The response-time distribution in a real-time database with optimistic concurrency control and constant execution times

Eindhoven University
of Technology

Department of Mathematics
and Computing Science

Memorandum COSOR 97-07

The response-time distribution in a real-time database with optimistic concurrency control and constant execution times

S.A.E. Sassen
J. van der Wal

Eindhoven, March 1997
The Netherlands

The Response-Time Distribution in a Real-Time Database with Optimistic Concurrency Control and Constant Execution Times

Simone Sassen and Jan van der Wal

Abstract

For a real-time shared-memory database with optimistic concurrency control, an approximation for the transaction response-time distribution is obtained. The model assumes that transactions arrive at the database according to a Poisson process, that every transaction uses an equal number of data-items uniformly chosen, and that the multiprogramming level is bounded. The execution time of all transactions is constant. The behavior of the system is approximated by an $M/D/c$ queue with feedback. The probability that a transaction must be fed back for a rerun depends on the number of transactions that has committed during its execution. Numerical experiments, which compare the approximative analysis with a simulation of the database, show that the approximation of the response-time distribution is quite accurate, even for high system loads.

1 Introduction

Real-time databases combine the requirements of both databases and real-time systems. In a database, transactions (database requests) should preserve database consistency. Subject to this consistency requirement, the transaction throughput of the database should be maximized. In a real-time system, the main requirement is timeliness, i.e., transactions must be executed before their deadlines. Soft real-time systems are allowed to miss some deadlines when the system is overloaded, but at least a certain fraction of the transactions should meet some prescribed deadline. In a real-time database, both consistency and timeliness are important. In this paper, we investigate soft real-time databases and are interested in the probability that a transaction meets its deadline.

To benefit from the increase in CPU power that parallel computer architectures offer, transactions on databases should be executed concurrently. However, concurrent execution can destroy database consistency if conflicting transactions are incorrectly scheduled. Two transactions conflict if they access the same data-item, at least one of them with the intention to write. To execute conflicting transactions, a concurrency control scheme is needed. Concurrency control schemes govern the simultaneous execution of transactions such that overall correctness of the database is maintained (see e.g. PAPADIMITRIOU [1986]). The two main concurrency control schemes are locking and optimistic concurrency control.

Under the locking scheme, an executing transaction holds locks on all data-items it needs for execution, thus introducing lock waits for transactions that conflict with it. Consistency is guaranteed, however chains of lock waits can lead to high transaction response times.

When the conflict probability is low, it can be advantageous to use the optimistic concurrency control (OCC) scheme proposed by KUNG and ROBINSON [1981] . Under OCC, all CPUs can be used for transaction processing at the same time, even for processing conflicting transactions. Each transaction is processed in three phases: an execution phase, a validation phase and a commit phase. In the execution phase a transaction T accesses any data-item it needs for execution, regardless of the number of transactions already using that data-item. During the execution phase, all actions T performs on data-items are only done on local copies of the data-items. In the validation phase, all items used by T are checked for conflicts. If a conflict has occurred with a transaction that committed after T started (that is, if at least one of the data-items read by T was in the meantime changed globally by another transaction), T must be rerun. The local changes T made to data-items then don't become global but are erased. If no conflicts occurred, T completes the validation phase successfully and enters the commit phase, where the data-items used by T are updated globally.

In this paper, we study the performance of a real-time database with OCC, where the execution time of all transactions is constant. An analytical model is formulated and solved in order to find the probability that a transaction meets its deadline. Existing analytical performance studies for OCC (such as MENASCÉ and NAKANISHI [1982], MORRIS and WONG [1985], and KLEINROCK and MEHOVIĆ [1992]) only consider *average* system performance, such as throughput, average response time, and the average number of restarts needed for a transaction. (The response time of a transaction is the total time between its arrival and its commit.) Knowledge about average response times is not enough to estimate the probability that a transaction meets its deadline: for this, an approximation of the response-time *distribution* is required. As far as we know, except for our previous study (SASSEN and VAN DER WAL [1997]), no analytical performance studies of real-time databases with OCC exist that address the distribution of the response time. In SASSEN and VAN DER WAL [1997], we approximated the response-time distribution in a real-time database with OCC where the execution times of transactions are exponentially distributed. The present paper is a follow-up on that study by considering transactions with constant execution times.

The rest of the paper is organized as follows. The model for a real-time database with OCC and constant execution times is explained in Section 2. In Section 3, we derive two approximative analyses for the response-time distribution. Numerical results, which compare analysis with simulation, are presented in Section 4. Moreover, Section 4 contains recommendations on how to choose the number of CPUs needed to achieve some prespecified performance level. Finally, Section 5 contains some concluding remarks.

2 The Model

In the introduction, we described the optimistic concurrency control (OCC) scheme in detail. In this section, we model OCC in a shared-memory environment with N parallel CPUs as a multi-server queueing system with feedback, see Figure 1 for an illustration.

In the dashed area, which represents the N CPUs, at most N transactions can be present. Each transaction is handled by one CPU and either leaves the system (after a successful execution), or is rerun

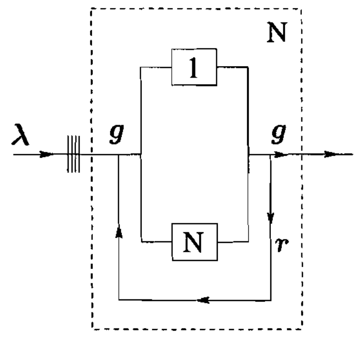


Figure 1: *Queueing model of the system*

(in case of a conflict). We assume the time needed for one execution plus validation of a transaction is constant and equal to D. Further, it is assumed that the commit phase takes negligible time compared to execution plus validation, and that validation can be efficiently done in parallel. The assumption that commit takes negligible time compared to execution plus validation is reasonable, since we consider a system where all data-items are in main memory so where no disks are attached. Transactions arrive at the database according to a Poisson process with rate λ. An arriving transaction that finds all CPUs busy joins the queue. As soon as a CPU is freed by a departing transaction, the transaction first in queue is taken into execution. We also refer to execution plus validation as one transaction run.

With regard to transaction behavior, we assume that two transactions conflict with probability b. The conflict probability b is an input parameter for characterizing the amount of data contention in the system. The value of b is larger when the (number of data-items in the) database is smaller or when transactions access more data-items.

The queueing model of Figure 1 is no standard feedback model. The probability that a transaction T must be rerun is not fixed, but depends on the number of transactions that departed (committed) during the execution of T. The number of departures during T's execution depends on the length of T's execution (but the length is constant and equal to D for all transactions) and on the number of concurrently executing transactions. Since it is an open system, the number of concurrently executing transactions varies during T's execution.

For an exact analysis of the system, in which transactions are only rerun if conflicting transactions committed during their execution, it is convenient to label the transactions in service by colors, say green and red. A transaction T is green at the start of every run. During its run, T is colored red as soon as a transaction commits that conflicts with T. A red transaction discovers at its validation that it has to be rerun (it then returns to the CPU as a green transaction); a transaction that is still green at validation time need not be rerun so is allowed to commit. In this way, the color of a transaction at validation time determines whether the transaction must be rerun. The colors red and green are depicted in Figure 1 as r and g, respectively.

Using this colorful representation of transactions, the state of the system at time t is exactly described by the vector $(w(t), c_1(t), r_1(t), \ldots, c_N(t), r_N(t))$ with $w(t)$ the number of waiting transactions at time t, and $c_i(t)$ and $r_i(t)$, respectively, the color (red, green) and the remaining execution time of the transaction at CPU i at time t ($r_i(t) \equiv 0$ if CPU i is free), $i = 1, \ldots, N$. With this state-description, a *simulation* program of the system is easily made. However, an exact *analysis* of the system with this state-description seems so intractable that we are not very optimistic about the chances of finding one. Therefore, in the next section we propose an approximative analysis of the system.

3 Approximative Analysis

As explained in the previous section, it is only possible to exactly determine whether a transaction must be rerun by either using a coloring of transactions or by registrating, for every transaction T in execution, the number of conflicting transactions that committed during T's execution. Such a large state-description is not tractable for an analysis of the system. Therefore, as an approximation of the feedback mechanism, suppose we know the probability that a transaction is still green at the end of its run, given that it found $n-1$ other transactions (of which the colors are not known) in execution when it started its run ($n = 1, \ldots, N$). Let us denote this *success probability* by $p(n)$. (If a transaction is still green at the end of its run this is called a success.) Then, in order to determine whether a transaction T must be rerun, we would only have to know how many transactions were present at the start of T's run; if this number is n, the probability that T does not have to be rerun is $p(n)$.

The only difficulty is, that we can't determine the exact value of the probability $p(n)$ that a transaction is still green at the end of its run, given that it found $n - 1$ other transactions in execution at the start of its run. The reason for this is, that success of a transaction does not only depend on the total *number* of transactions in execution at the start of the run, but also on the *colors* of these transactions. Nevertheless, we can approximate the probability $p(n)$. In Section 3.1, an approximation for $p(n)$ is derived by looking at a so-called 'closed' system where the number of transactions in service is constant at n (for $n = 1, \ldots, N$).

Using the probabilities $p(n)$ as success probabilities, we approximate the queueing model with colored transactions of Figure 1 by the queueing model with probabilistic feedback of Figure 2 below.

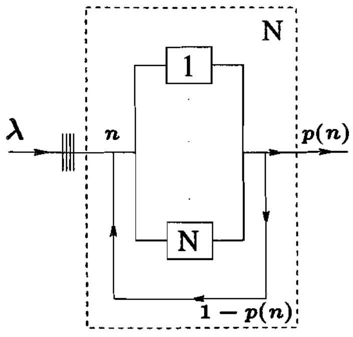


Figure 2: *M/D/N queue with starting-state dependent feedback*

We call the queueing model of Figure 2 an *$M/D/N$ queue with starting-state dependent feedback.* Without feedback, the model is an $M/D/N$ queue because of the Poisson arrival process, the deterministic execution times, and the N servers. The feedback mechanism is called *starting-state dependent*, because the probability of feedback is $p(n)$ if the total number of customers in service (i.e., the state of the system) at the *start* of the run was equal to n.

The $M/D/N$ queue with starting-state dependent feedback is not known in literature. Since no existing analysis of this queue is available, we analyzed it ourselves. An exact analysis seems impossible. In Section 3.2 and 3.3, respectively, we study two different approximations for computing the steady-state probabilities of the $M/D/N$ queue with starting-state dependent feedback. The two approaches are similar to those in SASSEN and VAN DER WAL [1996], where an $M/D/N$ queue with state-dependent feedback was analyzed (there the success probability of a run depends on the number of customers in service at the *end* of the run). Section 3.4 shows how to approximate the response-time distribution in an $M/D/N$ queue with starting-state dependent feedback by using the steady-state probabilities computed in Section 3.2 or 3.3. The approximation for the response-time distribution derived in Section 3.4 serves as approximation for the response-time distribution of the queueing model of Figure 1, so for the actual real-time database we are interested in.

3.1 Approximation for the success probability $p(n)$

The success probability $p(n)$ of a transaction that sees $n-1$ other transactions in execution at the start of its run is approximated by the success probability $p_c(n)$ in a closed system with a constant number of n transactions in service. Thus, we consider the system of Figure 3.

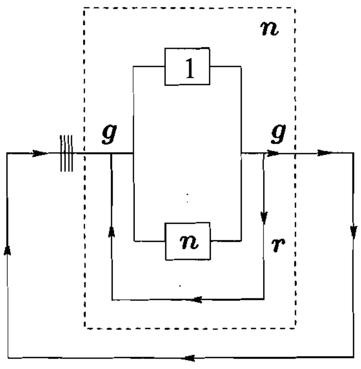


Figure 3: *Closed system with n transactions*

In the closed system of Figure 3, the number of transactions present (in execution) is kept fixed at n: as soon as a transaction commits and departs, a new transaction is started at the free CPU. For this system, we would like to derive $p_c(n)$, the probability that a transaction is still green at the end of its run. From now on we only consider $n > 1$, since $p_c(1) \equiv 1$.

Exact computation of $p_c(n)$

In order to compute $p_c(n)$, the state of the closed system is exactly described by the $(n-1)$-dimensional vector $(c_1, \ldots, c_{n-1})$, where c_i denotes the color (r for red and g for green) of the transaction that is the i-th one to finish its run. The most fresh transaction (the transaction that is the n-th to finish) is always green, so c_n need not be included in the state-description. Notice that for computing $p_c(n)$ the remaining service times are irrelevant, so it is not necessary to include them into the state-description.

The Markov chain $\{(c_1, \ldots, c_{n-1})_j, j = 1, 2, \ldots\}$ exactly describes the evolution of the transaction colors in the closed system. From the steady-state vector π of this Markov chain, the success probability $p_c(n)$ is computed as

$$p_c(n) = \sum_{c_2, \ldots, c_{n-1}} \pi(g, c_2, \ldots, c_{n-1}),$$

which is the sum over all states where the transaction completed next is green.

As an illustration, we compute $p_c(2)$ and $p_c(3)$. Let us first consider $n = 2$. Figure 4 is a transition diagram of the Markov chain.

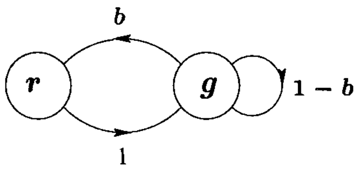


Figure 4: *Transition diagram of closed system with n=2*

The transition diagram is explained as follows. If the transaction that finishes first is red, it is restarted as a green transaction, so the state changes from r to g with probability 1. If a green transaction is finished, with probability b it colors the other green transaction red, and with probability $1-b$ it doesn't. As soon as the green transaction commits, a new green transaction enters execution. Thus, from state g the system state changes to r with probability b and to g with probability $1 - b$. From the balance equation $\pi(r) = b\pi(g)$ and the normalization equation $\pi(r) + \pi(g) = 1$ it easily follows that $\pi(r) = b/(1 + b)$ and $\pi(g) = 1/(1 + b)$.

The case $n = 3$ results in the transition diagram of Figure 5.

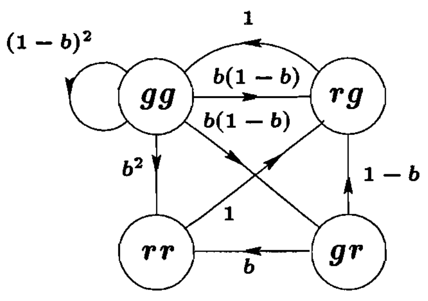


Figure 5: *Transition diagram of closed system with n=3*

From the balance equations we obtain

$$\pi(g,g) = c, \quad \pi(r,g) = b(2-b)c, \quad \pi(g,r) = b(1-b)c, \quad \text{and} \quad \pi(r,r) = b^2(2-b)c,$$

with $c = 1/(1 + 3b - b^3)$. The success probability $p_c(3)$ is

$$p_c(3) = \pi(g, g) + \pi(g, r) = (1 + b - b^2)/(1 + 3b - b^3).$$

The number of states of the Markov chain is 2^{n-1} so increases exponentially fast in n. For $n = 10$, the number of states that can be attained is 512. Solving the balance equations then requires solving a system of 512 linear equations in as many unknowns. This is implementable on a fast computer, but a less elaborate way to compute $p_c(n)$ is strongly desired. We next derive an extremely simple approximation for $p_c(n)$, which is very accurate as well.

Approximation for $p_c(n)$

In approximating $p_c(n)$, we don't use any information about the colors of the transactions present. We simply treat all transactions in the same way by assuming that every validating transaction has the same (still unknown) probability $p_c(n)$ of being successful. Now tag a transaction T. Every transaction that validates during T's execution is successful (commits) with probability $p_c(n)$. Thus, every transaction that validates during T's run colors T red (so makes T unsuccessful) with probability $bp_c(n)$. T is still green at the end of its run if all transactions that validate during T's run do not color T red. Now make the following important observation: since all transactions last equally long, exactly $n - 1$ transactions validate during T's run. Hence, the probability that T is still green at the end of its run is $(1 - bp_c(n))^{n-1}$. Thus we have the equality

$$p_c(n) = (1 - bp_c(n))^{n-1}.$$

The approximation we propose for $p_c(n)$ is the unique fixed point of this equation on the interval $(0, 1]$. A simple search procedure like bisection can be used to find the fixed point $p_c(n)$.

Comparison of the exact and approximate value of $p_c(n)$ shows, that the approximation is very accurate. In the special case of $n = 2$, the approximation is exact. For $n = 3$ and $n = 10$, we compare the exact and approximate values of $p_c(n)$ for some choices of b in Table 1. The fourth column shows the

	$p_c(3)$				$p_c(10)$		
b	exact	approximated	% diff	b	exact	approximated	% diff
0	1	1	0	0	1	1	0
0.01	0.98048639	0.98048645	0.01	0.01	0.92016213	0.92016844	7E-4
0.1	0.83910701	0.83920217	0.01	0.1	0.58108885	0.58260747	0.26
0.2	0.72864322	0.72949017	0.12	0.2	0.43262129	0.43810444	1.27

Table 1: *Exact and approximated* $p_c(3)$ *and* $p_c(10)$ *for various* b

relative difference. The difference increases with b, but when $b = 0.2$ it is still only 0.12% for $n = 3$ and 1.27% for $n = 10$. Values of b larger than 0.2 are not interesting, as OCC then isn't the appropriate concurrency control scheme anyway. OCC was designed for databases with a small conflict probability, see KUNG and ROBINSON [1981].

Summarizing, our approximation of $p_c(n)$ in the closed system of Figure 3 serves as approximation for the success probability $p(n)$ in the open system of Figure 2. For $n = 1$ to 10, this leads to the following approximate values for $p(n)$ (see Table 2).

$b \setminus n$	1	2	3	4	5	6	7	8	9	10
0.01	1.000	0.990	0.980	0.971	0.962	0.953	0.945	0.936	0.928	0.920
0.1	1.000	0.909	0.839	0.783	0.736	0.697	0.663	0.633	0.606	0.583
0.2	1.000	0.833	0.729	0.656	0.600	0.555	0.518	0.488	0.461	0.438

Table 2: *Approximate success probabilities $p(n)$ for various b*

3.2 Approximation I for the steady state of the $M/D/N$ queue with starting-state dependent feedback

Approximation I is based on the following approximation assumption regarding the time between two successive service completion epochs. The assumption is similar to the approximation assumption TIJMS et al. [1981] used for the $M/G/c$ queue, and was already used in SASSEN and VAN DER WAL [1996] to approximate the steady-state probabilities of an $M/D/c$ queue with state-dependent feedback. In an $M/D/c$ queue with state-dependent feedback, the probability that a service run is successful depends on the number of customers (transactions) in service at the end of the run, so not at the start. The approximation assumption for the $M/D/N$ queue with starting-state dependent feedback is as follows.

Approximation Assumption

1 a) If just after a *successful* service completion epoch k customers are in the system with $1 \leq k < N$, then the time until the next service completion epoch is distributed as the minimum of k independent random variables, each uniformly distributed over $(0, D)$.

b) If just after an *unsuccessful* service completion epoch k customers are in the system with $1 \leq k < N$, then the time until the next service completion epoch is distributed as the minimum of the deterministic variable D and $k-1$ independent random variables, each uniformly distributed over $(0, D)$.

2 If just after a *successful* or *unsuccessful* service completion epoch $k \geq N$ customers are in the system, then the time until the next service completion epoch equals D/N with probability 1.

In other words, when $k < N$, the approximation assumption states that the remaining service time of each service in progress is distributed as the equilibrium excess distribution of the original service time. The equilibrium excess distribution of a deterministic variable D is a uniform distribution over

$(0, D)$, see e.g. TIJMS [1994]. When $k \geq N$, the approximation assumption states that the system behaves like an $M/D/1$ system with feedback, in which the single server works N times as fast as each of the N servers in the original system.

This type of approximation assumption, based on the equilibrium excess distribution of the service times, is well known and was first applied successfully for approximating the steady-state probabilities of the $M/G/c$ queue by TIJMS et al. [1981]. We showed in SASSEN and VAN DER WAL [1996] that the approximation assumption also yields very accurate results for the $M/D/c$ queue with state-dependent feedback. Therefore, we think that the approximation assumption is also very useful for analyzing the $M/D/N$ queue with starting-state dependent feedback.

Using the above assumption, we model the $M/D/N$ queue with starting-state dependent feedback by an embedded Markov chain that only considers the system just after service (or run) completion epochs. The possible states of this embedded Markov chain are:

- $(k, 0)$: just after an unsuccessful service completion epoch k customers are in the system, $k \geq 1$. One of the services has just started.
- $(k, 1)$: just after a successful service completion epoch k customers are in the system, $k \geq 0$. If $k \geq N$, a new service has just started. Otherwise, all services were already in progress.

Since the state-description does not contain information about how many transactions were in service when a transaction, say T, started its run, the success probability of transaction T can't be determined exactly by this embedded Markov chain. To overcome this problem, we approximate the number of transactions in execution at the start of T's run by the number of transactions in execution just after the run completion epoch of the transaction that finishes its run last before T. The idea is that this latter number *is* known exactly (because the state is observed at every run completion epoch) and will not differ much from the first number. So again we sacrifice exactness for the sake of a smaller state space, thus for analyzability.

We want to compute the steady-state probabilities of this approximative embedded Markov chain. Once the steady state at service completion epochs has been found, the steady-state probabilities at arbitrary epochs in time are calculated very easily.

In order to derive the balance equations of the Markov chain, we need some notation. Let R_j be distributed as the minimum of j independent, uniform$(0, D)$-distributed random variables ($j = 1, \ldots, N-1$). Let $R_j(t)$ be the distribution function of R_j. Then

$$R_j(t) = \begin{cases} 1 - \left(1 - \frac{t}{D}\right)^j, & t < D \\ 1, & t \geq D \end{cases} \qquad \text{for} \quad 1 \leq j \leq N-1.$$

Define for $j = 1, \ldots, N-1$ and $\ell \geq 0$ the probability $a[j, \ell]$ as the probability that ℓ customers arrive during R_j. Also, define $a[0, \ell]$ as the probability that ℓ customers arrive during D, and $a[N, \ell]$ as the probability that ℓ customers arrive during D/N. Since the arrival process is Poisson with intensity λ,

$$a[0,\ell] = e^{-\lambda D}\frac{(\lambda D)^\ell}{\ell!} \qquad \text{and} \qquad a[N,\ell] = e^{-\lambda D/N}\frac{(\lambda D/N)^\ell}{\ell!}.$$

Computing $a[j,\ell]$ for $j = 1,\ldots,N-1$ is more cumbersome, but can be done by conditioning on R_j.

$$\begin{aligned} a[j,\ell] &= \int_0^D e^{-\lambda t}\frac{(\lambda t)^\ell}{\ell!}\,\mathrm{d}R_j(t) \\ &= \sum_{i=0}^{j-1}\frac{(\ell+i)!}{\ell!}\frac{j(-1)^i}{(\lambda D)^{i+1}}\binom{j-1}{i}\left[1-\sum_{m=0}^{\ell+i}e^{-\lambda D}\frac{(\lambda D)^m}{m!}\right], \quad 1\le j\le N-1,\ \ell\ge 0. \end{aligned}$$

Here we applied the binomium of Newton and the useful identity

$$\int_0^D \lambda e^{-\lambda t}\frac{(\lambda t)^k}{k!}\mathrm{d}t = 1-\sum_{m=0}^{k}e^{-\lambda D}\frac{(\lambda D)^m}{m!}.$$

Now the steady-state vector π of the embedded Markov chain is the unique non-negative solution to the balance equations

$$\begin{aligned}
\pi(k,1) &= \sum_{\ell=0}^{k} p(k-\ell+1)a[k-\ell,\ell]\pi(k-\ell+1,0) + p(1)a[0,k]\pi(0,1) \\
&+ \sum_{\ell=0}^{k} p(k-\ell+1)a[k-\ell+1,\ell]\pi(k-\ell+1,1), \qquad 0\le k\le N-2 \\
\pi(N-1,1) &= \sum_{\ell=1}^{N-1} p(N-\ell)a[N-1-\ell,\ell]\pi(N-\ell,0) + p(N)a[N,0]\pi(N,0) \\
&+ \sum_{\ell=0}^{N-1} p(N-\ell)a[N-\ell,\ell]\pi(N-\ell,1) + p(1)a[0,N-1]\pi(0,1) \\
\pi(k,0) &= \sum_{\ell=0}^{k-1}(1-p(k-\ell))a[k-\ell-1,\ell]\pi(k-\ell,0) + (1-p(1))a[0,k-1]\pi(0,1) \\
&+ \sum_{\ell=0}^{k-1}(1-p(k-\ell))a[k-\ell,\ell]\pi(k-\ell,1), \qquad 1\le k\le N-1 \\
\pi(k,1) &= \sum_{\ell=0}^{k-N+1} p(N)a[N,\ell]\pi(k-\ell+1,0) + \sum_{\ell=k-N+2}^{k} p(k-\ell+1)a[k-\ell,\ell]\pi(k-\ell+1,0) \\
&+ \sum_{\ell=0}^{k} p(\min\{k-\ell+1,N\})a[\min\{k-\ell+1,N\},\ell]\pi(k-\ell+1,1) + p(1)a[0,k]\pi(0,1), \\
&\qquad k\ge N \\
\pi(k,0) &= \sum_{\ell=0}^{k-N}(1-p(N))a[N,\ell]\pi(k-\ell,0) + \sum_{\ell=k-N+1}^{k-1}(1-p(k-\ell))a[k-\ell-1,\ell]\pi(k-\ell,0) \\
&+ \sum_{\ell=0}^{k-1}(1-p(\min\{k-\ell,N\}))a[\min\{k-\ell,N\},\ell]\pi(k-\ell,1) + (1-p(1))a[0,k-1]\pi(0,1), \\
&\qquad k\ge N
\end{aligned}$$

together with the normalization equation

$$\sum_{k=1}^{\infty}[\pi(k,0)+\pi(k,1)]+\pi(0,1)=1.$$

Notice, that a transaction that arrives at an empty system is always successful, so $p(1) = 1$. For clarity, we did not substitute this value for $p(1)$ in the above balance equations.

The balance equations can be solved by truncating the state space at a large level M (say), so at the states $(M, 0)$ and $(M - 1, 1)$, and rejecting customers that find M customers in the system.

Another way to solve the balance equations is by exploiting the geometric-tail behavior of the embedded Markov chain, as in TIJMS and VAN DE COEVERING [1991]. It can be shown that the Markov chain has a single geometric tail. For details, see SASSEN and VAN DER WAL [1996]. Thus there exist a large M and a $\tau \in (0,1)$ such that for $k \geq M$, $\pi(k,0) \approx \pi(M,0)\tau^{k-M}$ and $\pi(k,1) \approx \pi(M,1)\tau^{k-M}$. From the balance equations for $\pi(k,1)$ and $\pi(k,0)$ for $k \geq N$, we find that τ is the unique root of the equation

$$1-p(N)(1-y)=\exp(\lambda D(1-1/y)/N) \tag{1}$$

on the interval $(0,1)$. When $p(N) = 1$, this equation simplifies to

$$1/y=\exp(-\lambda D(1-1/y)/N),$$

the equation for the geometric tail of the ordinary $M/D/N$ queue.

Computing τ from (1) and substituting $\pi(k,0) = \pi(M,0)\tau^{k-M}$ and $\pi(k,1) = \pi(M,1)\tau^{k-M}$ for $k \geq M$ in the balance equations and in the normalization equation leads to a system of $2M + 1$ linear equations. This system can easily be solved since M does not have to be very large to obtain reasonable accuracy of the solution. Typically, the value of M required by the geometric-tail approach to obtain some desired accuracy is much smaller than the value of M required when solving for the steady-state probabilities by truncating the state space, especially when the traffic intensity ρ is large, see TIJMS and VAN DE COEVERING [1991].

Next, we show how the steady-state probabilities of the $M/D/N$ queue with starting-state dependent feedback can be computed from the steady-state probabilities of the embedded Markov chain. Denote by $\varphi_I(k)$ the fraction of departing customers that leaves k customers behind in the system. Once $\pi(i,1)$ is known for $i \geq 0$, $\varphi_I(k)$ can be computed as

$$\varphi_I(k)=\frac{\pi(k,1)}{\sum_{i\geq 0}\pi(i,1)}, \qquad k\geq 0.$$

Since customers arrive one at a time and are served one at a time, the fraction of real departures that leaves k customers behind equals the fraction of new customers that finds k customers in the system upon arrival. Further, because of the Poisson arrival process, we have by the PASTA property (WOLFF [1982]) that the long-term fraction of time that k customers are in the system equals the fraction of arrivals that finds k customers in the system.

Hence, the probabilities $\varphi_I(k)$ are our first approximation for the steady-state probabilities of the $M/D/N$ queue with starting-state dependent feedback.

3.3 Approximation II for the steady state of the $M/D/N$ queue with starting-state dependent feedback

In Approximation I, the time between successive service completions is approximated. Moreover, the success probability is approximated. Approximation II, which will be discussed next, is inexact with respect to the success probability but exact with respect to time. In SASSEN and VAN DER WAL [1996] a similar Approximation II was derived for the $M/D/c$ queue with state-dependent feedback.

Let us explain Approximation II. Just as in the exact analysis of the ordinary $M/D/c$ queue by CROMMELIN [1932], we observe the state of the system every D time units. Since the service times are constant and equal to D, any customer in service at some time t will have completed his service — either successfully or unsuccessfully — at time $t + D$. The customers present at time $t + D$ are exactly those customers who completed an unsuccessful service during $(t, t + D]$, plus the customers who were either waiting in queue at time t or who arrived in $(t, t+D]$. Hence, we can relate the number of customers in the system at time $t + D$ to the number in the system at time t.

To do this, let $q_k(u)$ be the probability that k customers are in the system at time u. Also, let $a[\ell]$ be the probability that ℓ customers arrive in $(t, t + D]$, so $a[\ell] = e^{-\lambda D}(\lambda D)^\ell/\ell!$ for $\ell \geq 0$. Finally, let B_i^j denote the probability that i services are completed successfully during a time-interval $(0, D]$, given that j customers are in the system at the start of the interval. How to find B_i^j is discussed in detail below, but first we state the relation between the number of customers present at time t and at time $t + D$.

By conditioning on the state at time t we find

$$q_k(t + D) = \sum_{j=0}^{N+k} q_j(t) \sum_{i=\max\{0,j-k\}}^{\min\{j,N\}} B_i^j a[k - j + i] \quad \text{for} \quad k \geq 0.$$

Thus, an approximation for the time-average probabilities q_k is found from the linear equations

$$\begin{aligned} q_k &= \sum_{j=0}^{N+k} q_j \sum_{i=\max\{0,j-k\}}^{\min\{j,N\}} B_i^j a[k-j+i], \qquad k \geq 0, \\ \sum_{k=0}^{\infty} q_k &= 1. \end{aligned} \tag{2}$$

It remains to specify the probability B_i^j, that is, the probability that i services are completed successfully during a time-interval $(0, D]$ if j customers are present at time 0. Of course, in the special case that $p(n) = 1$ for all n, $B_i^j = 1$ for $i = \min\{N, j\}$ and $B_i^j = 0$ otherwise. Then the model reduces to the ordinary $M/D/N$ queue and the analysis is exact. However, for the general $M/D/N$ queue with starting-state dependent feedback, it is not possible to compute the exact value of B_i^j if the system state is observed only after every D time units.

The probability that a service is successful depends on the number of customers present at the moment the service is started. This number is not known exactly, because the system state is not observed at service start epochs. Therefore, we studied the following approximation for B_i^j. We approximated

B_i^j by the probability that a binomial($\min\{N,j\}, p(\min\{N,j\})$) distributed random variable equals i. This approximation ignores that the transactions that finish service during $(0, D]$ have a success probability that is dependent on the number of transactions in execution at their starting time, which lies before time 0 instead of at time 0.

Just as for Approximation I, it can be seen that the probabilities q_k have a geometric tail, i.e., $q_k \approx q_{k-1}\tau$ as $k \to \infty$. The geometric-tail factor τ is exactly equal to the tail of Approximation I, so τ is the root of equation (1) on the interval $(0, 1)$. Hence, the probabilities q_k can be computed by choosing a large M and substituting $q_k = q_M\tau^{k-M}$ in (2) for $k \geq M$.

The steady-state probabilities q_k obtained by solving (2) are our second approximation for the steady-state probabilities of the $M/D/N$ queue with starting-state dependent feedback. We denote these probabilities by $\varphi_{II}(k)$, $k \geq 0$.

3.4 Approximation for the response-time distribution

Define S as the response time of an arbitrary transaction. Using the approximation assumption of Section 3.2 and Approximation I or II for the steady-state probabilities of the $M/D/N$ queue with starting-state dependent feedback, we approximate the distribution of S.

Let the random variable L denote the steady-state number of transactions in the system. Denote our approximation for the distribution of L by $\{\varphi(k),\ k \geq 0\}$. (This can be either $\varphi_I(k)$ or $\varphi_{II}(k)$.) According to Little's theorem, $E[L] = \lambda E[S]$. Hence, we compute our approximation for the expected response time $E[S]$ as

$$E[S] = \frac{1}{\lambda}\sum_k k\varphi(k).$$

To approximate the distribution of S, we need to approximate the distribution of the waiting time and the total service time of a transaction.

Let us first discuss the service-time distribution. Every service run of a transaction takes D time. The probability that another run is needed depends on the number of transactions in execution at the moment the present run was started. This number is only known for the first run of a transaction and is given by the steady-state probabilities. For the later runs, we approximate the number in execution at the start of a run of transaction T by the number of transactions in execution at the start of the *first* run of transaction T. Thus we approximate the service time by a geometrical distribution.

Next, we discuss the waiting-time distribution. If a transaction T finds $i \geq N$ transactions in the system upon arrival, it has to wait until $i - N + 1$ service completions have been successful. Using part 2 of the approximation assumption of Section 3.2, the time between the arrival of T and the next service completion is approximately uniform$(0, \frac{D}{N})$-distributed. As an approximation, we say that with probability $p(N)$ that service is successful. Then T still has to wait for $i - N$ successful service completions. With probability $1 - p(N)$, that service is unsuccessful. Then T still has to wait for $i - N + 1$ successful service completions. As long as all CPUs are busy, the number of service completions needed for j successful services is approximately negative-binomially distributed with parameters j and $p(N)$. Hence, using part 2 of the approximation assumption, the time needed for j

successful service completions (starting just after a service completion epoch) is approximately D/N times a negative-binomial$(j, p(N))$ distributed variable.

Denote by G_i a geometrically distributed random variable with success probability $p(i)$, denote by NB_j a negative-binomially distributed variable with parameters j and $p(N)$, and let $U(0,a)$ be a uniform $(0,a)$-distributed random variable. Summarizing the above discussion, the approximation we suggest is as follows.

Total Service-Time Distribution If a transaction T sees i other CPUs busy at the start of its first service run, the distribution of the total service time of T is approximated by $D\,G_{i+1}$.

Waiting-Time Distribution If a transaction T finds $i \geq N$ transactions in the system upon arrival, the waiting-time distribution of T is approximated by

$$\begin{cases} U(0,\frac{D}{N}) + \frac{D}{N}NB_{i-N} & \text{w.p.} \quad p(N) \\ U(0,\frac{D}{N}) + \frac{D}{N}NB_{i-N+1} & \text{w.p.} \quad 1-p(N). \end{cases}$$

Here 'w.p.' means 'with probability'.

Our approximation for the response-time distribution thus is

$$\begin{aligned} P(S \leq t) &= \sum_{i=0}^{N-1} \varphi(i)P(D\,G_{i+1} \leq t) + p(N)\sum_{i=N}^{\infty} \varphi(i)P(U(0,\tfrac{D}{N}) + \tfrac{D}{N}NB_{i-N} + D\,G_N \leq t) \\ &\quad +(1-p(N))\sum_{i=N}^{\infty} \varphi(i)P(U(0,\tfrac{D}{N}) + \tfrac{D}{N}NB_{i-N+1} + D\,G_N \leq t). \end{aligned}$$

Similarly, for a direct approximation of $E[S^2]$ we propose

$$\begin{aligned} E[S^2] &= \sum_{i=0}^{N-1} \varphi(i)E[D^2G_{i+1}^2] + p(N)\sum_{i=N}^{\infty} \varphi(i)E[(U(0,\tfrac{D}{N}) + \tfrac{D}{N}NB_{i-N} + D\,G_N)^2] \\ &\quad +(1-p(N))\sum_{i=N}^{\infty} \varphi(i)E[(U(0,\tfrac{D}{N}) + \tfrac{D}{N}NB_{i-N+1} + D\,G_N)^2]. \end{aligned}$$

Note, that we could have approximated $E[S]$ in this way as well. However, we already have an approximation for $E[S]$ from Little's theorem.

4 Numerical Results

In Section 4.1, we test the accuracy of the Approximations I and II by comparing the approximations for the response-time distribution with the values produced by a simulation of the system. In Section 4.2, we discuss an example of the applicability of the analysis for real-time database design. We show how to choose the number of CPUs needed to achieve some prespecified performance level.

4.1 Response-time distribution

We tested Approximation I and II by comparing them with a simulation of the system. Without loss of generality, the execution time D of the transactions was taken equal to 1. In the simulation program, the system state was registered exactly in the record $(w(t), c_1(t), r_1(t), \ldots, c_N(t), r_N(t))$ as described at the end of Section 2. Applying Approximation I and II, we computed $E[S]$, sdev(S) (the standard deviation of S), $P(S > 2)$, $P(S > 5)$, and $P(S > 10)$ as described in Section 3.4.

We looked at systems with $N = 2, 4, 8$, and 10. Besides N, the input parameters were the conflict probability b and the arrival intensity per CPU λ_1 (so $\lambda_1 = \lambda/N$). The parameter b was chosen at 0, 0.01, 0.1, and 0.2. To explain how λ_1 was varied, we define

$$\rho_U := \frac{\lambda D}{N p_c(N)} = \frac{\lambda_1 D}{p_c(N)}.$$

Since $p_c(n)$ is decreasing in n, ρ_U is an upper bound on the server utilization. The arrival intensity per CPU, λ_1, was varied such, that for every choice of b systems with ρ_U from 0.50 to about 0.98 were investigated.

Table 3 and Table 4 contain a representative selection of the simulation and analysis results. For various choices of the input parameters N, λ_1, b, and ρ_U, the tables show $E[S]$ and sdev(S). In addition, Table 3 shows $P(S > 2)$ and $P(S > 5)$, and Table 4 shows $P(S > 5)$ and $P(S > 10)$.

				$E[S]$			sdev(S)			$P(S > 2)$			$P(S > 5)$		
N	λ_1	b	ρ_U	ApI	ApII	Sim	ApI	ApII	Sim	ApI	ApII	Sim	ApI	ApII	Sim
2	0.70	0.01	0.71	1.56	1.53	1.53	0.77	0.76	0.76	0.20	0.19	0.19	0.005	0.004	0.004
		0.10	0.77	1.92	1.94	1.93	1.26	1.24	1.20	0.34	0.34	0.34	0.031	0.031	0.029
	0.80	0.01	0.81	2.02	1.98	1.98	1.24	1.23	1.23	0.37	0.35	0.35	0.034	0.032	0.032
4	0.60	0.10	0.77	1.58	1.63	1.61	0.97	0.92	0.86	0.23	0.23	0.22	0.009	0.009	0.007
	0.90	0.00	0.90	2.04	2.00	2.00	1.20	1.20	1.20	0.38	0.36	0.37	0.032	0.030	0.031
8	0.40	0.10	0.63	1.33	1.40	1.40	0.86	0.79	0.74	0.10	0.10	0.10	0.004	0.004	0.003
		0.20	0.82	2.15	2.26	2.21	1.67	1.63	1.52	0.36	0.38	0.37	0.061	0.065	0.054
10	0.35	0.10	0.60	1.35	1.42	1.43	0.90	0.83	0.78	0.10	0.10	0.10	0.004	0.005	0.003
		0.20	0.80	2.12	2.23	2.21	1.69	1.65	1.58	0.34	0.35	0.35	0.058	0.061	0.055

Table 3: *Distribution of the response time S: analysis versus simulation*

The simulated values are accurate up to the last digit shown. The number of transactions simulated was such, that the width of the 95% confidence interval is smaller than the last shown decimal place. For instance, a simulated value of 2.56 for $E[S]$ means, that the 95% confidence interval lies inside $[2.55, 2.57]$.

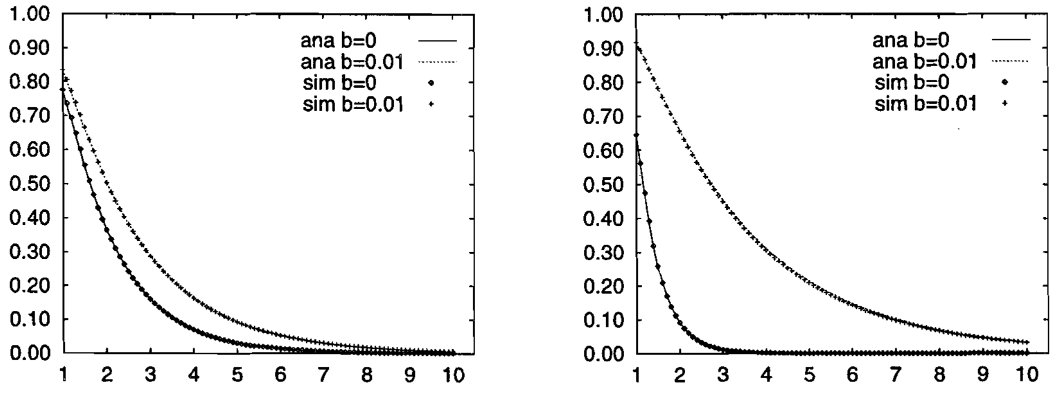
The numerical experiments indicate, that both approximative analyses give reasonably accurate results. For both approximations, the error in $E[S]$ is typically smaller than 7%, where the worst cases are those with $b = 0.1$ or 0.2 and $\rho_U \geq 0.91$. The relative error in sdev(S) is typically below 10%, with a few exceptions again for $b = 0.1$ or 0.2 and $\rho_U \geq 0.90$, or for sdev(S)< 1, with errors of about

12%. For $b = 0$ and $b = 0.01$, the approximations are very good: irrespective of the (high) value of ρ_U, the errors in $E[S]$ and sdev(S) are always below 2%.

				$E[S]$			sdev(S)			$P(S>5)$			$P(S>10)$		
N	λ_1	b	ρ_U	ApI	ApII	Sim	ApI	ApII	Sim	ApI	ApII	Sim	ApI	ApII	Sim
2	0.70	0.20	0.84	2.70	2.80	2.72	2.17	2.15	2.01	0.13	0.13	0.12	0.012	0.013	0.010
	0.80	0.10	0.88	3.12	3.13	3.10	2.45	2.43	2.38	0.17	0.17	0.17	0.022	0.022	0.020
		0.20	0.96	9.26	9.36	8.97	8.7	8.7	8.3	0.60	0.61	0.60	0.34	0.34	0.33
	0.90	0.00	0.90	3.20	3.15	3.15	2.42	2.41	2.40	0.18	0.17	0.17	0.022	0.021	0.021
		0.01	0.91	3.50	3.45	3.44	2.72	2.72	2.71	0.21	0.21	0.21	0.034	0.033	0.033
4	0.60	0.20	0.91	3.69	3.82	3.58	3.10	3.08	2.74	0.25	0.26	0.23	0.047	0.049	0.034
	0.70	0.10	0.89	2.59	2.62	2.53	1.89	1.86	1.74	0.10	0.10	0.090	0.007	0.007	0.005
	0.90	0.01	0.93	2.60	2.56	2.56	1.78	1.77	1.76	0.10	0.09	0.09	0.006	0.006	0.005
8	0.45	0.20	0.92	3.62	3.71	3.55	2.85	2.83	2.59	0.24	0.25	0.22	0.037	0.038	0.028
	0.90	0.01	0.96	2.70	2.68	2.66	1.83	1.83	1.80	0.10	0.10	0.10	0.007	0.007	0.006
10	0.40	0.20	0.91	3.40	3.42	3.31	2.58	2.55	2.38	0.20	0.21	0.19	0.025	0.025	0.020
	0.55	0.10	0.94	3.30	3.32	3.10	2.41	2.39	2.14	0.19	0.19	0.16	0.020	0.020	0.012
	0.90	0.01	0.98	3.57	3.55	3.53	2.68	2.68	2.67	0.22	0.21	0.21	0.033	0.033	0.032

Table 4: *Distribution of the response time S: analysis versus simulation*

As for the tail probabilities of the response time, the tables show that Approximation I and II are not accurate up to the second decimal, but the absolute error almost always is smaller than 0.02. Hence, Approximation I and II are sufficiently accurate for database design purposes. Moreover, the analyses tend to give a slight *over*estimation, so a pessimistic view, of $P(S > t)$, which in the design phase is preferred to a too optimistic view.



(a) $N = 4$ (b) $N = 10$

Figure 6: *$P(S > t)$ against t for $b = 0$ and $b = 0.01$*

In Figure 6(a) and (b), respectively, we plotted $P(S > t)$ against t for $b = 0$ and $b = 0.01$, with $\lambda_1 = 0.90$, and $N = 4$ and $N = 10$, respectively. The symbols depict simulation results, the (dotted) lines analysis results from Approximation II. According to the subfigures, simulation and analysis results agree very well. Both subfigures also clearly show the effect of data conflicts on the response time under OCC. For $N = 10$, the difference in $P(S > t)$ between the cases $b = 0$ and $b = 0.01$ is much larger than for $N = 4$: more parallelism implies more conflicts.

Finally we point out, that the approximations for the system behavior are not only good, but also very fast. The runtime of the simulations exceeded the runtime of the analyses by a factor up to 1000 (so where simulation took a day, the analysis took only about 1 minute).

4.2 How many CPUs are needed?

How can we determine which number of CPUs will yield the best system performance, given the conflict probability b and the total arrival intensity λ?

The answer to this question depends on the performance measure of interest. As explained in Section 1, the performance measure of interest for a database is throughput, whereas for a *real-time* database (RTDB) the fraction of transactions that meets its deadline is more important. The latter performance measure is called a *real-time* performance measure. In the following, we first consider the maximum throughput of a RTDB, given the number of CPUs N and the conflict probability b. Next, we discuss how to determine which number of CPUs is needed to guarantee some prespecified *real-time* performance level, for a RTDB with arrival intensity λ.

Maximum throughput

An approximation for the maximum throughput (MT) can be calculated as $Np_c(N)$ using Table 2. For $N = 1$ to 10, Table 5 shows MT and the marginal efficiency (ME) of having one extra CPU. The marginal efficiency ME at N is the difference of the throughput with $N - 1$ and N CPUs. It shows the extra useful capacity available if CPU N is added. Dependent on the cost of one extra CPU, and on the transaction arrival rate λ, the appropriate value of N can be chosen using Table 5.

N	1	2		3		4		5		6		7		8		9		10	
b	MT	ME	MT	ME	MT	ME	MT	ME	MT	ME	MT	ME	MT	ME	MT	ME	MT	ME	MT
0	1	1	2	1	3	1	4	1	5	1	6	1	7	1	8	1	9	1	10
0.01	1	0.98	1.98	0.96	2.94	0.94	3.88	0.93	4.81	0.91	5.72	0.90	6.62	0.87	7.49	0.86	8.35	0.85	9.20
0.1	1	0.82	1.82	0.70	2.52	0.62	3.13	0.55	3.68	0.50	4.18	0.46	4.64	0.42	5.06	0.39	5.45	0.38	5.83
0.2	1	0.67	1.67	0.52	2.19	0.44	2.62	0.38	3.00	0.33	3.33	0.30	3.63	0.28	3.90	0.25	4.15	0.23	4.38

Table 5: *Maximum throughput and marginal efficiency for various N and b*

Notice, that if the number of CPUs N tends to infinity, the maximum throughput MT(N) also tends to infinity, though very slowly. Since this is theoretically more interesting than practically, we refer to the Appendix for a further discussion of this asymptotic behavior.

Real-time performance

In a RTDB, the performance is measured as the percentage of transactions that meets its deadline. Let us introduce the following notion. A RTDB is (t, α)-*efficient* if at least $\alpha\%$ of the transactions meets its deadline t. Formally: $P(S \leq t) \geq (\alpha/100)$ with $0 \leq \alpha \leq 100$. We call α the *efficiency level* or the *service level*. For example, from Table 4 it can be seen that a RTDB with $N = 4$, $b = 0.01$ and $\lambda = 3.6$ is $(10, 99)$-efficient and $(5, 90)$-efficient, but not $(5, 95)$-efficient.

For a database with given b and N, it is interesting to determine the maximum value of the arrival intensity λ for which the system is still (t, α)-efficient. We denote this maximum value of λ by $\lambda^*_{t,\alpha}(N, b)$. Using our approximative analysis II, we have computed $\lambda^*_{t,\alpha}(N, b)$ for various N, b, t, and α. For every choice of N, b, t, and α, a bisection method with respect to λ was applied to find $\lambda^*_{t,\alpha}(N, b)$. In this way, the performance of the system had to be recomputed for many different values of λ. (Being able to deal with this kind of complicated performance questions underlines the tremendous importance of having a fast analytic approach for performance calculations. With simulations it would take weeks to get an answer, whereas with the analysis it is only a matter of minutes.)

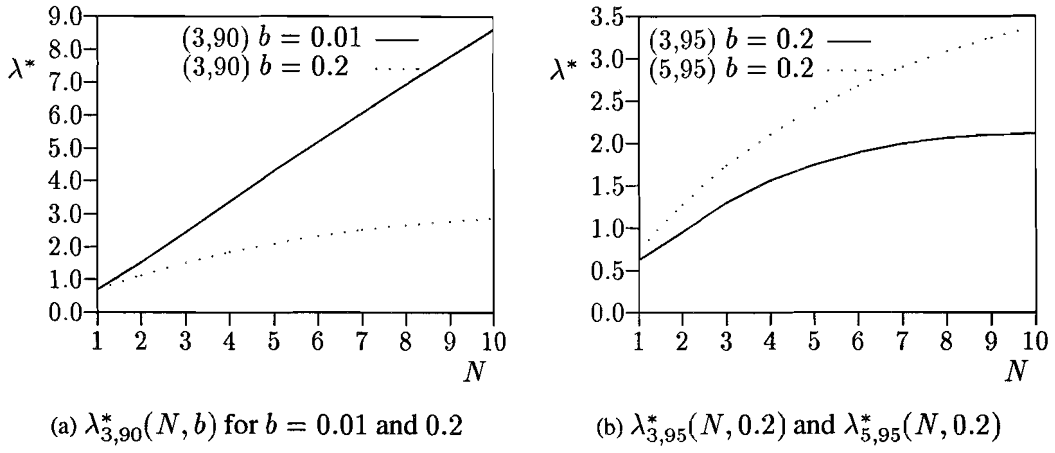


Figure 7: *Maximum arrival rate $\lambda^*_{t,\alpha}(N, b)$ as a function of N*

Figure 7(a) plots $\lambda^*_{3,90}(N, b)$ against N, for a system with $b = 0.01$ and a system with $b = 0.2$. The influence of conflicts ($b = 0.2$ versus $b = 0.01$) is clear: it reduces the value of the maximum allowable arrival intensity dramatically.

Figure 7(b) plots $\lambda^*_{t,95}(N, 0.2)$ against N for $t = 3$ and $t = 5$. We remark that each of the graphs converges to a finite value as N gets large, which was not visible for the system with $b = 0.01$ in Figure 7(a). The interpretation of the flat curves at large N is that, from a certain value of N on, the real-time performance (i.e., the service level α) of the system cannot be improved: adding more CPUs becomes practically useless.

Plots like Figure 7 are very helpful for design purposes: they can be used to determine which number of CPUs is needed to guarantee (t, α)-efficiency if the conflict probability is b and the arrival intensity is λ.

For example, suppose we have a conflict probability b of 0.2 and an arrival intensity λ of 1.5. Also, suppose we require that the system to be built must be $(3, 95)$-efficient, i.e., at least 95% of the transactions must have a response time smaller than 3. Then Figure 7(b) shows us, that this service requirement is not fulfilled with 3 CPUs, because $\lambda^*_{3,95}(3, 0.2) = 1.28 < 1.5$. Since $\lambda^*_{3,95}(4, 0.2) = 1.55$, 4 CPUs suffice to satisfy this 95% service level.

As a second example, suppose we have built a system with 8 CPUs for the situation $b = 0.2$ and $\lambda = 2$. As can be seen from Figure 7(b), the system with $N = 8$ is $(3, 95)$-efficient. However, what happens to the efficiency level of the RTDB if the traffic intensity λ increases by 20% to 2.4? How many CPUs must be added in order to restore the $(3, 95)$-efficiency? As can be guessed from the figure, and as is concluded from numerical experiments, the value of $\lambda^*_{3,95}(N, 0.2)$ never exceeds 2.3. Theoretical support for this is given in VAN DER WAL and SASSEN [1997]. Hence, if λ increases to 2.4, the RTDB cannot be made $(3, 95)$-efficient anymore, no matter how many CPUs are added. It is important to keep this observation in mind when designing a RTDB with OCC.

5 Concluding Remarks

In this paper, we derived two approximations for the response-time distribution in a real-time database with optimistic concurrency control, N CPUs, and constant execution times of length D. Approximation I was based on an embedded Markov chain analysis and the well-known residual life approximation of TIJMS et al. [1981] was used for the remaining execution times of the transactions in execution. Approximation II resembled the exact analysis of the $M/D/N$ queue (CROMMELIN [1932]) by observing the state of the system after every D time units.

Numerical experiments for various system loads and conflict probabilities indicated, that both approximations produce reasonably accurate estimates for the transaction response-time distribution. The error in the average response time is typically smaller than 7%, and the error in the standard deviation of the response time is typically below 10%. The absolute error in the distribution of the response time is at most 0.03.

The combination of reasonable accuracy and very short computing times (compared to simulation) makes our approximative analyses very well suited for the purpose of real-time database design. With little effort, it is possible to compute the number of CPUs needed to achieve some prespecified real-time performance. We have proposed to measure the performance of a real-time database in terms of (t, α)-efficiency. A real-time database is (t, α)-efficient if, under a given arrival intensity of transactions, at least $\alpha\%$ of the transactions have a response time smaller than (their time to deadline) t. Using the analyses, we can for instance check whether the real-time database remains (t, α)-efficient when an increase in the transaction arrival rate occurs, and if the database doesn't remain (t, α)-efficient, we can compute how many extra CPUs are needed to restore (t, α)-efficiency. In this way we can account for future grow of traffic already in the design phase.

References

CROMMELIN, C.D. [1932]. Delay probability formulae when the holding times are constant. *Post Office Electrical Engineers Journal*, **25**, 41–50.

KLEINROCK, L., AND F. MEHOVIĆ [1992]. Poisson winner queues. *Performance Evaluation*, **14**, 79–101.

KUNG, H., AND J. ROBINSON [1981]. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, **6**, 213–226.

MENASCÉ, D.A., AND T. NAKANISHI [1982]. Optimistic versus pessimistic concurrency control mechanisms in database management systems. *Information Systems*, **7**, 13–27.

MORRIS, R.J.T., AND W.S. WONG [1985]. Performance analysis of locking and OCC algorithms. *Performance Evaluation*, **5**, 105–118.

PAPADIMITRIOU, C.H. [1986]. *The Theory of Database Concurrency Control.* Computer Science Press, Rockville, Maryland.

SASSEN, S.A.E., AND J. VAN DER WAL [1996]. Two approximations for the steady-state probabilities and the sojourn-time distribution of the $M/D/c$ queue with state-dependent feedback. Technical Report COSOR 96-34, Dept. of Mathematics and Computing Science, Eindhoven University of Technology.

SASSEN, S.A.E, AND J. VAN DER WAL [1997]. The response-time distribution in a real-time database with optimistic concurrency control and exponential execution times. In V. Ramaswami and P.E. Wirth (editors), *Proceedings of the International Teletraffic Congress (ITC) 15, Washington, D.C., U.S.A., June 23–27, 1997.* North-Holland.

TIJMS, H.C. [1994]. *Stochastic Models: An Algorithmic Approach.* John Wiley & Sons, Chichester.

TIJMS, H.C., AND M.C.T. VAN DE COEVERING [1991]. A simple numerical approach for infinite-state Markov chains. *Probability in the Engineering and Informational Sciences*, **5**, 285–295.

TIJMS, H.C., M.H. VAN HOORN, AND A. FEDERGRUEN [1981]. Approximations for the steady-state probabilities in the $M/G/c$ queue. *Advances in Applied Probability*, **13**, 186–206.

WAL, J. VAN DER, AND S.A.E. SASSEN [1997]. The $M/G/\infty$ queue with OCC. Technical Report COSOR, Dept. of Mathematics and Computer Science, Eindhoven University of Technology.

WOLFF, R.W. [1982]. Poisson arrivals see time averages. *Operations Research*, **30**, 223–231.

Appendix

In this Appendix, we discuss the asymptotic behavior of the maximum throughput MT(N). What happens to MT(N) if the number of CPUs N tends to infinity?

The algorithm of Section 3.1, which computes the exact value of $p_c(N)$ (and thus the exact value of MT$(N) = Np_c(N)$) from a Markov chain, is computationally not feasible for $N \to \infty$. Moreover, the approximation we derived for $p_c(N)$ in Section 3.1 is asymptotically not applicable, because it becomes more and more inaccurate as N becomes larger.

Hence, we used the following heuristic argument to derive the asymptotic behavior of MT(N). MT(N) denotes the long-run average number of committing transactions per unit time in a closed system with N CPUs. Since every run of a transaction takes exactly 1 time unit, the probability that a run is successful equals $\sum_{n=1}^{N} P(n$ transactions commit during 1 time unit$)(1-b)^n$. For N large, we assume that this approximately equals $(1-b)^{\mathrm{MT}(N)}$, with MT$(N) = \sum_{n=1}^{N} nP(n$ transactions commit during 1 time unit). Thus, for N large, approximately $N(1-b)^{\mathrm{MT}(N)}$ transactions commit per unit time. Hence, an approximation for MT(N) with N large is found as the fixed point of the equation

$$\mathrm{MT}(N) = N(1-b)^{\mathrm{MT}(N)}. \tag{3}$$

From this equation it is immediately clear that MT(N) cannot converge to a finite value when N tends to infinity. We thus conclude that MT$(N) \to \infty$ as $N \to \infty$.

Simulations of closed systems with a large number of CPUs support this conclusion. Moreover, they showed that the solution of equation (3) for a fixed, large N gives a fairly accurate indication of MT(N). For instance, a simulation of a closed system with $b = 0.2$ and $N = 1000$ produces an MT(1000) of about 18.4. The solution of (3) in that case is 18.0.

Summarizing, if the number of CPUs N tends to infinity, the maximum throughput MT(N) also tends to infinity. However, the rate of convergence can be extremely small. As an example, having 1000 CPUs instead of 500 in a system with $b = 0.2$ yields an additional throughput of only about 2.5. Also, to go from a MT of 18 (with 1000 CPUs) to 36 we need more than 100000 CPUs!