Performance modeling of real-time database schedulers

Eindhoven University
of Technology

Department of Mathematics and Computing Science

Memorandum COSOR 96-32

Performance modeling of real-time database schedulers *

P.D.V. van der Stok
S.A.E. Sassen
M.P. Bodlaender
J. van der Wal
A.T.M. Aerts

* To appear in Aspects of RT Databases, ed. by Sang Son, Kluwer Academic Publishers, Boston/London/Dordrecht.

Eindhoven, October 1996
The Netherlands

CONTENTS

1

PERFORMANCE MODELING OF REAL-TIME DATABASE SCHEDULERS

P.D.V. van der Stok, S.A.E. Sassen, M.P. Bodlaender, J. van der Wal and A.T.M. Aerts

Dept. of Math. and Comp. Sc., Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands

1 INTRODUCTION

Real-Time (RT) applications have grown from applications running on small microprocessors with a few Kilobytes of internal memory to applications running on a large set of interconnected processors with several Megabytes of internal memory and Gigabytes of secondary storage space (e.g. [4, 3]).

An increasing number of real-time applications acts on large sets of structured data, that have a longer lifetime than the programs accessing them. For such applications database techniques become interesting. An increasing amount of research is devoted to the area of RT databases. This research is motivated from two directions: (1) traditional database applications become subject to bounds on their response times and (2) existing RT applications have an increasing need for data modeling.

A database consists of a set of items. The database is consistent when a set of predicates over database items evaluates to True. Database applications are composed of transactions. The performance criterion for RT databases is the number of transactions that meet their deadline. Three directions can be discerned in RT database research.

1. Conditions are formulated under which the structure of the application implies serializability (e.g. [7]).
2. Application domain dependent serializability criteria are formulated that allow more concurrency. This increases the number of transactions that meet their deadline (e.g. [12]).

3. Database schedulers are constructed that handle transactions dependent on their serializability and deadline (e.g. [9]).

Our effort combines directions (2) and (3) for distributed, main memory databases, with mixed deadline types [8, 17]. Only main memory databases are considered because in RT systems only a reasonably recent state of the database is needed in case of failures and not the whole history of database states.

The Durability requirement for main memory database systems is therefore weakened with respect to more traditional databases. Mixed deadline types are important because larger RT applications are based on a mix of components with hard, firm or soft deadlines (e.g. [4]).

Contacts have been established with various companies that are interested in RT databases. Their requirements lead to the construction of database schedulers.

2 APPROACH TO PERFORMANCE CALCULATIONS

The performance of new schedulers can be compared with existing schedulers with the aid of simulations. Simulations are straightforward to construct, and provide useful insights. However, there is a major drawback.

While the construction of a simulation is fast, using the simulation to gain insight in the behavior of a scheduler is time consuming. For each unique combination of parameters (number of CPUs, system load [2], type of transaction-mix, size of database, etc.) several long test-runs are required to obtain acceptable confidence levels. Often, only a limited number of cases is simulated, which leads to incomplete and misleading results (as has been observed in [2]).

Some of the confusion (see e.g. the account given in [10]) is also caused by conclusions about properties based on mean values. As can be seen from Fig. 1, histograms with the same mean value may have very different profiles. The shape of such histograms becomes more important when we consider composite systems, where the interplay of the various components in the system leads to non-trivial performance results.

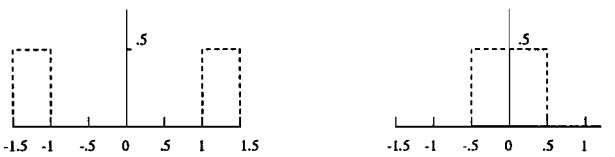

Figure 1 Histograms from which the same mean value is calculated.

Our approach in this paper is to construct mathematical models of the system environment and the scheduler in terms of queueing networks. Queueing models have their origin in the study of design problems of automatic telephone exchanges. They are widely used to evaluate the performance of (amongst other) manufacturing systems, communication systems and computer systems. In Lavenberg [1], many examples of queueing models for computer systems can be found. We use stochastic analysis to approximate the mean, variance and also higher moments of the transaction *response time* (response time is the time from transaction request to transaction termination). This allows us to estimate response time distributions. These distributions provide information about the probability that deadlines are met.

The mathematical models are parameterized, such that a wide range of situations is covered by one model. Extreme cases that are not covered by the assumptions made in the analysis are simulated and their results are used to verify the accuracy of the mathematical models.

We characterize the behavior of the system in terms of parameters, such as the number and capacity of the processors. Adding a processor then only means changing a parameter and doing a fast recomputation of the distribution function. Vice versa, we can specify requirements on the system's response time given a certain work load and then answer the question how many processors are needed to realize these results.

Obtaining the distribution function of the response time is the hard part. Known analytic performance calculations (e.g. [19, 11]) are limited to mean response times. The response time distribution is needed when the number of transactions that meet their deadline is of interest. So far, we have obtained analytical results for relatively simple schedulers that were designed to be analyzable. More complicated schedulers may require still more advanced analysis techniques. Designing schedulers that are amenable to analysis requires strong interplay between Computing Science and Operations Research. On the one hand, this approach should yield guidelines about the characteristics of ana-

lyzable schedulers and estimates about the performance increase provided by improvements on the initial analyzable scheduler. On the other hand, new Operations Research techniques are developed to enlarge the class of analyzable schedulers.

3 APPLICATION DOMAINS

Four application domains are being investigated. Most of them do not require permanent storage updates. A separation of the applications in components with different types of deadlines appears to be a promising approach.

a) Telecommunication A Private Automatic Branch eXchange (PABX) contains a database in which a.o. the signatures of PABX users are stored. The database can be distributed over several machines. A signature contains information about the different networks accessible to the user and the correspondence between a (short) user defined number and the physical telephone number within a network. RT read-only access to the database is required when the user makes a call. Less frequent and less critical updates of the database are provided.

b) High Energy Physics (HEP) The interaction of two particles in a magnetic field generates a set of particles following tracks with a certain curvature. The passage of the particles is measured by a number of detectors. The type of the interaction can be determined after the spatial reconstruction of the particle tracks from the detector data. The final reconstruction data are stored in a database. Hard deadlines on the reconstruction transactions are determined by the interaction rate and the amount of storage space available for intermediate results. Periodic transactions with soft deadlines display statistics on the accuracy of the measurements over the last few hours or visualize spatial reconstructions of particularly interesting interactions.

c) Container port Ships loaded with containers are scheduled for arrival in time slots (days) during which the quay is at their disposal. Containers, stored at predetermined locations in the ship's hold, are transported to specified storage locations on the quay (hours). Automatic Guided Vehicles (AGV) transport the containers over a predetermined route from crane to the specified storage location or vice-versa (minutes, seconds). Transactions access the database on three different time scales: (1) planning of the ships' arrivals, (2)

planning of the storage of containers once the ship arrived and (3) the almost continuous routing and collision avoidance of the AGVs.

d) Automatic Teller Machines (ATM) ATMs are linked to a number of central computers connected to a database with information on clients and their accounts. Requests for information on the account are sent with a high rate from the ATMs to the central site. Bounds on the response times of these requests are firm. On the other hand updates to the account are handled with soft deadlines. Actual developments indicate that there is a growing market for accounts with a continuous RT access especially in connection with stock transfer. However, many system states need to be recoverable.

4 TRANSACTION MODELING

The deadline class of the RT transactions is essential in determining the appropriate technique for calculating the probability that transactions will meet their deadlines. An essential question is whether all deadlines can be met. When the rate of database access requests is unbounded, the number of requests can become larger than the capacity of the supporting computer platform; the deadlines of all transactions cannot be met. A classic example is the track following of enemy fighters by a radar system. The radar system can only follow a bounded number of tracks. An overload of enemy fighters should result in a correct tracking of a maximum number of fighter planes. Deadlines of a certain number of track finders will be missed.

On the other hand the request rate can be bounded but the cost of meeting 100% of all deadlines may be prohibitively high. A lower cost solution for which a certain number of deadlines can be missed is advisable. A well informed decision about the acceptable success rate and the involved costs should be based on an analysis of the performance of the database as a function of the platform properties. In the four applications cited above the cost aspect is an important one. We have decided to use stochastic techniques to produce estimates on the performance of the proposed database as a function of different scheduler techniques and transaction properties. In the applications a), c) and d), a sizable number of the proposed transactions involves a low number of data-items and implies a short duration. Transaction preemption involves a large amount of time with respect to transaction duration. However, a high transaction request rate necessitates a computer platform with high performance. Therefore,

our first analyses are based on non-preemptable transactions with soft or firm deadlines executing on a parallel platform.

The two database scheduling strategies mentioned below are used as starting points for our performance analysis.

5 DATABASE SCHEDULING STRATEGIES

The application domains Telecommunication, HEP and ATM indicate a growing rate of short transactions. Parallel architectures seem the only solution to cope with the expected rate as explicitly stated for HEP [6]. The architecture in Fig. 2 is taken as an example. It consists of a number of processors with a local memory which communicate with each other via shared memory.

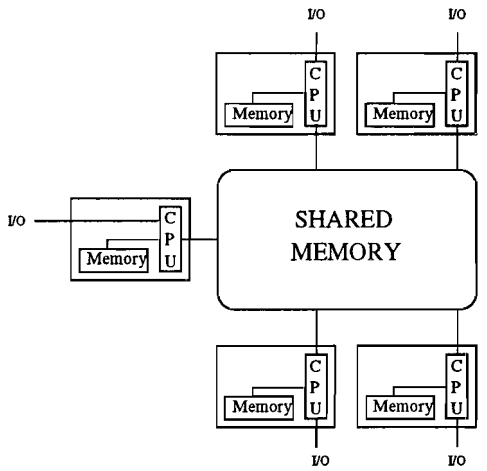

Figure 2 Parallel computer platform.

The database is stored in shared memory. One processor receives database transaction requests and stores these in shared memory. A free processor transfers a request from shared memory into its local memory. Two database scheduling strategies are investigated in more detail: Optimistic Concurrency Control (OCC) and Single Queue Static Locking (SQSL) .

5.1 OCC

The OCC scheduler assumes that the order in which transactions are committed is the serialization order. When a transaction T_i reads the value from an item X written by T_j then T_i should commit after T_j and there should be no other transaction T_k, committed between T_i and T_j, which also writes to X. Every transaction first reads all specified values and creates new values which are written to the database after validation. When at validation time the read and write order turns out to be invalid, the validating transaction is restarted. OCC schedulers look promising for RT databases [9] because the choice of the restarting transaction(s) allows the consideration of deadline criteria.

5.2 SQSL

Basic SQSL maintains database consistency by demanding that concurrently executing transactions use different data. To enforce this, the scheduler needs to know the set I_i of data items used by transaction T_i, before T_i starts its execution. When E is the set of data items in use by already executing transactions, a free processor can execute transaction T_i if I_i and E are disjoint (deadlock freedom). Transactions are executed in a First-Come, First-Served (FCFS) fashion (life-lock freedom).

Several optimizations have been applied to improve the basic SQSL scheduler. When the deadline and the execution time of each transaction are known, this information can be used to optimize the number of transactions that meet their deadlines. Transactions that cannot meet their deadline are removed from the waiting-queue, and are discarded. A nice feature of the scheduler is that a transaction that has started its execution is guaranteed to finish successfully before its deadline.

Examples of queue-handling strategies that increase performance are: (i) weakening the FCFS principle while still guaranteeing life-lock freedom, (ii) executing transaction requests according to their deadlines (Earliest Deadline First (EDF) scheduling) and (iii) early detection of unsuccessful transactions, so they are discarded at the earliest possible time.

6 OCC ANALYSIS

Transactions processed by a database go through three phases: an execution phase, a validation phase and a commit phase. In the execution phase a transaction T accesses all data-items it needs for the execution, regardless of the number of transactions already using these data-items. In the validation phase, all data-items used by T are checked on conflicts with recently committed transactions. If no conflicts occurred, T enters its commit phase. Otherwise, T has to be rerun and re-enters its execution phase.

The validation can be implemented either in a serial way or in a parallel way. For serial validation, no analysis of the average response time $E[S]$ was available. For neither serial nor parallel validation, an analysis of the response time distribution was available. In [14], we derive an approximation for $E[S]$ for OCC with serial validation. For OCC with parallel validation, we derive an approximation for the complete distribution of the response time in [13]. Both analyses are described in the next sections.

6.1 OCC with serial validation

We base our analytic approach in [14] on the model shown in Fig. 3.

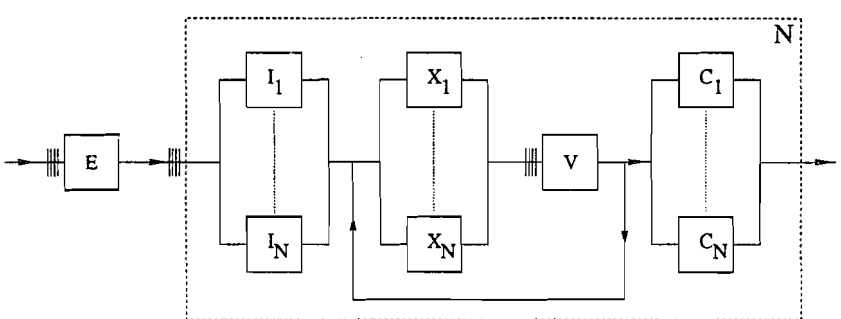

Figure 3 Queueing network for OCC with serial validation.

Arrivals of transactions are modeled as a Poisson process. Upon arrival, transactions are stored one by one in shared memory by a dedicated CPU E. Transactions wait for execution if all N CPUs are occupied. This is represented by

the queue outside the dotted box. As soon as a CPU (say u) becomes available, it retrieves the first transaction from the queue. Boxes I_u represent the initialization: copying the transaction from shared memory to the local memory of u. The execution phase of the transactions is depicted by boxes X_u, the validation phase by box V and the commit phase by boxes C_u. Initialization, execution, and commit can be done by up to N transactions concurrently. Concurrent validation is not permitted; only one transaction is allowed to validate at a time. Thus a queue of at most $N-1$ transactions can arise at box V. After validation, the transaction is restarted or committed. A committed transaction leaves the system and makes its CPU u available for a new transaction.

Although all service times in the queueing representation of Fig. 3 are taken exponentially distributed, the queueing network model does not allow for an exact analysis of the (mean) response time. An exact model would have to address the set of items accessed by each transaction and their modification history. However, this enormous state description is practically infeasible for doing computations.

Hence, an approximation is wanted for the average response time $E[S]$. A probabilistic model is used for the occurrence of data conflicts. We say that two transactions conflict if their datasets overlap. Let p be the probability that two transactions conflict. We color all transactions green on entering the enveloping dotted box. During its execution or during the time spent in the queue waiting for validation, a transaction is marked red with probability p whenever another transaction starts its commit phase. A red transaction always discovers at its validation that it cannot commit and must be rerun, a transaction that is still green at the start of its validation has had no conflicts so is allowed to commit. In addition to the assumption of probabilistic conflicts, the fake-restart assumption (see [2]) is made. At every rerun the transaction is replaced by a new, independent transaction whose execution and validation times are independent of the times in the previous run.

Due to these assumptions, it suffices to take as state description the number of red and green transactions present at every service station, and the sequence of red and green transactions present at the single-server validation station. The analysis approach is a decomposition-aggregation approach. First we approximate the mean response time of a transaction in the dotted box, treating it as a closed queueing network with a constant population of k customers (transactions). The population is kept constant by admitting a new transaction to the box as soon as another transaction has committed. For all $k \leq N$ the mean response time in the box, given a population of k customers, is calculated in subsection 3. In subsection 4 we consider the dotted box as a single service

station with a service rate dependent on the number of transactions present in front of and inside the box, such that we can approximate the mean response time of a transaction on its complete path through the system. The state-dependent service rate of the box follows from the analysis of the closed system with k transactions.

Analysis of the closed system

The analysis is simplified by taking the time needed for initialization and the time needed for commit together as one exponentially distributed variable. This leaves a closed queueing network of 3 stations: station X, V and IC. The network is shown in Fig. 4.

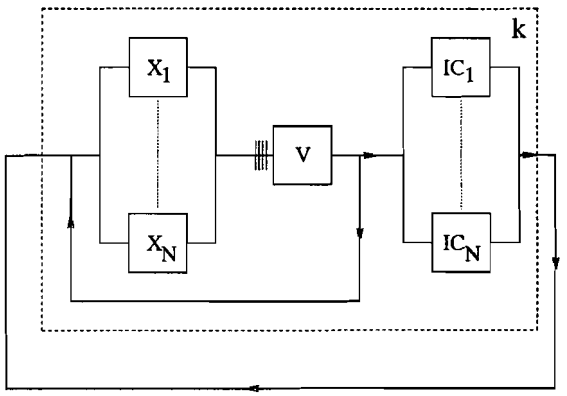

Figure 4 Closed system.

In [14], two approaches for analyzing the closed system are studied. The first (method **I**) is rather straightforward and uses a continuous-time Markov chain in which only the number of red and green transactions at the various stations are included into the state description. The actual sequence of red and green transactions at V is not modeled, and the probability that a transaction that leaves V is red is approximated by the fraction of transactions at V that is red. A large system of balance equations must be solved in order to get the steady-state probabilities of the Markov chain. Using these probabilities, the mean response time and the throughput $\mu_{Box}(k)$ of the system with fixed population k is computed. The second approach (method **II**) for analyzing the closed system is much less elaborate (with respect to computing times) and will be discussed in more detail below.

Method **II** is based on the following assumption.

Assumption:

Each transaction present at station X or in the queue of station V of the closed system is invalidated (colored red) by other transactions according to a Poisson process.

Denote the rate of the Poisson invalidation process by $\lambda(k)$ for a system with population k. In the sequel, parameter k is used to indicate a fixed population k.

The service times at X, V, and IC are exponentially distributed with service rates μ_x, μ_v, and μ_c, respectively. Define $p_{suc}(k)$ as the long-run average probability that a validation is successful, i.e., that the validating transaction is green. A transaction can be colored red while it is in its execution phase or in the queue at V. Because of the exponential service times and the Poisson invalidation assumption, the probability that a transaction is still green after its execution phase is $\mu_x/(\mu_x + \lambda(k))$. The probability that a transaction T is still green when it goes into service at station V given that it was green when it entered the queue of V is equal to $(\mu_v/(\mu_v + \lambda(k)))^{i_v}$ when i_v is the number of transactions in front of T at station V upon arrival. Hence

$$p_{suc}(k) = \sum_{i_v=0}^{k-1} P(\text{T finds } i_v \text{ tr. at V}) \left(\frac{\mu_x}{\mu_x + \lambda(k)}\right) \left(\frac{\mu_v}{\mu_v + \lambda(k)}\right)^{i_v}.$$

It remains to find an expression for $\lambda(k)$ and $P(\text{T finds } i_v \text{ tr. at V})$. Therefore, we assume that given $p_{suc}(k)$, all transactions validating have this fixed probability of success, independent of everything else in the queueing system. Then given $p_{suc}(k)$, the closed queueing network is of product form. Thus the equilibrium distribution is known and the Arrival Theorem for closed queueing networks holds. Using the state description (i_x, i_v, i_c) to denote that i_x transactions are present at station X, the steady-state distribution is

$$\pi_k(i_x, i_v, i_c) = C \frac{1}{i_x!} \left(\frac{1}{\mu_x}\right)^{i_x} \left(\frac{1}{\mu_v}\right)^{i_v} \frac{1}{i_c!} \left(\frac{p_{suc}(k)}{\mu_c}\right)^{i_c}, \tag{1.1}$$

with C the normalizing constant. From the Arrival Theorem it follows that

$$P(\text{T finds } i_v \text{ tr. at V}) = \sum_{i_x=0}^{k-1-i_v} \pi_{k-1}(i_x, i_v, k-1-i_x-i_v),$$

where $\pi_{k-1}(\cdot,\cdot,\cdot)$ is the steady-state distribution of a closed network with a population of $k-1$ customers. Thus,

$$p_{suc}(k) = \sum_{i_v=0}^{k-1} \sum_{i_x=0}^{k-1-i_v} \pi_{k-1}(i_x, i_v, k-1-i_x-i_v) \times \left(\frac{\mu_x}{\mu_x+\lambda(k)}\right)\left(\frac{\mu_v}{\mu_v+\lambda(k)}\right)^{i_v}. \tag{1.2}$$

Further, we approximate $\lambda(k)$ by

$$\lambda(k) = n_c(k-1)p, \tag{1.3}$$

where $n_c(k-1)$ is the mean number of commits per unit time in a closed system with $k-1$ customers. Given $p_{suc}(k-1)$,

$$n_c(k-1) = \sum_{i_x=0}^{k-1} \sum_{i_c=0}^{k-1-i_x} i_c \mu_c \pi_{k-1}(i_x, k-1-i_x-i_c, i_c).$$

Hence, we have a recursive procedure for computing $\lambda(k)$ and $p_{suc}(k)$. Starting with $p_{suc}(1) = 1$, $\pi_1(i_x, i_v, i_c)$ follows from (1.1), $\lambda(2)$ from (1.3), and $p_{suc}(2)$ from (1.2). The recursion is repeated until the specified k-value is reached.

The mean response time of a transaction in the closed system with population k is approximated by $E[S_{Box}(k)] = k/n_c(k)$. The throughput of the closed system, $\mu_{Box}(k)$, that is used in the next subsection to approximate the expected total response time $E[S]$ of a transaction, is given by $n_c(k)$.

The advantage of this second approach for estimating $\mu_{Box}(k)$ is that the equilibrium probabilities used are given by the explicit formula (1.1). No system of equations has to be solved in order to compute the equilibrium probabilities: this method can handle any value of k, no matter how big k is. It only takes k steps to find the approximating value for $p_{suc}(k)$.

Analysis of the open system

We use the throughput results of the closed system as input for the complete system. Transactions arrive at the system according to a Poisson process with rate λ. We now have two stations, E and Box, see Fig. 5.

Box is considered as a FCFS exponential service station with service rate $\mu_{Box}(k)$ when the number of customers in front of plus inside the box is $k < N$, and $\mu_{Box}(N)$ when the total number of customers at the box is bigger than or

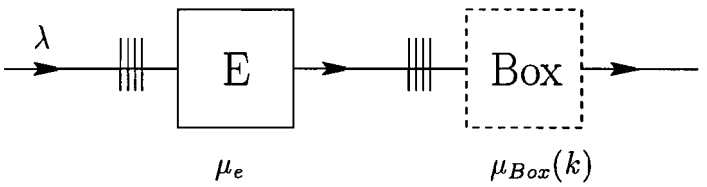

Figure 5 Aggregated system.

equal to N. E is a single exponential server with rate μ_e. Denoting the state of the aggregate system by (j_e, j_b) when j_e and j_b transactions are present at station E and station Box respectively, the steady-state probabilities are given by a product form. Let $E[L_{Box}]$ be the long-run average number of customers present in front of plus inside Box. Then

$$E[L_{Box}] = \sum_{j_b} j_b \pi(j_b),$$

where the marginal probability $\pi(j_b)$, of having j_b transactions at Box, is given by

$$\pi(j_b) = C\lambda^{j_b} \prod_{j=1}^{j_b} \frac{1}{\mu_{Box}(\min\{j, N\})},$$

with C a normalizing constant. Using Little's law, the total mean response time is approximated by

$$E[S] = \frac{1}{\mu_e - \lambda} + \frac{1}{\lambda} E[L_{Box}].$$

Numerical results

The approximations for $E[S]$ are compared with a simulation of the queueing model. For every transaction the simulation program keeps a record of its color and its time spent at the stations E, I, X, V, and C. Every time a transaction is rerun, a fresh execution time is drawn from the exponential distribution. Simulation programs were built both for the complete queueing network of Fig. 3 and for the closed queueing network of Fig. 4. For a heavy-loaded database system, the simulation results for the time spent in the dotted box in both models coincide.

Numerical results show that method **II** is an excellent method for approximating the average response time of a transaction in a system with a fixed

number of transactions. For such a closed system, method **II** is better than the time-consuming method **I**. For the open system, both methods perform well compared with simulation. Method **I** is slightly more accurate than method **II** for systems that are not overloaded. Nevertheless, method **II** is preferred because of its simplicity and its negligible computation times. Moreover, for heavy-loaded systems with a large number of CPUs, method **II** produces a much better approximation for the average response time than method **I**.

Fig. 6 shows the analysis and simulation results for $E[S]$ in a system with $\lambda = 2.4$, $\mu_e = 40$, $\mu_x = 1$, $\mu_v = 5$, and $\mu_c = 15$. The probability p that two transactions conflict is taken 0.1, which corresponds to a database with 1000 data-items where every transaction uses exactly 10 items uniformly picked from the total of 1000 items. Another example with $p = 0.1$ is a database of size 250 and transactions that all use 5 data-items. The number of CPUs N in the system is varied from 5 to 20. For $N \leq 4$, the system capacity is too small to cope with the stream of arriving transactions. For N large, the system shows a degradation in throughput because too many conflicts arise. This has a dramatic effect on the response time. Depending on the system parameters, there is a number N^* of CPUs the system should have in order to minimize the average response time. For the situation of Fig. 6, we have $N^* = 8$.

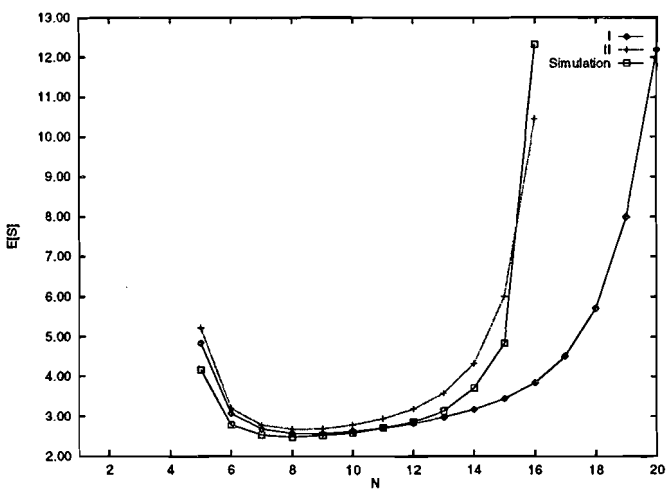

Figure 6 Results for OCC with serial validation.

An important conclusion from this study is that for applications of real-time databases with short transactions and serial validation, the duration of the val-

idation phase (including waiting) is not negligible but contributes significantly to the response time (the queue at the validation station can be quite long). To our knowledge, all previous performance studies neglected the time needed for validation.

6.2 OCC with parallel validation

In [13], the response time distribution for OCC with parallel validation is derived. The analysis is based on the throughput-analysis of [11]. The system is represented by the queueing model of Fig. 7.

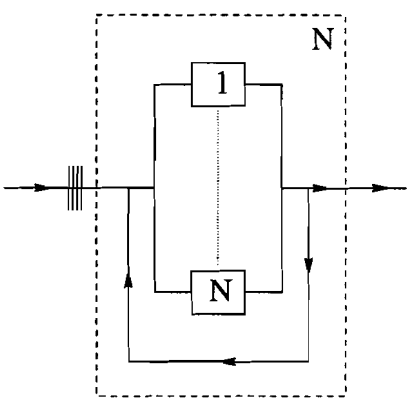

Figure 7 Queueing model for OCC with parallel validation.

Transactions arrive at the system according to a Poisson process with rate λ. In contrast to OCC with serial validation, under OCC with parallel validation the time needed for one run of a transaction does not depend on the (number of) other executing transactions. In the queueing model, the total time needed for one execution plus one validation is taken exponentially distributed with parameter μ. The time needed for the commit phase is assumed to be negligible. Hence, we can model OCC with parallel validation as a queueing network with only one multi-server station (so without a single-server validation station). At the multi-server station, both execution and validation are done.

As in the analysis of OCC with serial validation, a probabilistic model is used for the occurrence of data conflicts, with conflict probability p. The fake-restart assumption, however, is not made: the time needed for a rerun of a transaction is taken exactly equal to the time of the first run. It was possible to drop the fake-restart assumption, because the queueing model of Fig. 7 is much less

complicated (and thus better analyzable) than the model of Fig. 3.

An approximation for the response time distribution $P(S \leq t)$ is derived using a decomposition approach. First, the response time distribution $P(S_k \leq t)$ of a transaction in a closed system with fixed population k is approximated. Next, the approximations for $P(S_k \leq t)$ with $1 \leq k \leq N$ are used to approximate the distribution $P(S \leq t)$ in the open system with Poisson arrivals.

The approximation for the response time distribution in a closed system with k transactions is

$$P(S_k \leq t) = 1 - e^{-\mu t} - \int_0^t (1 - e^{-\alpha_k b x})^{\lfloor \frac{t}{x} \rfloor} \mu e^{-\mu x} \mathrm{d}x,$$

with

$$\alpha_k = \frac{\mu(1 + 2(k-1)b) - \mu\sqrt{1 + 4(k-1)b}}{2(k-1)b^2},$$

see [13] for details. To derive this approximation, we make — in accordance with [11] — the assumption that a transaction in execution observes other transactions to commit according to a Poisson process.

Since we are primarily interested in $P(S_k > t)$ (the probability that a transaction does not meet its deadline t), the approximation for $P(S_k > t)$ is compared with the value produced by a simulation of the queueing model. Relative differences of approximation compared with simulation are only a few percent for systems with $p = 0.01$ and $p = 0.1$. For $p = 0.2$, relative differences up to 10% can occur. Higher values of p were not considered, as then OCC is not the appropriate concurrency control algorithm anyway.

The extension to the open system is not treated here, but can be found in [13]. The approximation we found for $P(S > t)$ is also good compared with simulation, provided the load of the system is not too high (≤ 0.80, say). In Fig. 8, the approximative and simulated values of $P(S > t)$ are plotted for a system with 8 CPUs, $p = 0.1$, $\lambda = 2.9$ and $\mu = 1$. The throughput of the closed system with 8 CPUs (so when all CPUs are always busy) is 3.7, so the load of the system is approximately $2.9/3.7 = 0.78$.

From Fig. 8, one can easily read off the probability that a transaction does not meet its deadline t, with t ranging from 0 to 20.

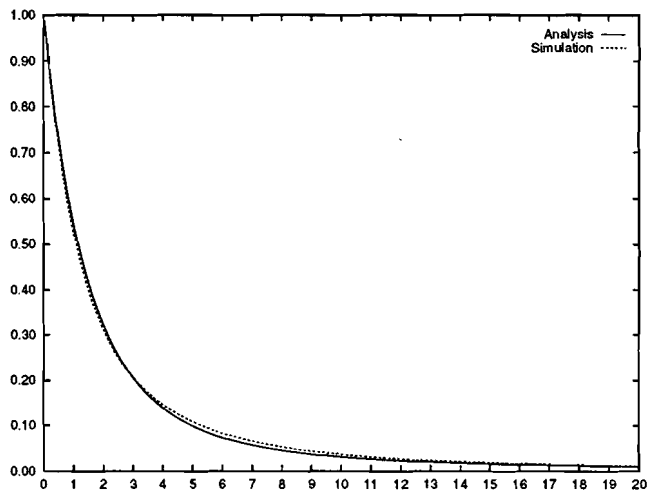

Figure 8 Results for $P(S > t)$, for OCC with parallel validation.

7 SQSL ANALYSIS

The analysis in [5] of the basic SQSL scheduler assumes a Poisson arrival process (parameter λ) and exponentially distributed execution times (parameter μ).

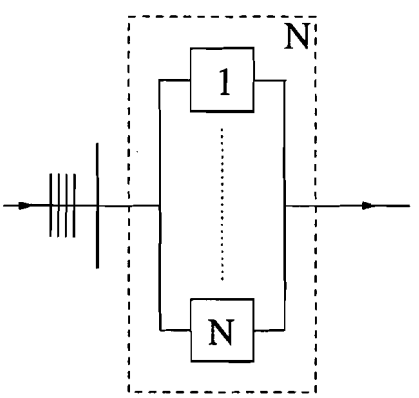

Figure 9 Queueing network for basic SQSL.

Up to N transactions can be executing at the same time, and the queue is unbounded. We assume that the database stores a fixed number d of data items.

Each transaction accesses a data items. All items have an equal probability of being accessed.

The queueing model of the SQSL scheduler is quite straightforward, as can be seen in Fig. 9. Under the given assumptions, it proved possible to completely analyze the timing behavior of the scheduler, using Markov models.

7.1 A Markov model

A continuous time Markov chain is the basis for our analysis. The assumptions stated above allow us to describe the system state by the tuple (i, j), where i is the number of executing and j the number of waiting transactions.

When the number of executing transactions is lower than the number of available CPUs ($i < N$) and the number of waiting transactions is positive ($j > 0$), the first transaction in the queue has a data conflict with at least one executing transaction. If all CPUs are executing transactions ($i = N$), the first transaction in the queue is blocked independent of data conflicts.

Some probabilities

Let $B(i)$ be the probability that a transaction T has a data conflict with one or more out of i executing transactions:

$$B(i) = 1 - \binom{d - a \cdot i}{a} \Big/ \binom{d}{a}.$$

Also, we define $B(N) = 1$, as all CPUs are occupied when N transactions can block transaction T.

If transaction T at the head of the queue has a data conflict with at least one of i executing transactions, $B(i-1 \mid i)$ is the probability that T still has a data conflict with at least one of the remaining $i - 1$ executing transactions, after one of the i executing transactions has left: $B(i - 1 \mid i) = B(i - 1)/B(i)$. We define $A(i) = 1 - B(i)$ and $A(i - 1 \mid i) = 1 - B(i - 1 \mid i)$.

The Markov property

The processing of transactions can be described by a continuous time Markov chain with state descriptor (i, j). This follows from the exponential (thus mem-

oryless) inter-arrival and execution times, the fixed number of items used by each transaction, and the fact that all items have an equal probability of being accessed. The future state of the system depends on the current state (i, j) and not on the past states: the Markov property holds.

Transitions

We analyze what state transitions are possible in the model.

First, transactions arrive at the system with rate λ. If there are no waiting transactions, and i transactions are executing, with probability $B(i)$ the arriving transaction is blocked and with probability $A(i)$ it is allowed to execute. So $(i, 0) \rightarrow (i, 1)$ with rate $\lambda B(i)$ and $(i, 0) \rightarrow (i+1, 0)$ with rate $\lambda A(i)$. When the number of waiting transactions j is greater than zero, the arriving transaction enters the queue: $(i, j) \rightarrow (i, j + 1)$ with rate λ.

Second, if $i > 0$ transactions are executing, transactions finish execution at rate $i\mu$. If the queue is empty, finished transactions are not replaced, so $(i, 0) \rightarrow (i - 1, 0)$ with rate $i\mu$.

If at least one transaction is waiting $(j > 0)$ just before a transaction completes execution, with probability $B(i - 1 \mid i)$ the first transaction f in the queue remains blocked:

$$(i, j) \rightarrow (i - 1, j) \text{ with rate } i\mu B(i - 1 \mid i).$$

With probability $A(i - 1 \mid i)$ f begins execution.

When there are l CPUs available and the first m transactions in the queue do not conflict with the transactions in execution, the scheduler permits $k = \min(m, l)$ new transactions to execute. The remaining transitions that can arise from a departure are included in the following summarizing expression:

$$(i, j) \rightarrow (i - 1 + k, j - k) \quad \text{with rate} \quad i\mu A(i - 1 \mid i) B(i - 1 + k) \prod_{z=0}^{k-2} A(i + z).$$

Note that the term $B(i - 1 + k)$ is dropped if $k = j$.

7.2 Response time distribution

The distribution of the response time S of a transaction is completely described by the moments of the response time. We aim to find $E[S^r]$, the r-th moment of S, for $r \geq 1$.

First the steady state distribution π is computed from the Markov model. This means that $\pi(i,j)$ gives the probability that the system is in state (i,j). These probabilities are used in the following analysis.

We follow the path of an arbitrary transaction through the model, from arrival to departure. With a 'path' we mean the states that are reached during the presence of the transaction under consideration. Tuple $[i,j]$ describes the situation where i transactions are in execution and j or more transactions are waiting in the queue. The tuple (i,j) refers to the system state as defined before. Define $S_{[i,j]}$ as the time until a transaction T leaves the system, when i transactions are executing, and $j-1$ transactions are ahead of T in the queue. If $j=0$, the transaction under consideration is in execution. When the system is in state (i,j) after an arrival, $S_{[i,j]}$ is the response time of the newly arrived transaction.

Important is the observation that $S_{[i,j]}$ does not depend on transactions that arrive at the system after the transaction under consideration. This follows from the property of the Single-Queue, Static-Locking scheduler: transactions waiting in the queue cannot be overtaken.

Consequently, arrivals of other transactions need not be considered when $E[S^r_{[i,j]}]$ is computed. Let X_i be the time till the next departure when i transactions are executing (X_i is exponentially distributed with rate $i\mu$). Let $p_{[i,j][m,\ell]}$ be the probability that the next departure leads to a state with m transactions in execution and $\ell-1$ transactions present in the queue ahead of the transaction under consideration. From the transitions in the Markov model we have

$$p_{[i,j][m,\ell]} = \begin{cases} A(i-1 \mid i)B(m)\prod_{z=0}^{m-i-1} A(i+z) & i \leq m < N,\ \ell = j+i-1-m \\ B(i-1 \mid i) & j>0,\ m=i-1, \ell=j \\ A(i-1 \mid i)\prod_{z=0}^{m-i-1} A(i+z) & m=N \vee \ell = 0 \\ 1 & j=0,\ m=i-1,\ \ell=j \\ 0 & \text{otherwise.} \end{cases}$$

Then for all $[m, \ell]$:

$$S_{[i,j]} = X_i + S_{[m,\ell]} \quad \text{with probability} \quad p_{[i,j][m,\ell]}.$$

As $m+\ell = i+j-1$, the moments of $S_{[i,j]}$ can be computed from the moments of $S_{[m,\ell]}$ with $m+\ell < i+j$. Once a transaction is in execution, its service time is exponentially distributed with mean $1/\mu$. Thus the boundary condition for the recursion is $S_{[i,0]} = X$ for all $i > 0$, where X is exponentially distributed with parameter μ.

Let $a_{(r,\ell)(i,j)}$ be the probability that a transition to state (i, j) is caused by an arbitrary transaction T that sees state (r, ℓ) on arrival. An expression for T's response time S is found by conditioning on state (r, ℓ) and by using the PASTA [18] property:

$$S = S_{[i,j]} \quad \text{with probability} \quad \sum_{(r,\ell):i+j=r+\ell+1} \pi(r,\ell) a_{(r,\ell)(i,j)}.$$

The probability $a_{(r,\ell)(i,j)}$ is given by

$$a_{(r,\ell)(i,j)} = \begin{cases} \lambda A(r) & \ell = 0,\ i = r+1,\ j = 0 \\ \lambda B(r) & \ell = 0,\ i = r,\ j = 1 \\ 1 & \ell > 0,\ i = r,\ j = \ell + 1 \\ 0 & \text{otherwise.} \end{cases}$$

Moments of the response time

The moments of the response time are derived directly from the recursive relation. Two important rules are used to find $E[S^r]$ for $r \geq 1$:

- **Choice.** The transaction follows path l with r-th moment $E[S_l^r]$, or path m with r-th moment $E[S_m^r]$. The probability that path l is taken is p. Then
$$E[S^r] = pE[S_l^r] + (1-p)E[S_m^r].$$
- **Addition.** The transaction first follows path l with duration S_l, followed by path m with duration S_m. Then
$$E[S^r] = E[(S_l + S_m)^r].$$

Based on these rules, the moments of S can be found using dynamic programming. Note that the analysis produces the exact values of the moments $E[S^r]$.

Fitting a distribution to the moments

In [15] it is proved that each positive random variable can be approximated arbitrarily well by a weighted sum of independent exponentially distributed variables. We used this result to find a mixture of exponentially distributed variables that has the same moments as S. The choice of this mixture influences the quality of the approximation. Denote the random variable corresponding to the chosen mixture by $\hat{S}$. Then $P(S \leq x)$, the probability that a transaction meets its deadline, is approximated by $P(\hat{S} \leq x)$. We say the distribution of $\hat{S}$ is fitted to the moments of S.

We used the two-moment fit as described in [16]. The fitting procedure is not given here for reasons of brevity, but it can fit a distribution to any combination of $E[S]$ and $E[S^2]$.

Simulation versus fitting

Parameters $N = 4$, $\lambda = \frac{5}{3}$ and $\mu = 1$ were used to compare our fit results with simulation results. We used moments $E[S]$ and $E[S^2]$ from our analysis to approximate $P(S \leq x)$ for $x = 1$, 3 and 5.

	$P(S \leq 1)$		$P(S \leq 3)$		$P(S \leq 5)$	
$B(1)$	Fit	Sim.	Fit	Sim.	Fit	Sim.
0	0.61	0.61	0.95	0.95	0.99	0.99
0.010	0.59	0.59	0.94	0.94	0.99	0.99
0.039	0.54	0.54	0.90	0.90	0.98	0.98
0.088	0.45	0.45	0.83	0.83	0.95	0.95
0.153	0.32	0.32	0.68	0.68	0.85	0.85
0.230	0.16	0.17	0.41	0.42	0.59	0.60

Table 1 Response time probabilities.

Conflict probability $B(1)$ was varied from 0 to 0.230, corresponding with $a = 0$ to $a = 5$ in a database with $d = 100$. We also estimated $P(S \leq x)$ by simulation. Table 1 shows these numbers. The simulated values are the midpoints of a 95% confidence interval with a width smaller than 0.02. It is clear from Table 1 that the fitting procedure gives an excellent approximation of the response time distribution.

8 CONCLUSIONS

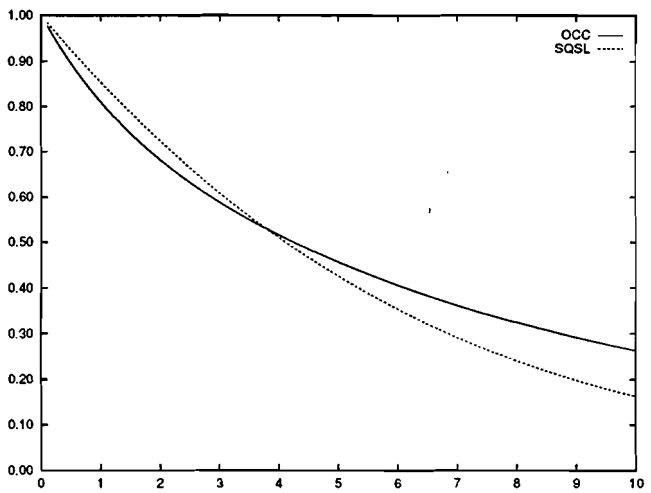

Figure 10 $P(S > t)$ for OCC and SQSL, for $N = 4$, $\lambda = 2$, $a = 4$, $d = 100$.

Several application areas have been identified where RT database techniques are of interest: Telecommunication, High Energy Physics, Container Port and Automatic Teller Machines. In these areas, applications are often divided in an RT part composed of short transactions with soft or firm deadlines and another part which should not perturb the first part.

Not only the mean response time, but the entire response time distribution of the RT transactions is necessary to calculate the probability that these transactions will meet their deadline. The variance and higher moments of the response time are used to obtain approximations of the response time distribution.

Analytic work has been done for the calculation of the response time distribution of transactions. The OCC analysis shows that the validation time cannot be neglected for short transactions that occur for example in telecommunication applications. The basic SQSL scheduler has been satisfactorily analyzed. The analysis of the more complex OCC scheduler is nearing completion. Once the analysis of the OCC scheduler is complete, we plan an extensive comparison between these two schedulers.

Such a comparison is then easy, as the analysis results allow fast recomputations for a wide range of parameters. Fig. 10 is an example of a comparison between OCC and SQSL for one set of parameters. Actually, Fig. 10 shows the results

for a large number of experimental settings, where the deadline-length is varied between 0 and 10. It is interesting to see that (for these specific parameters) a break-even point exists. OCC outperforms SQSL when deadlines are short, and SQSL outperforms OCC when deadlines are long.

REFERENCES

[1] S.S. Lavenberg. Computer Performance Modeling Handbook. Academic Press, Orlando, Fla., 1983.

[2] R. Agrawal, M.J. Carey, and M. Livny. Concurrency Control Performance Modeling: Alternatives and Implications. *ACM Transactions on Database Systems*, 12(4):609–654, December 1987.

[3] J. Altaber, P.G. Innocenti, and R. Rausch. Multiprocessor architecture for the LEP storage ring. In *6th Annual Workshop on Distributed Computer Control Systems*, Monterey, May 1985. IFAC.

[4] Committee Arinc 651. ARINC report 651, draft 9. Technical Report 91-207/SAI-435, Airlines Electronic Engineering Committee, September 1991.

[5] M.P. Bodlaender, S.A.E. Sassen, P.D.V. van der Stok, and J. van der Wal. The Response Time Distribution in a Multiprocessor Database with Single Queue Static Locking. In *Proc. of the Workshop on Parallel and Distributed RT Systems*, pages 118–121, Hawaii, April 1996.

[6] *Proc. of the 8th Conference on Computing in High Energy Physics*, Santa-Fé, USA, 1990.

[7] M.H. Graham. How to get serializability for real-time transactions without having to pay for it. In *Proceedings of the 14th Real-Time Systems Symposium*, pages 56–65. IEEE, December 1993.

[8] D.K. Hammer, E.J. Luit, P.D.V. van der Stok, J. Verhoosel, and O.S. van Roosmalen. DEDOS: A Distributed Real-Time Environment. *IEEE Parallel & Distributed Technology*, 2(4):32–47, 1994.

[9] J. Lee and S.H. Son. Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems. In *Proceedings of 14th Real-Time Systems Symposium*, pages 66–75, Raleigh-Durham, December 1993. IEEE.

[10] J. Lee and S.H. Son. Performance of Concurrency Control Algorithms for Real-Time Databases. In V. Kumar, editor, *Performance of Concurrency Control Mechanisms in Centralized Database Systems.* Prentice-Hall, 1996.

[11] R.J.T. Morris and W.S. Wong. Performance Analysis of Locking and OCC Algorithms. *Performance Evaluation*, 5:105–118, 1985.

[12] K. Ramamritham and C. Pu. A Formal Characterization of Epsilon Serializability. Technical Report COINS Tech. rep. 91-92, Univ. of Mass., 1991.

[13] S.A.E. Sassen and J. van der Wal. The Response Time Distribution in a Real-Time Database with Optimistic Concurrency Control. Technical Report COSOR 96-17, Eindhoven University of Technology, 1996.

[14] S.A.E. Sassen, J. van der Wal, and M.P. Bodlaender. Mean Response Times for Optimistic Concurrency Control in a Multi-Processor Database with Exponential Execution Times. Technical Report COSOR 95-43, Eindhoven University of Technology, 1995.

[15] R. Schassberger. *Warteschlangen.* Springer Verlag, 1973.

[16] H.C. Tijms. *Stochastic Models, an Algorithmic Approach.* John Wiley & Sons, Chichester, 1994.

[17] P.D.V. van der Stok. Real-Time Distributed Concurrency Control Algorithms with Mixed Time Constraints. Technical Report CSN 96/18, Eindhoven University of Technology, 1996.

[18] R.W. Wolff. Poisson Arrivals see Time Averages. *Operations Research*, 30:223–231, 1982.

[19] P.S. Yu, D.M. Dias, and S.S. Lavenberg. On the Analytical Modeling of Database Concurrency Control. *Journal of the ACM*, 40:831–872, 1993.