Performance analysis of a real-time database with optimistic concurrency control

Memorandum COSOR 97-21

Performance analysis of a real-time database with optimistic concurrency control

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

Eindhoven, December 1997
The Netherlands

Performance Analysis of a Real-Time Database with Optimistic Concurrency Control

Summary

For a real-time shared-memory database with Optimistic Concurrency Control (OCC), an approximation for the transaction response-time distribution and thus for the deadline miss probability is obtained. Transactions arrive at the database according to a Poisson process. There is a limited number of CPUs that can handle transactions in parallel. Transactions have soft deadlines, and the probability of data conflicts is equal for all transactions. The response time of a transaction consists of possible waiting time (if at arrival all CPUs are occupied) plus a number of execution runs (due to the occurrence of conflicts).

In this study, we analyze the case where the execution time of all transactions is constant. Although in practice execution times are never really constant, it is important to analyze this simplifying constant case first, before trying to analyze more general execution-time distributions.

We model the real-time database (RTDB) with OCC by a multi-server queueing system with a very special kind of feedback. The probability that a transaction is 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 simulation, show that the analysis provides a good and very fast approximation for the response-time distribution and thus for the percentage of transactions that meets its deadline. We also discuss how the model and the analysis can be extended such that more realistic assumptions, e.g. non-uniform data access, several transaction types, and general execution-time distributions, can be handled.

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 (RTDB), both consistency and timeliness are important. In this paper, we investigate soft real-time databases and are interested in an *analytical* method for computing the probability that a transaction meets its deadline. By analyzing the response-time distribution (the response time is the total time between a transaction's arrival and its commit), we can easily compute the deadline miss probability for any value of a transaction's deadline. (For a *simulation* study of this probability, see Chapter 16 in KUMAR [1996].)

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 can 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, or if resources are plentiful, 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.

Despite the existence of extensive studies of the performance of OCC compared to locking, with clear recommendations as to in what situations OCC performs better than locking and vice versa, OCC is still not accepted in practice. In conventional databases and in RTDBs with soft deadlines, OCC is preferable to locking if resources (CPUs) are not the limiting factor (AGRAWAL et al. [1987]). For RTDBs with firm deadlines, where transactions that miss their deadline are discarded immediately, HARITSA et al. [1990] concluded that OCC generally performs much better than locking, even if resources are not plentiful.

Hence there seems to be no good reason why OCC is not accepted in practice. This is a motivation for us to come up with an *analysis* of OCC for RTDBs, as opposed to all existing and time-consuming *simulation* studies. We start by analyzing soft deadlines. In later work we will analyze OCC systems with firm deadlines.

So we aim for an analytical method to evaluate the performance of a RTDB with OCC and soft deadlines. The method should provide an accurate estimate of the percentage of transactions that meet their deadlines. Existing analytical performance studies for OCC (such as MENASCÉ and NAKANISHI [1982], MORRIS and WONG [1985], KLEINROCK and MEHOVIĆ [1992], and YU et al. [1993]) only consider *average* system performance, such as throughput, average response time, and the average number of restarts needed for a transaction. 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, 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 [1997b] we analyze a real-time database in which 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 ultimate goal is to analyze an RTDB with OCC where the execution time of a transaction can have any general probability distribution. In current work (SASSEN and VAN DER WAL [1997c]) we use the constant and exponential cases as building blocks in the analysis of the general case. Thus, although the assumption of constant execution times may not be very realistic, the analysis of the constant case is of vital importance for the analysis of the general case.

The rest of this paper is organized as follows. The model is explained in Section 2. In Section 3, we derive an approximative analysis 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 and a discussion of possible extensions of the analysis in order to handle more realistic assumptions.

2 The Model

In the introduction, we described the 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.

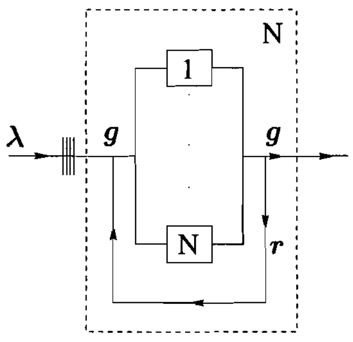


Figure 1: *Queueing model of the system*

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 (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. How the analysis can be extended to handle non-uniform data access is discussed in Section 5.

The queueing model of Figure 1 is no standard feedback model and has not been analyzed before. 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.

For an exact analysis of the queueing model, 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 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 queueing model at time t is *exactly* described by the number $w(t)$ of waiting transactions at time t, and the color $c_i(t)$ (red, green) and the remaining execution time $r_i(t)$ of the transaction at CPU i at time t, $i = 1, \dots, N$. So by the vector $(w(t), c_1(t), r_1(t), \dots$ $\dots, c_N(t), r_N(t))$. With this state-description, a *simulation* program of the system is easily made. However, an exact *analysis* seems intractable, so we are very pessimistic about the chances of finding one. Therefore, in the next section we propose an approximative analysis of the system.

3 Approximative Analysis

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 ($n = 1, \dots, N$). Let us denote this *success probability* by $p(n)$. 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 $p(n)$. The reason 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 $p(n)$. 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 $n = 1, \dots, N$).

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(a) below.

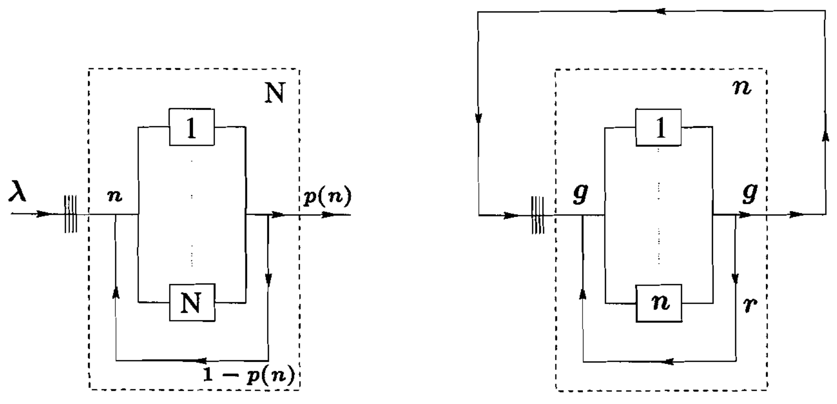


Figure 2: (a) *M/D/N queue with starting-state dependent feedback*; (b) *Closed OCC system*

We call the queueing model of Figure 2(a) an *$M/D/N$ queue with starting-state dependent feedback.* The $M/D/N$ queue with starting-state dependent feedback is not known in literature. An exact analysis seems impossible. In Section 3.2 we briefly describe our approximative analysis (details can be found in SASSEN and VAN DER WAL [1997d]). The response-time distribution derived in Section 3.2 serves as approximation for the actual real-time database we are interested in.

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

The success probability $p(n)$ in the open system of Figure 2(a) is approximated by the success probability $p_c(n)$ in a closed system with a constant number of n transactions in service, see Figure 2(b) for an illustration.

In order to compute $p_c(n)$, the closed system can be described exactly by the $(n-1)$-dimensional Markov chain $\{(c_1,\ldots,c_{n-1})_j, j = 1,2,\ldots\}$, 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. A transition occurs every time a transaction run is finished. 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.

The number of states of the Markov chain is 2^{n-1} so increases exponentially fast in n. For n not too large, computing the steady-state distribution can be done on a fast computer, but a less elaborate way to calculate $p_c(n)$ is strongly desired.

A very simple and also very accurate approximation for $p_c(n)$ is based on the assumption that the colors of successive transactions are independent, which is only approximately true. So suppose all transactions in the closed system independently have a success probability of $p_c(n)$. Now consider a transaction T. Then every transaction that validates during T's execution colors T red (makes T unsuccessful) with probability $bp_c(n)$. Since exactly $n-1$ transactions validate during T's execution, T's success probability $p_c(n)$ is the unique fixed point of the equation

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

on the interval $(0, 1]$.

Comparison of the exact and approximate value of $p_c(n)$ shows (see SASSEN and VAN DER WAL [1997d]), that the approximation is very accurate. In the special case of $n = 2$, the approximation is exact. For $n = 1$ to 10, we get the following approximate values for $p(n)$. The largest relative error made by the approximation is 0.26%, for the case of $n = 10$ and $b = 0.1$.

$b \setminus n$	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

Table 1: *Approximate success probabilities $p(n)$ for various b*

3.2 Approximation for the response-time distribution

In SASSEN and VAN DER WAL [1997d] we derived two quite accurate approximations for the response-time distribution. Because of space limitations, here we only briefly discuss one of these approximations.

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 transaction in service at some time t will have completed its run — either successfully or unsuccessfully — at time $t + D$. The transactions present at time $t + D$ are exactly those transactions that completed an unsuccessful execution during $(t, t + D]$, plus the transactions that were either waiting in queue at time t or that arrived in $(t, t + D]$. Hence, we can relate the number of transactions in the system at time $t + D$ to the number in the system at time t.

Define

$a[\ell]$ as the probability that ℓ transactions arrive in $(t, t + D]$, so $a[\ell] = e^{-\lambda D}(\lambda D)^{\ell}/\ell!$,

B_i^j as the probability that i transactions depart (are successful) during a time-interval $(0, D]$, given that j transactions are present at the start of the interval. How to find B_i^j is discussed below.

Let q_k denote the steady-state probability that k transactions are in the system. By conditioning on the state at time t, an approximation for q_k is found from the linear equations

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

It remains to specify the probability B_i^j. It is not possible to compute the exact value of B_i^j, because the system state is observed only after every D time units and not at every service start epoch. Therefore, we approximate B_i^j by $\binom{N}{i}p(N)^i(1-p(N))^{N-i}$ if $j \geq N$ and by $\binom{j}{i}p(j)^i(1-p(j))^{j-i}$ if $j < N$.

Define S as the response time of an arbitrary transaction. Let the random variable L denote the steady-state number of transactions in the system. Our approximation for the distribution of L is $\{q_k,\ k \geq 0\}$. According to Little's theorem (LITTLE [1961]), $E[L] = \lambda E[S]$. Hence, we compute our approximation for the expected response time as $E[S] = \frac{1}{\lambda}\sum_k kq_k$. To approximate the *distribution* of S we need more. We have 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 transaction run 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. For the first run of a transaction this number is obtained from the steady-state probabilities. For the later runs, we approximate the number in execution at the start of a run by the number in execution at the start of the *first* run. As a result 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. To determine the time needed for these successful services, we use the following approximation. As long as all CPUs are busy, we treat the system as a single server that works N times as fast as each of the N CPUs in the original system. Support for this method is given in SASSEN and VAN DER WAL [1997d]. Then the time between two service completions equals $\frac{D}{N}$, and 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 $p(N)$ this first service completion is successful. Then T still has to wait for $i - N$ successful service completions. With probability $1 - p(N)$, the first service completion 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 $p(N)$.

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, our approximation for the response-time distribution is

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

4 Numerical Results

4.1 Response-time distribution

We tested the approximation by comparing it 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.

We looked at systems with $N = 2, 8, 16$, and 32. 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 ≤ 0.10 and such that the average service time $E[G_N]$ in the closed system was equal to 1.05, 1.1, 1.3, 1.5, and 1.7 (according to the analysis). To explain how λ_1 was varied, we define

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

Since $p_c(n)$ is decreasing in n (see Table 1), 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 0.70 to about 0.95 were investigated. Table 2 contains a representative selection of the simulation and analysis results. For various choices of the input parameters N, b, $E[G_N]$, λ_1, and ρ_U, the table shows $E[S]$, sdev(S) (the standard deviation), $P(S > 2)$, and $P(S > 5)$.

From Table 2 we can draw a number of conclusions.

1. In all cases the approximative results are sufficiently accurate to enable the RTDB designer to judge the performance of the system.

2. The approximative analysis gives excellent results in the region where the mean number of reruns per transaction is less than or equal to 0.5, say. This corresponds with the region $Nb < 0.8$, say.

3. For utilizations close to 1 as the result of many reruns the relative errors become larger, but are still sufficiently accurate for the designer. Note however that this is just the case in which pure OCC becomes less attractive.

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).

					$E[S]$		sdev(S)		$P(S>2)$		$P(S>5)$	
N	b	$E[G_N]$	λ_1	ρ_U	App	Sim	App	Sim	App	Sim	App	Sim
2	0.050	1.05	0.76	0.80	2.02	2.02	1.29	1.28	0.37	0.36	0.037	0.036
	0.100	1.1	0.73	0.80	2.14	2.12	1.43	1.39	0.40	0.39	0.050	0.047
	0.100	1.1	0.82	0.90	3.63	3.59	2.93	2.88	0.64	0.63	0.23	0.23
	0.100	1.1	0.86	0.95	6.63	6.54	5.93	5.82	0.80	0.80	0.48	0.48
8	0.007	1.05	0.86	0.90	1.53	1.53	0.69	0.68	0.19	0.19	0.002	0.002
	0.007	1.05	0.90	0.95	2.21	2.20	1.37	1.36	0.43	0.43	0.048	0.047
	0.048	1.3	0.69	0.90	1.94	1.89	1.13	1.06	0.35	0.33	0.023	0.018
	0.084	1.5	0.53	0.80	1.64	1.61	0.95	0.89	0.22	0.20	0.010	0.008
16	0.023	1.3	0.69	0.90	1.54	1.52	0.75	0.73	0.20	0.19	0.003	0.003
	0.023	1.3	0.73	0.95	2.04	1.98	1.15	1.09	0.39	0.38	0.026	0.021
	0.040	1.5	0.60	0.90	1.78	1.73	1.00	0.95	0.28	0.26	0.013	0.011
	0.059	1.7	0.56	0.95	2.75	2.59	1.81	1.64	0.55	0.53	0.11	0.086
32	0.007	1.2	0.79	0.95	1.48	1.47	0.64	0.62	0.17	0.17	0.001	0.001
	0.020	1.5	0.63	0.95	1.87	1.81	1.03	0.99	0.33	0.31	0.016	0.013
	0.029	1.7	0.53	0.90	1.76	1.73	1.07	1.05	0.25	0.24	0.016	0.015
	0.029	1.7	0.56	0.95	2.13	2.04	1.28	1.21	0.40	0.38	0.036	0.029

Table 2: *Distribution of the response time S: analysis versus simulation*

4.2 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.

Definition

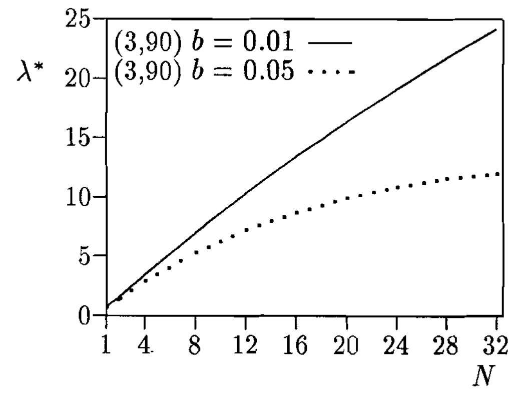
- A RTDB is (t, α)-*efficient* if at least $\alpha\%$ of the transactions meets its deadline t. Formally: $P(S \leq t) \geq (\alpha/100)$ with $0 \leq \alpha \leq 100$. We call α the *efficiency level*.
- $\lambda^*_{t,\alpha}(N, b)$ is the maximum value of the arrival intensity for which a RTDB with N CPUs and conflict probability b is still (t, α)-efficient.

For example, from Table 2 it can be seen that a RTDB with $N = 8, b = 0.007$ and $\lambda = 6.9$ is $(5, 99)$-efficient, but not $(2, 90)$-efficient.

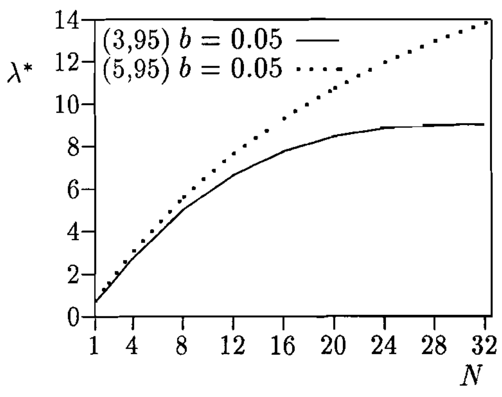
Using our approximative analysis, we did bisection to compute $\lambda^*_{t,\alpha}(N, b)$ for various N, b, t, and α. The bisectional search requires that the performance of the system has to be recomputed for many different values of λ. With our fast analytic approach, this is no problem. With simulations it would take weeks to get an answer, whereas with the analysis it is only a matter of minutes.

Figure 3(a) shows the dramatic decrease in the maximum allowable arrival intensity if the conflict probability b increases from 0.01 to 0.05. Figure 3(b) shows the drop of λ^* if the performance requirement tightens from $(5, 95)$- to $(3, 95)$-efficiency.

It is important to note that each of the graphs converges to a finite value as N gets large. Theoretical support for this can be found in SASSEN and VAN DER WAL [1997a]. For the system with $b = 0.01$ in Figure 3(a), the convergence of λ^* is not clearly visible yet.



(a) $\lambda^*_{3,90}(N, b)$ for $b = 0.01$ and 0.05



(b) $\lambda^*_{3,95}(N, 0.05)$ and $\lambda^*_{5,95}(N, 0.05)$

Figure 3: *Maximum arrival rate $\lambda^*_{t,\alpha}(N, b)$ as a function of N*

The interpretation of the flat curves at large N is that, from a certain value of N on, the real-time performance of the system cannot be improved: adding more CPUs becomes practically useless. Thus, an increase in the transaction arrival rate cannot always be resolved by adding CPUs. It is important to keep this observation in mind when designing a RTDB with OCC.

5 Concluding Remarks

In order to come to an analysis of a real-time database (RTDB) with OCC and generally distributed execution times (SASSEN and VAN DER WAL [1997c]), it is necessary to first understand the behavior of a RTDB with OCC for constant execution times. In this paper, we have presented an approximation for the response-time distribution in a RTDB with OCC, N CPUs, and constant execution times of length D. Numerical experiments for various system loads and conflict probabilities indicate, that the approximation produces sufficiently accurate estimates for the transaction response-time distribution.

This study is an essential step in deriving an analysis of a RTDB with OCC and generally distributed execution times. The response time in a system with general execution times can be approximated by an interpolation between systems with constant and with exponentially distributed execution times (the latter was analyzed in SASSEN and VAN DER WAL [1997b]). In SASSEN and VAN DER WAL [1997c], we study such an interpolation.

We are extending the analysis to non-uniform data-access patterns. That is, there can be several transaction types where every type uses a different number of data-items from (possibly) a different part of the database. Then the probability that a type-i transaction conflicts with a type-j transaction can be modeled as b_{ij}, and the success probability of a transaction will depend on its type: $p_i(n)$ for a transaction of type i.

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 RTDB in terms of (t, α)-efficiency. Using the analysis, we can for instance check whether the RTDB 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

AGRAWAL, R., M.J. CAREY, AND M. LIVNY [1987]. Concurrency control performance modeling: alternatives and implications. *ACM Transactions on Database Systems*, **12**, 609–654.

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

HARITSA, J.R., M.J. CAREY, AND M. LIVNY [1990]. On being optimistic about real-time constraints. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2–4, 1990, Nashville, Tennessee*, pages 331–343. ACM Press.

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

KUMAR, V. [1996]. *Performance of concurrency control mechanisms in centralized database systems*. Prentice-Hall, Englewood Cliffs, New Jersey.

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

LITTLE, J.D.C. [1961]. A proof of the queueing formula $L = \lambda W$. *Operations Research*, **9**, 383–387.

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 [1997a]. The $M/G/\infty$ queue with OCC. Technical Report COSOR 97-18, Dept. of Mathematics and Computer Science, Eindhoven University of Technology.

SASSEN, S.A.E, AND J. VAN DER WAL [1997b]. 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 15th International Teletraffic Congress - ITC 15, Washington, DC, USA, 22–27 June, 1997*, pages 145–156. North-Holland.

SASSEN, S.A.E., AND J. VAN DER WAL [1997c]. The response-time distribution in a real-time database with optimistic concurrency control and general execution times. Technical Report COSOR 97-21, Dept. of Mathematics and Computing Science, Eindhoven University of Technology.

SASSEN, S.A.E., AND J. VAN DER WAL [1997d]. The response-time distribution in a real-time database with optimistic concurrency control and constant execution times. Technical Report COSOR 97-07, Dept. of Mathematics and Computing Science, Eindhoven University of Technology.

YU, P.S., D.M. DIAS, AND S.S. LAVENBERG [1993]. On the analytical modeling of database concurrency control. *Journal of the ACM*, **40**, 831–872.