if not as well as the mean blocking time $b(k)$ above. The 2-D chain matches the simulation the closer, as expected, including the increase in z_k as k approaches N. The only significant error in the 1-D model's predictions is at larger values of k which correspond to states which occur with very small probability. Hence the overall effect of any such discrepancy should be very small.

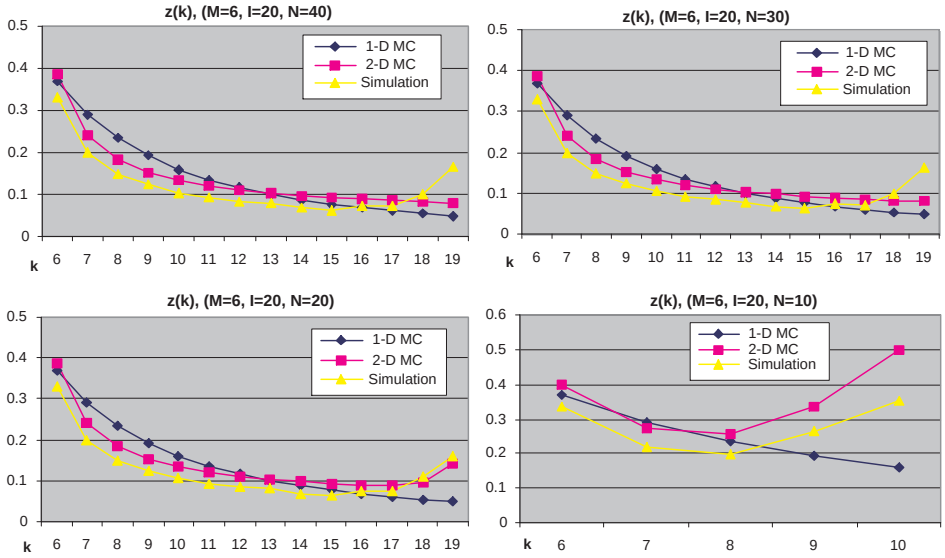

Fig. 12. z_k comparison for simulated and analytical (1-D Markov chain and 2-D Markov chain) results for $M = 6$, $I = 20$ and $N = 40, 30, 20, 10$.

5 Conclusion

We have developed an analytical, queueing-based model for the blocking characteristics in a scheduler in an object-oriented, EJB architecture implementation. In particular, we modelled the method execution call mechanism and were able to isolate the blocking that occurs when an instance is inactive and all servers are busy, yielding an approximate, two-server, aggregated model. This type of architecture has become common as a subsystem in large scale distributed systems.

Numerical results show excellent agreement with simulation, especially at large M, where contention is less. However, even at high contention, the error is less than 3%, suggesting a good building block for the modelling of complex,

distributed, object-based scheduling systems. It should be possible to integrate the model as a component in layered software/hardware models such as [11].

Immediate future work will focus on using intervals as input parameters of the model, based on the techniques of [6], to perform a sensitivity analysis. This study will demonstrate the extent of the influence of these model parameters in the results predicted for throughput and response time. The adaptation of the model to interval parameters has already been done together with some experimentation, see [7].

The Kensington Enterprise Data Mining System at Imperial College [2] is a real example of a distributed client-server architecture of the type analysed. Using it, we intend to collect real performance data to validate our models with respect to an operational environment, and hence use the models to guide the quantitative design of future generations of Kensington and other EJB-type systems.

Acknowledgements

The authors would like to thank William Knottenbelt for the advice and help given in using the Markov chain analyser and for his always useful comments.

References

[1] P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley, 1993.

[2] InforSense Ltd. Kensington 2000. http://Kensington.doc.ic.ac.uk.

[3] F.P. Kelly. *Reversibility and Stochastic Networks.* Wiley, 1979.

[4] W. Knottenbelt. *Parallel Performance Analysis of Large Marcov Models.* PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, 1999.

[5] C. M. Lladó and P. G. Harrison. Performance evaluation of an enterprise javabean server implementation. In *Proceedings* 2^{nd} *International Workshop on Software and Performance (WOSP 2000)*, pages 180–188, September 2000.

[6] J. Lüthi and G. Haring. Mean value analysis for queueing network models with intervals as input parameters. *Performance Evaluation*, 32(3):185–215, April 1998.

[7] J. Lüthi and C. M. Lladó. Interval parameters for capturing uncertainties in an EJB performance model. In *Proceedings ACM Sigmetrics 2001 / Performance 2001.* ACM, June 2001.

[8] Sun Microsystems. Enterprise JavaBeans 1.1 Architecture. http://java.sun.com/products/ejb, 1999.

[9] H.G. Perros and T. Altiok, editors. *Queuing Networks With Blocking.* North-Holland, 1989.

[10] C.U. Smith. *Performance Engineering of Software Systems.* Addison-Wesley, 1990.

[11] C.M Woodside. Performability modelling for multi-layered service systems. In *Third Int. Workshop on Performability of Computer and Communications Systems*, Sept 1996.

Vitae

Peter Harrison is currently a Professor of Computing Science at Imperial College, London where he became a lecturer in 1983. He graduated at Christ's College Cambridge as a Wrangler in Mathematics in 1972 and went on to gain Distinction in Part III of the Mathematical Tripos in 1973, winning the Mayhew prize for Applied Mathematics. He obtained his Ph.D. in Computing Science at Imperial College in 1979. He has researched into analytical performance modelling techniques and algebraic program transformation for some twenty years, visiting IBM Research Centers for two summers in the last decade. He has written two books, had over 100 research papers published in his research areas and held a series of research grants, both national and international. The results of his research have been exploited extensively in industry, forming an integral part of commercial products such as Metron's Athene Client-Server capacity planning tool. He has taught a range of subjects at undergraduate and graduate level, including Operating Systems: Theory and Practice, Functional Programming, Parallel Algorithms and Performance Analysis.

Catalina M. Lladó studied *Enginyeria Informàtica* at the *Universitat de les Illes Balears*. She is currently completing her Ph.D. research program at Imperial College, London. Her research interests include performance evaluation of distributed and concurrent systems, web performance and performance engineering, on which she has published several papers.