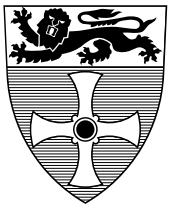
UNIVERSITY OF
NEWCASTLE



University of Newcastle upon Tyne

COMPUTING SCIENCE

Performance Modelling and Evaluation of E-Business Systems

G. Ferrari, P. Ezhilchelvan and I. Mitrani.

TECHNICAL REPORT SERIES

No. CS-TR-954 March, 2006

TECHNICAL REPORT SERIES

No. CS-TR-954 March, 2006

Performance Modelling and Evaluation of E-Business Systems

Giovanna Ferrari, Paul Ezhilchelvan and Isi Mitrani

Abstract

A general model of an e-business system comprising an application server and a database engine is analyzed and evaluated. Incoming jobs may have to wait in an external queue; when admitted for processing, they compete for the available resources. Database accesses are of different types and are subject to time-outs. An efficient approximate solution is provided; its accuracy is evaluated by comparing the model estimates with those obtained from simulations. The effect of several controllable parameters on the performance of the system is examined in a series of numerical and simulation experiments.

Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

FERRARI, G., EZHILCHELVAN, E., MITRANI, I..

Performance Modelling and Evaluation of E-Business Systems
[By] G. Ferrari, P. Ezhilchelvan, I. Mitrani.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2006.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-954)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-954

Abstract

A general model of an e-business system comprising an application server and a database engine is analyzed and evaluated. Incoming jobs may have to wait in an external queue; when admitted for processing, they compete for the available resources. Database accesses are of different types and are subject to time-outs. An efficient approximate solution is provided; its accuracy is evaluated by comparing the model estimates with those obtained from simulations. The effect of several controllable parameters on the performance of the system is examined in a series of numerical and simulation experiments.

About the author

Giovanna joined the Distributed Systems research group in May 2002, as a Research Associate within the TAPAS - Trusted and QoS Aware Provision of Application Services - Project. She obtained the degree in Computing Science at the University of Bologna in 2001, with a MS thesis on architecting mobile collaboration at the middleware level, developed during an industrial training at the NOKIA Research Center, Helsinki.

Paul Devadoss Ezhilchelvan received Ph.D. degree in computer science in 1989 from the University of Newcastle upon Tyne, United Kingdom. He received the Bachelor of Engineering degree in 1981from the University of Madras, India, and the Master of Engineering degree in 1983 from the Indian Institute of Science, Bangalore. He joined the School of Computing Science of the University of Newcastle upon Tyne in 1983 where he is currently a lecturer.

Isi Mitrani studied Mathematics at the Universities of Sofia and Moscow (Diploma, 1965), and Operations Research at the Technion, Haifa (MSc, 1967). He joined the University of Newcastle in 1968 as a Programming Advisor, became a Lecturer in 1969 (PhD, 1973), Reader in 1986 and Professor in 1991. He has held several visiting positions, including sabbatical years at INRIA (Le Chesnay) and Bell Laboratories (Murray Hill). His research interests are in the areas of probabilistic modelling, performance evaluation and optimization. Publications include 4 authored books, 3 edited books, more than 100 journal and conference papers and one patent.

Suggested keywords

APPLICATION SERVERS,
QUALITY OF SERVICE,
ANALYTICAL MODELLING,
QUEUING NETWORKS,
SIMULATIONS.

Performance Modeling and Evaluation of E-Business Systems

Giovanna Ferrari Paul Ezhilchelvan Isi Mitrani

School of Computing Science University of Newcastle, NE1 7RU, UK
[giovanna.ferrari, paul.ezhilchelvan, isi.mitrani]@ncl.ac.uk

Abstract. A general model of an e-business system comprising an application server and a database engine is analyzed and evaluated. Incoming jobs may have to wait in an external queue; when admitted for processing, they compete for the available resources. Database accesses are of different types and are subject to time-outs. An efficient approximate solution is provided; its accuracy is evaluated by comparing the model estimates with those obtained from simulations. The effect of several controllable parameters on the performance of the system is examined in a series of numerical and simulation experiments.

Keywords: Application Servers, Quality of Service, Analytical Modelling, Queuing Networks, Simulations.

1 Introduction

Providers of e-Business services, such as on-line banking, auctioning and retailing, must utilize their computing resources efficiently and offer a high quality of service to their users. In order to do that, it is important not only to implement effective admission and scheduling policies, but also to be able to predict the performance that the system can achieve for a given level of demand. To make quantitative predictions, one needs a model which is (a) sufficiently detailed to take into account the essential system features and (b) sufficiently simple to be analytically and numerically tractable. The development and solution of such a model is the aim of this paper.

Since the user demand is random in character, the measures of performance are usually described in terms of probabilities and random variables. The following metrics are commonly considered to be important for an e-Business system (or *site*) [12]:

1. *Average response time* (the time taken to process an accepted request).
2. *Throughput* (the average number of requests processed per second).
3. *Rejection probability* (the long-term fraction of requests that are rejected because the load is too heavy).

A model that allows fast evaluation of these performance measures in terms of measurable or controllable parameters can be used as a system management

tool. When the demand parameters change, the settings of various operational parameters (e.g., the number of server threads) can be changed so as to maintain optimal performance levels.

Existing work on the Quality of Service of application servers falls into three broad categories. The first of these treats the system as a 'black box' (see [1, 4–6, 9, 14, 18]). The behaviour of that box is measured and a policy that controls some operational characteristic (e.g., job admissions), reacts to observed changes in demand. In some cases a simple linear control model is postulated, e.g. [1, 4, 9, 14]. However, these approaches do not attempt to predict the performance of a system in relation to its structure.

The studies in the second category rely on a series of experiments whereby emulated clients are used to calibrate the behavior of the system [15, 19]. The statistics gathered during the experimental phase are used to configure the system for the required performance metrics during run time. Although there is an element of prediction here, it is limited by the fact that only situations similar to those encountered during the experiments can be effectively managed on-line [2, 3].

The third category includes works based on the analysis and evaluation of models where the system is represented by some kind of a queueing network [11, 8, 7, 16]. These studies differ from each other by the system features they choose to model, and the simplifying assumptions they introduce in order to make the model tractable.

We construct, analyze and simulate a rather general queueing model of an e-Business site. The contribution of the present paper lies in the incorporation into the model of *all* of the following features:

1. A user job may involve a random number of accesses to a database, each access being either read-only (lightweight) or transactional (heavyweight). A heavyweight access may terminate successfully (commit) or unsuccessfully (abort).
2. The database itself may be shared among more than one e-Business site. Hence, the database processing capacity (modelled as a number of parallel servers) allocated to a particular site is a controllable operational parameter.
3. Database access requests are subject to time-out intervals: if a request is not granted access before its time-out interval expires, the job that generated it is ejected. This is common in commercial database servers [10].
4. Computational and database resources are possessed simultaneously; that is, while a request waits for and receives a database service, the job that generated it continues to occupy an application server thread.
5. The physical components of the application server (central processor, one or more discs) are modelled explicitly.
6. There is an arrival process of e-Business jobs; those that are not admitted for processing (because all threads are occupied) wait in an external queue of finite or infinite capacity; if the former, and the queue is full when a job arrives, that job is lost. The presence of that external queue implies that the system model is not, in general, a closed queueing network.

Some of the above features are taken into account in other studies (e.g., Menasce et al [11] handle 4, 5 and 6; Liu et al [8] include a form of 4; Kounev and Buchmann [7] allow different types of requests, which may include 1). However, as far as we are aware, the entire collection is considered here for the first time.

The price paid for the generality of the model is that it cannot be solved exactly. The aim of the analysis is therefore to obtain a useable and efficient approximate solution. The accuracy of that solution is evaluated by comparisons with simulations.

The system architecture is described in section 2, while the mathematical model is defined in section 3. The analysis and approximate solution are detailed in section 4. The system performance is examined under a variety of loading conditions; the performance estimates provided by the model, and those obtained from the simulations, are presented in section 5.

2 System Architecture: an overview

E-Business service provisioning involves performing a variety of activities ranging from simple information display to data processing and updating of stored data. An e-Business site is usually structured into three logical *tiers* as shown in figure 1; each tier is responsible for a distinct set of activities.

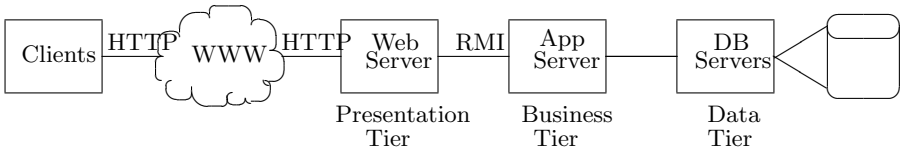


Fig. 1. Structure of an E-business site

Presentation Tier: This runs a Web Server which maintains the presentation logic of the application in the form of pre-formatted information. It receives the client requests coming from the outside network and passes them for processing to the next tier (by means of Remote Method Invocation calls) if business-specific computations are needed.

Business Tier: This is the Application Server which responds to invocations from the first tier: threads hosted by it perform computations using business logic and enterprize data. The former is typically implemented using some container component technology such as Enterprize Java Beans. The data required by the computation has to be fetched from the Data Tier (when not cached locally). If the computation modifies the data, then the updates are

implemented at the database by means of a transaction. If there are more calls to the application server than the available threads, the excess calls are queued.

Data Tier: This tier is represented by a set of database 'servers' which maintain the persistent data. In this context, the term 'server' is used in the sense of 'dedicated unit of database capacity', which may include processing power and disk storage; there is no connotation of a physical entity but there is an assumption that these units do not interfere with each other. The database servers allocated to an e-business site are, in general, a subset of a larger set provided by the database system and used by competing sites. That allocation is thus subject to management control.

Three remarks are in order. First, the work performed in the presentation tier is much less demanding than that carried out in the other two tiers, and therefore a well-provisioned Tier 1 is usually not a performance bottleneck (see [17]). Hence, in what follows, we will consider that the e-business site is made up of just the last two tiers, subjected to an external stream of requests.

Second, the database access requests from tier 2 are blocking in nature. That is, an application server thread waits until a database server returns the requested data or terminates the transaction. This means that the number of pending database access requests cannot exceed the number of application server threads.

Finally, if there are more application server threads than database servers, a request may have to wait in a queue for a database server to become available. That queue is special, in that each request has a timeout interval; the timeout is cancelled when the request is taken up for processing by a database server; if and when the timeout expires, the request is dropped and the calling thread is freed, resulting in an unsuccessful termination for the user job.

3 The model

The Business and Data Tiers are modelled by the (non-separable) queueing network illustrated in figure 2.

Requests (or 'jobs', as they will be called from now on) arrive into the system in a Poisson stream with rate λ. A maximum of m jobs are admitted for processing (they are referred to as the 'active' jobs); this limit is imposed by the number of parallel threads that have been made available. If there are more then m jobs present, the ones that do not occupy a thread wait in an external FIFO queue.

The execution of an active job consists of alternating 'computations' and 'database accesses'. Computation services are provided by a computer which, for the purposes of this model, is assumed to consist of one processor and one disc (the single disc assumption is not significant; one could easily generalize it by allowing multiple discs, at the price of complicating the solution). During a computation, jobs visit the processor at least once, and the disc zero or more

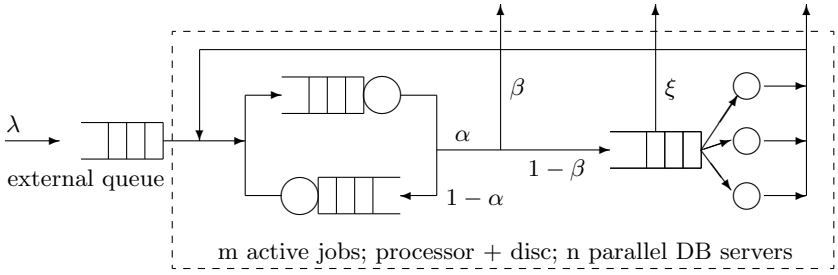


Fig. 2. An application server with database access

times. More precisely, after receiving a service at the processor, a job goes to the disc with probability $1-\alpha$ and completes the computation with probability α ($0 < \alpha \leq 1$). This models a geometrically distributed number of visits to the disc. After a disc service, a job returns to the processor with probability 1. There is a FIFO queue at both the processor and the disc.

Service times at the processor and the disc are assumed to be independent random variables distributed exponentially with means b_0 and b_1, respectively. These approximative assumptions are necessary for the tractability of the model.

Having completed a computation, a job terminates execution (and leaves the application server) with probability β, and makes a database access with probability $1-\beta$. In other words, a job makes a geometrically distributed number of database requests. Database accesses are handled by a database engine, which may possibly be shared by more than one application server. As far as this application server is concerned, up to n of its active jobs may have their database accesses processed in parallel without significant interference. This is modelled by assuming that there are n parallel DB servers. If more than n jobs need to access the database, n of them are being served while the others wait in a FIFO queue.

A time-out policy operates at the database engine: if, having asked for for a database access, a job does not acquire a DB server before its time-out period expires, that job is terminated and leaves the application server. In practice, the lengths of the time-out periods are fixed; however, for purposes of analytical tractability, it is assumed that they are independent random variables distributed exponentially with mean $1/\xi$.

Database accesses are further complicated by the fact that they can be of two types. Some are read-only queries; their service times are distributed exponentially with mean b_2. Others are transactions which typically take longer; they are distributed exponentially with mean b_3. After the completion of a read-only query, a job enters a new computation phase. On the other hand, a transaction is always the last of a job's database accesses; whether it ends with a commit or an abort, after it the job terminates and leaves the application server. A fraction θ of all jobs that make database accesses terminate with a transaction ($0 \leq \theta \leq 1$).

When an active job leaves the application server, whether successfully or unsuccessfully, the job at the head of the external queue (if any), moves to the processor queue; i.e., it becomes active and starts a computation.

Remarks.

1. When the external queue is non-empty, the application server behaves like a closed queueing network, with m jobs circulating between the computation and database engines. However, that network does not have a product-form solution (and hence is not tractable by mean-value analysis), because of the time-out departures from the database queue. That feature necessitates the approximate solution that will be presented in the next section.
2. The assumption that the computation part of the application server consists of a single processor and a single disc is not essential. One could envisage considerably more complex networks of processors and discs. An approximation would still be available, but would have to rely on numerical solution of equations; the closed-form expressions for the state-dependent computation throughput would not apply.

4 Approximate solution

Consider first the computation subsystem, with k jobs circulating between the processor and the disc ($k = 0, 1, \ldots, m$). Suppose that that circulation continues for a long time, i.e. the computation subsystem reaches steady-state with k jobs. Then the processor queue would behave like an M/M/1 queue with a bounded buffer of size k, into which jobs arrive at rate $1/b_1$ (when not full), and from which jobs depart at rate $(1-\alpha)/b_0$ (when not empty). Denote by r the ratio between those two rates:

$$r = \frac{(1-\alpha)b_1}{b_0} \ . \tag{1}$$

Known results for the M/M/1/k queue yield the probability, U_k, that the processor is busy, given that k active jobs are in their computational phase (e.g., see [13]):

$$U_k = \frac{1-r^k}{1-r^{k+1}} \ . \tag{2}$$

Hence, the state-dependent throughput of the computation subsystem when there are k jobs in it, t_k, is given by

$$t_k = \frac{\alpha}{b_0} U_k = \frac{\alpha}{b_0} \frac{1-r^k}{1-r^{k+1}} \ . \tag{3}$$

So, the first approximation is to replace the computation subsystem by a single state-dependent server whose service rate, when there are k jobs present, is t_k.

Now suppose that there are J active jobs, circulating between the computing and database engines for a sufficiently long period to reach steady state ($J = 1, 2, \ldots, m$). In other words, model the business and data tiers by the closed queueing network illustrated in figure 4.

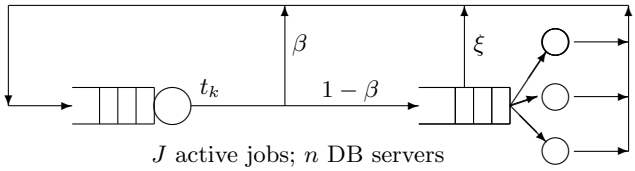


Fig. 3. Closed queueing network model

The state of this network is described by a single integer, k, specifying how many jobs are at the computing node; then $J-k$ jobs are at the database node. A transition from state k to state $k+1$ occurs when a job leaves the database node (through service completion or time-out); similarly, a transition from state $k+1$ to state k occurs when a job leaves the computing node.

Let p_k be the probability that the network is in state k ($k = 0, 1, \ldots, J$). The above implies that these probabilities satisfy the following balance equations:

$$[\min(n, J-k)\nu + \max(0, J-n-k)\xi]p_k = (1-\beta)t_{k+1}p_{k+1} \ , \qquad (4)$$

where ν is the average service rate of a database server.

This is a simple set of recurrences, of the form $p_{k+1} = a_k p_k$. Together with the normalizing equation ($p_0 + p_1 + \ldots + p_J = 1$), and the expressions (3) for t_k, they are easily solved. However, the average service rate ν is yet to be determined.

Recall that there are two types of database services, read-only queries and transactions. Given that a job visits the database at least once, it makes i such visits with probability $(1-\beta)^{i-1}\beta$. The last of those is a transaction with probability θ. Hence, the fraction, τ, of all visits to the database that are transactions, is given by

$$\tau = \theta\beta \sum_{i=1}^{\infty} \frac{1}{i}(1-\beta)^{i-1} = \frac{\theta\beta}{1-\beta} \sum_{i=1}^{\infty} \frac{1}{i}(1-\beta)^i = \frac{\theta\beta}{1-\beta} \ln \frac{1}{\beta} \ . \qquad (5)$$

Since the average service times of read-only queries and transactions are b_2 and b_3, respectively, the overall average database service time is

$$\frac{1}{\nu} = (1-\tau)b_2 + \tau b_3 \ , \qquad (6)$$

with τ given by (5). A mixture of different exponentially distributed random variables is not, in general, exponentially distributed. Nevertheless, the approximate

solution treats the database services as if they were distributed exponentially with mean $1/\nu$.

Having solved equations (4) and determined the probabilities p_k, the state-dependent throughput of the application server when it has J active jobs, $T(J)$, is obtained from

$$T(J) = \sum_{k=0}^{J} p_k[t_k\beta + \min(n, J-k)\nu\tau + \max(0, J-n-k)\xi] \ . \tag{7}$$

That throughput includes both successful and unsuccessful terminations.

The final approximation is to treat the closed queueing network in figure 4 as a single state-dependent server whose service rate is $T(J)$ when there are J active jobs. Now the external queue, together with the active jobs, behave like a one-dimensional Birth-and-Death process which is in state M if there is a total of M jobs present ($M = 0, 1, \ldots$). Then the number of active jobs is $\min(M, m)$ and the number of jobs in the external queue is $M - \min(M, m)$. There are transitions from state M to state $M+1$ with rate λ, and from state $M+1$ to state M with rate $T(\min(M+1, m))$.

Denote by q_M the steady-state probability that there are M jobs present. These probabilities are determined by the balance equations

$$\lambda q_M = T(\min(M+1, m))q_{M+1} \ , \ M = 0, 1, \ldots \ , \tag{8}$$

and the normalizing equation

$$\sum_{M=0}^{\infty} q_M = 1 \ . \tag{9}$$

Note that the departure rate in the right-hand side of (8) ceases to depend on M when the latter exceeds m. Therefore, we have

$$q_{m+i} = \rho^i q_m \ , \ i = 1, 2, \ldots \ , \tag{10}$$

where $\rho = \lambda/T(m)$ is the offered load when the number of active jobs is the maximum allowable. This leaves only a finite set of equations to solve.

The process is ergodic (i.e., the queue is stable), if $\lambda < T(m)$, or $\rho < 1$.

One could also assume a bounded external queue, with maximum size K. Then the possible states would be $M = 0, 1, \ldots, m+K$; equations (8) would apply only up to $M = m + K - 1$; the sum in the normalizing equation would be finite and the question of stability would not arise.

Given the steady-state probabilities q_M, one can compute certain performance measures. The average number of jobs in the system, L, is

$$L = \sum_{M=1}^{\infty} Mq_M = \sum_{M=1}^{m-1} Mq_M + \frac{q_m}{1-\rho}\left[m + \frac{\rho}{1-\rho}\right] \ . \tag{11}$$

In the case of a bounded external queue, with maximum size K, that expression becomes

$$L = \sum_{M=1}^{m-1} M q_M + \frac{q_m(1-\rho^{K+1})}{1-\rho}\left[m + \frac{\rho}{1-\rho}\right] - \frac{K q_m \rho^{K+1}}{1-\rho} \ . \tag{12}$$

The average system throughput, T, is equal to λ when the external queue is unbounded and stable. When it is bounded at level K, the throughput is

$$T = \lambda(1 - q_{m+K}) \ . \tag{13}$$

When the system throughput is not equal to λ (i.e., the external queue is bounded), some incoming jobs are rejected. In that case, the probability of rejection, R, is given by

$$R = \frac{\lambda - T}{\lambda} = q_{m+K} \ . \tag{14}$$

The average response time, W, is obtained from Little's result:

$$W = \frac{L}{T} \ . \tag{15}$$

Note that some of the system throughput is in fact due to database requests that have timed out, or to transactions that have aborted. Hence, one may be interested in the 'successful throughput', S, defined as the average number of jobs that complete successfully per unit time. Denote also the 'state-dependent successful throughput', given that there are J active jobs, by $S(J)$. That quantity is given by an expression similar to (7), but excluding the unsuccessful terminations:

$$S(J) = \sum_{k=0}^{J} p_k[t_k\beta + \min(n, J-k)\nu\tau(1-\gamma)] \ , \tag{16}$$

where γ is the probability that a transaction fails to commit (that is a database parameter).

Then, the unconditional successful throughput is obtained from

$$S = \sum_{M=0}^{m-1} q_M S(M) + S(m) \sum_{M=m}^{\infty} q_M \ , \tag{17}$$

where q_M is the solution of (8) and (9).

5 Numerical and simulation results

The performance of the system under different loading conditions and parameter settings was examined in a series of numerical and simulation experiments. The main purpose of the simulations was to evaluate the accuracy of the approximate solution. To that end, the simulated system contained all queues and both types of database accesses; there was no aggregation of database service times into a

single exponential distribution; the time-out intervals were of fixed size as in the real system, rather than being exponentially distributed random variables.

As the number of parameters is quite large, some of them were kept fixed throughout:

Probability of visiting the disc $= 1 - \alpha = 0.6$;
Probability of requiring a database access $= 1 - \beta = 0.6$;
Fraction of database accesses containing a transaction $= \theta = 0.2$
Average service time for read-only access $= b_2 = 20$ msec;
Average service time for transactions $= b_3 = 50$ msec;
Database time-out interval $= 1/\xi = 5$ sec (except in the last figure);
Bound on external queue size $= K = 50$.

Of the parameters that were were varied, the main ones are those that are directly controllable: the number of threads (m) and the number of database servers allocated (n).

The first set of experiments aimed to quantify the effect of the number of threads on the performance of the system. In figure 4, the average response time is plotted against m for different values of the external arrival rate, λ. The figure shows a marked initial improvement in performance as the number of active jobs increases; the servers are being utilized more efficiently by being able to work on different jobs at the same time.

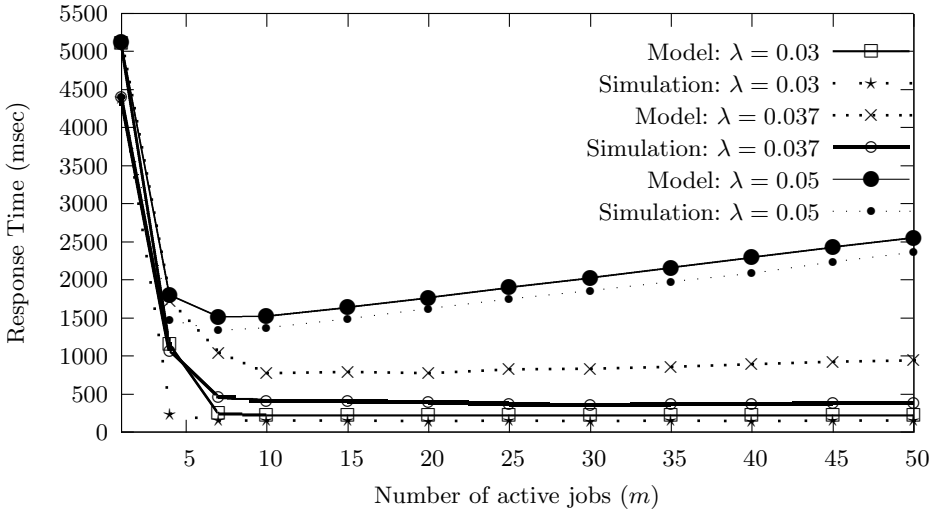


Fig. 4. Average response time as function of m; $n = 10$, $b_0 = b_1 = 5$ msec

However, the downward trend stops, or is reversed, when m increases further. That behaviour is readily explained by looking at figure 5, where the throughput, T, is plotted against m for the same arrival rates. When all servers are busy most of the time, the throughput reaches a maximum achievable value, T_{max}. In the present configuration, that value is approximately 0.04 jobs/msec (or 40 jobs/sec). If the external arrival rate is lower than T_{max}, the throughput flattens before reaching that maximum. Further increases in m make no difference to the average number of jobs present and the average response time; the extra threads remain unused.

If, on the other hand, $\lambda \geq T_{max}$, then the application server cannot cope with the demand; the external queue is full; adding more threads simply increases the average number of jobs present, and hence the average response time.

To summarize, there is an optimal number of threads, which depends on the average service times at the CPU, disk and database servers. For the present parameters, that number is approximately $m = 10$. Offering more threads brings no benefits when the system is lightly loaded, and is detrimental at heavy loads.

Note that the approximate solution is almost exact at light loads, and its relative error is 10% or less for heavily loaded systems. Also, it is worth pointing out that the approximate solution tends to be pessimistic, i.e., it overestimates the average response time and underestimates the throughput. That is probably due to the fact that replacing the constant time-out intervals with exponentially distributed ones increases their variance and hence has an adverse effect on performance (aggregating the different database service times into a single exponential distribution has the opposite, but less pronounced effect).

In the second set of experiments, illustrated in figures 6 and 7, the number of threads was kept fixed, while the number of available database servers varied. The value of m was deliberately set quite high ($m = 50$), so that the rate of database accesses is not restricted by under-utilized CPU and disc.

As expected, the system performance improves (the response time decreases, the throughput increases) with the addition of database servers, but only up to a point. Beyond that, adding more database service capacity ceases to make a difference; the extra servers are not utilized. What is perhaps less predictable is that this 'optimal number of database servers' does not seem to depend very much on the external arrival rate. For the present parameter set, under both light loads and heavy loads, it is best to allocate about 5 database servers.

The approximate solution predicts the optimal number of database servers very accurately. The largest differences between analysis and simulation are observed in the overloaded case ($\lambda = 0.05$) for the throughput, and at heavy load ($\lambda = 0.037$) for the response time.

Again, the model predictions are pessimistic, compared with the simulations. This phenomenon is observed in all experiments.

The third set of experiments is similar, except that a rather extreme service configuration is assumed. The average service times at the CPU and disc are reduced by a factor of 50 ($b_0 = b_1 = 0.1$ msec), while the number of threads is doubled ($m = 100$). In figures 8 and 9, the average response time and the

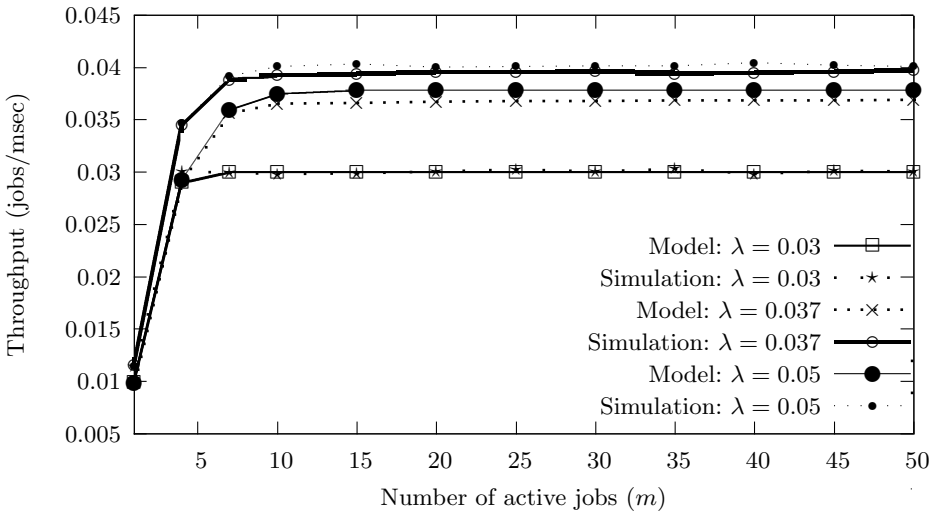


Fig. 5. Average throughput as function of m; $n = 10$, $b_0 = b_1 = 5$ msec

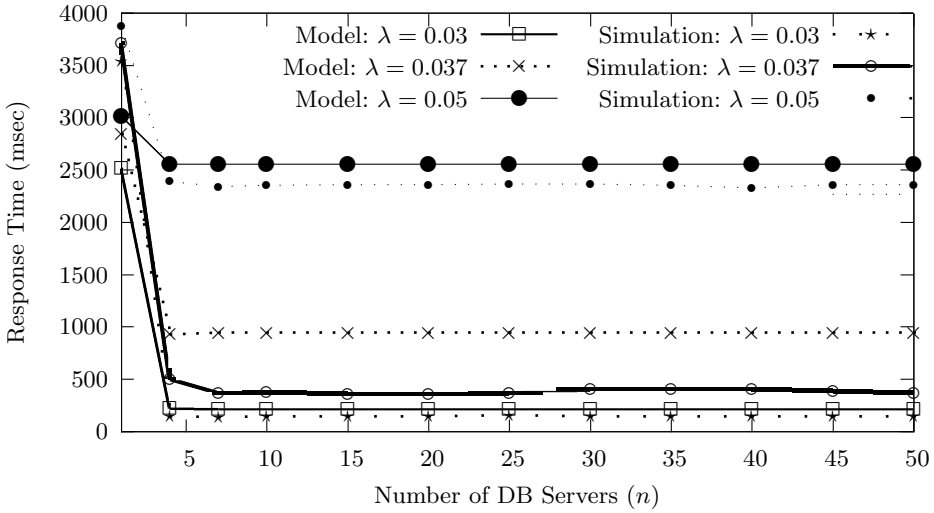


Fig. 6. Average response time as function of n; $m = 50$, $b_0 = b_1 = 5$ msec

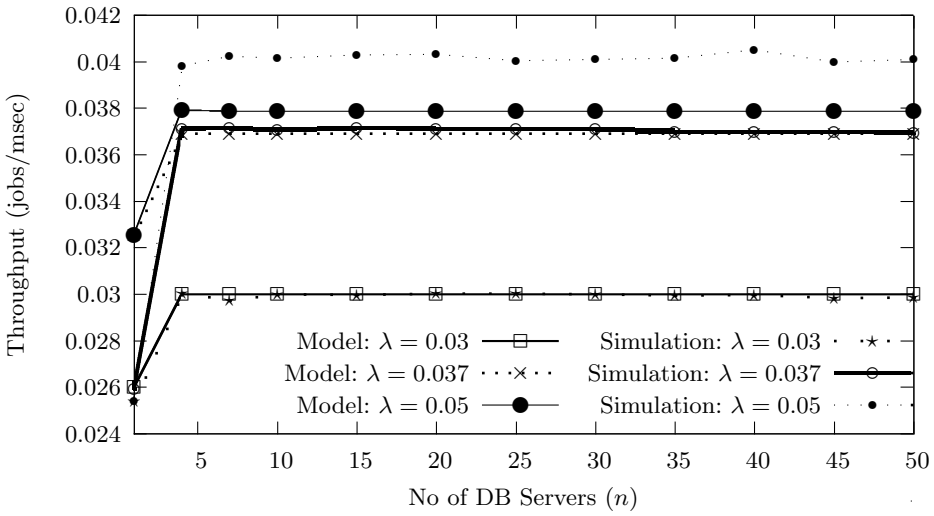


Fig. 7. Average throughput as function of n; $m = 50$, $b_0 = b_1 = 5$ msec

throughput, respectively, are plotted against the number of database servers for different external arrival rates.

The database is now subjected to much heavier traffic of accesses. It is still the case that increasing the number of database servers improves performance up to a point, and then ceases to do so. However, that point takes longer to reach, and it depends on the arrival rate: the higher the value of λ, the more database servers can be usefully employed. Also, because the model predictions are pessimistic, it tends to overestimate the the value of n beyond which there are no further performance benefits.

Figure 10 shows the rejection probability (i.e., the fraction of jobs that are lost because they find the external buffer full), plotted against the arrival rate, for different numbers of threads. There are no surprises here. On one hand, the rejection probability increases with λ because the system becomes more heavily loaded and hence the external queue is more often full. On the other hand, increasing m leads to a reduction in the rejection probability because more jobs can be admitted into the system.

The model estimates are reasonably accurate.

The final experiment is similar to that in figure 5, where the system throughput was plotted against the number of threads. All parameters are the same except that, in the present setting, the database timeout interval (or its average in the case of the model) is reduced by a factor of 100: $1/\xi = 50$ msec. Consequently, jobs are now more likely to be ejected from the application server

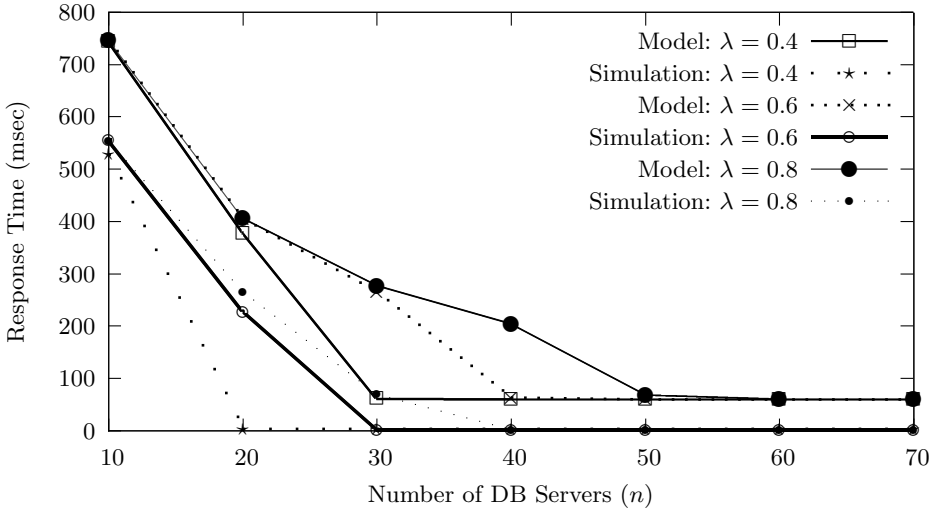


Fig. 8. Average response time as function of n; $m = 100$, $b_0 = b_1 = 0.1$ msec

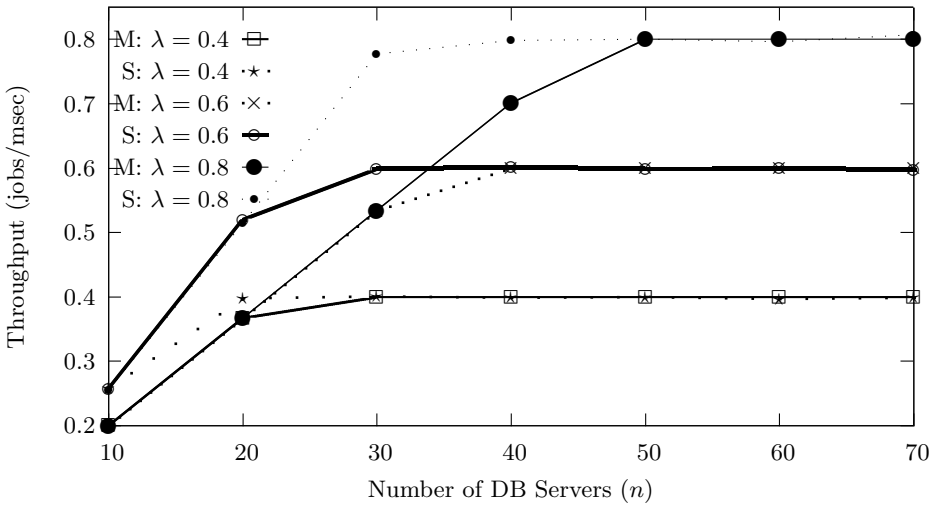


Fig. 9. Average throughput as function of n; $m = 100$, $b_0 = b_1 = 0.1$ msec

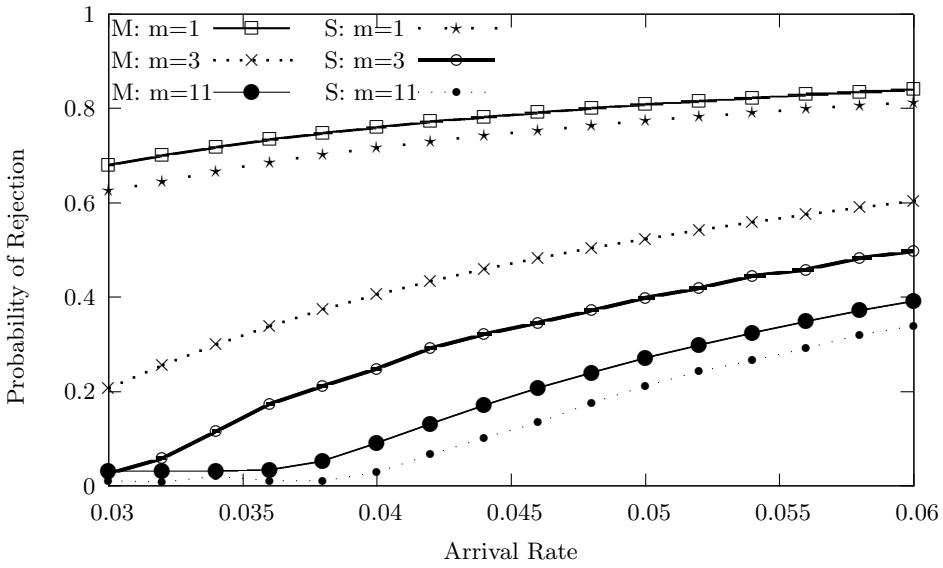


Fig. 10. Rejection probability as function of λ; $n = 10$, $b_0 = b_1 = 5$ msec

because their timeouts expire. Figure 11 illustrates the behaviour of the 'successful throughput', S, as a function of m, for different values of λ.

Comparing the results in figure 11 with those in figure 5, we observe that at light load ($\lambda = 0.3$) the shorter timeout interval has made very little difference to the throughput: it is still almost entirely successful. At medium to heavy load ($\lambda = 0.37$), there is some reduction in successful throughput. In the overloaded case ($\lambda = 0.5$), the difference is again minimal. It is worth noting that the model is, if anything, more accurate in predicting the successful throughput than the total one.

6 Conclusions

The model described here is sufficiently general to capture the behaviour of a large class of e-business systems. The proposed approximate solution is numerically efficient and fast. Its accuracy has been shown to be acceptable over a wide range of system configurations. Therefore, that solution can be used in order to (a) predict the performance of a system under given or assumed loading conditions, and (b) to choose optimal settings for certain controllable parameters with respect to specified performance measures.

An important future step would be to apply and evaluate this approach in a real-life e-business system of non-trivial size and complexity. Measurements would be taken in order to quantify both the system parameters and the observed

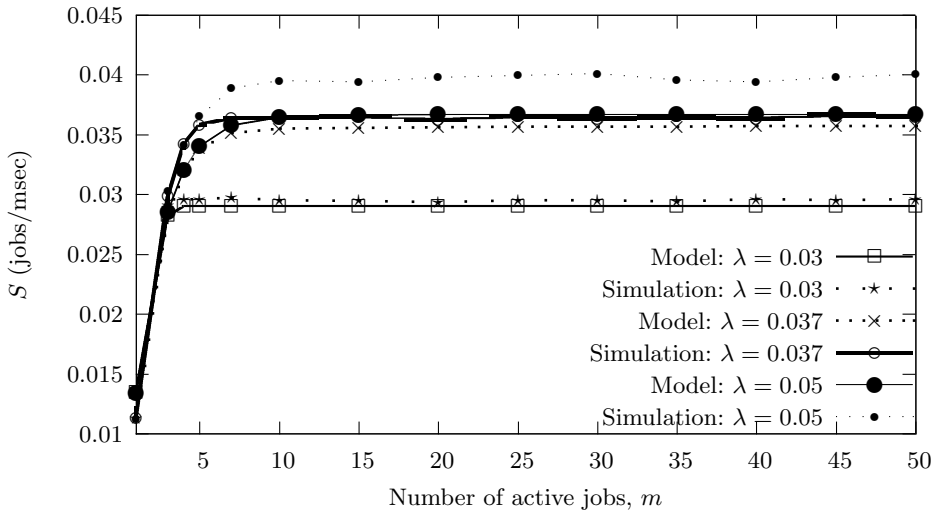


Fig. 11. Successful throughput as function of m; $n = 10$, $b_0 = b_1 = 5$ msec

performance. The latter would then be compared with the metrics predicted by the model.

We also plan to employ the model as analytical foundation for a Controller component, which monitors the service provisioning of and E-business site and adapts it to load variations.

Acknowledgement

This work was supported by the research project PACE (Protocols for Ad-hoc Collaborative Environments) and the Platform Grant (EP/D037743/1) funded by the UK Engineering and Physical Sciences Research Council. Also acknowledged is the support of the Network of Excellence EuroNGI (Next Generation Internet), funded by the EU FP6 programme.

References

1. T.F. Abdelzaher, K.G. Shin and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach", IEEE Trans. on Parallel and Distributed Systems, 13, pp. 80-96, 2002.
2. P. Brebner, S. Chen, I. Gorton, J. Gosper, L. Hu, A. Liu, D. Palmer and S. Ran, "Report: Evaluating J2EE Application Servers", Technical Report CSIRO Middleware Technology Evaluation Series, 2002

3. E. Cecchet, J. Marguerite, and W. Zwaenepoel, "Performance and scalability of ejb applications", Procs. 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, pp. 246 - 261, Seattle, 2002.
4. Y. Diao, N. Gandhi, J.L. Hellerstein, S. Parekh and D.M. Tilbury, "Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Server", Procs., IEEE/IFIP Network Operations and Management, pp. 219-234, Florence, 2002.
5. S. Elnikety, E. Nahum, J. Tracey and W. Zwaenepoel, "A Method for Transparent Admission Control and Request Scheduling in Dynamic E-Commerce Web Sites", Procs., Int. World Wide Web Conf., pp. 276-286, 2004.
6. A. Kamra, V. Misra and E. Nahum, "Yaksha: A Self-Tuning Controller for Managing the Performance of 3-Tiered Web sites", IWQoS, pp. 47-56, Montreal, 2004
7. S. Kounev and A. Buchmann, "Performance Modeling and Evaluation of Large-Scale J2EE Applications", Procs. 29th Int. Conf. on Resource Management and Performance Evaluation of Enterprise Computing Systems (CMG2003), pp. 273-283, 2003.
8. Y. Liu, A. Fekete and I. Gorton, "Design Level Performance Modeling of Component-based Applications", TR 543, School of Information Technologies, Univ. Sydney, 2003.
9. C. Lu, T.F. Abdelzaher, J.A. Stankovic and S.H. Son, "A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers", IEEE Real-Time Technology and Applications Symposium (RTAS'01), p. 51, 2001.
10. J. McGovern, R. Adatia, Y. Fain, J. Gordon, E. Henry, W. Hurst, A. Jain, M. Little, V. Nagarajan, H. Oak and L. Phillips, Java 2 Enterprise Edition 1.4 (J2EE 1.4), Wiley, 2003.
11. D.A. Menasce, D. Barbara and R. Dodge, "Preserving QoS of e-commerce sites through self-tuning: a performance model approach", Procs. 3rd ACM Conf. on Electronic Commerce, pp. 224-234, Florida, 2002.
12. D.A. Menasce and V.A. Almeida, Capacity Planning for Web Services, Prentice-Hall, 2002.
13. I. Mitrani, Probabilistic Modelling, Cambridge University Press, 1998.
14. S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram and J. Bigus, "Using Control Theory to Achieve Service Level Objectives In Performance Management", Real-Time Systems Journal, 23, 1-2, pp. 127-141, 2002.
15. M. Raghavachari, D. Reimer and R. D. Johnson, "The Deployer's Problem: Configuring Application Servers for Performance and Reliability", Procs. 25th Int. Conf. on Software Engineering (ICSE'03), pp. 484 - 489, Washington, 2003.
16. B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications", Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS). pp. 291-302. Canada, 2005.
17. D. Villela, P. Pradhan and D. Rubenstein, "Provisioning Servers in the Application Tier for E-Commerce Systems", Procs. IWQoS'04, pp. 57-66, Montreal, 2004.
18. M. Welsh and D. Culler, "Adaptive overload control for busy Internet servers", USENIX Symposium on Internet Technologies and Systems (USITS), San Francisco, 2003.
19. B. Xi, Z. Liu, M. Raghavachari, C. Xia and L. Zhang, "A Smart Hill-Climbing Algorithm for Application Server Configuration", International WWW Conference, pp. 287 - 296, New York, 2004.