Approximate Analysis of Real-Time Database Systems

Jayant R. Haritsa *
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560012, INDIA

Abstract

During the past few years, several studies have been made on the performance of real-time database systems with respect to the number of transactions that miss their deadlines. These studies have used either simulation models or database testbeds as their performance evaluation tools. We present here a preliminary analytical performance study of real-time transaction processing. Using a series of approximations, we derive simple closed-form solutions to reduced real-time database models. Although quantitatively approximate, the solutions accurately capture system sensitivity to workload parameters and indicate conditions under which performance bounds are achieved.

1 Introduction

In a broad sense, a real-time database system (RT-DBS) is a transaction processing system that is designed to handle workloads where transactions have service deadlines. The objective of the system is to meet these deadlines, that is, to process transactions before their deadlines expire. Therefore] in contrast to a conventional DBMS where the goal usually is to minimize transaction response times, the emphasis here is on satisfying the timing constraints of transactions.

Transactions may miss their deadlines in a real-time database system due to contention for physical resources (CPUs, disks, memory) and logical resources (data). During the last few years, several detailed studies [1, 3, 5] have evaluated the performance of various real-time transaction resource scheduling policies with respect to the number of missed transaction deadlines. These studies have either used simulation models [1, 3] or used database testbeds [5] as their performance evaluation tools. The lack of *analytical* studies may be attributed to the complexity of real-time database systems. Accurately modeling a real-time database system involves incorporating transaction time constraints, scheduling at multiple resources, concurrency control, buffer management] etc., and this appears to be mathematically intractable. In fact, the exact solutions to extremely simplified special cases are themselves complex (e.g. [8]).

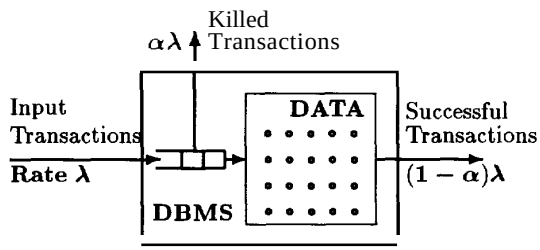

Figure 1.1: RTDBS Model

While exact solutions appear infeasible or too complex to be of utility, we show in this paper that it is possible to derive simple ***approximate*** solutions to reduced models of real-time database systems. Although the solutions are quantitatively approximate, they satisfactorily capture system sensitivity to workload parameters and indicate conditions under which performance limits are achieved. In essence, we are able to estimate performance ***trends*** and ***bounds.***

We investigate here the performance of real-time database systems where transactions have deadlines to the ***start*** of service (i.e. laxities). In our reduced model (Figure 1.1), transactions arrive in a stream to the real-time database system. Each transaction upon arrival requests the scheduler for access (read or write) to a set of objects in the database. A transaction that is granted access to its data before its laxity expires is considered to be "successful". Successful transactions access their data for some period of time and then exit the system. Transactions that are not successful are "killed", that is, they are removed from the system wait queue when their laxities expire. Our goal is to

*This work was initiated while the author was with the Systems Research Center, Univ. of Maryland (College Park) and was supported in part by a SRC Post-Doctoral Fellowship.

derive the steady-state fraction of input transactions that are killed (α in Figure 1.1), as a function of the workload and system parameters. We consider only data contention in our model since it is a ***fundamental*** performance limiting factor, unlike hardware resource contention which can be reduced by purchasing more resources and/or faster resources. While abundant resources are usually not to be expected in conventional database systems, they may be more common in RT-DBS environments since many real-time systems are sized to handle transient heavy loading.

Using a series of approximations, we develop here a simple closed-form solution for the above RTDBS model, which merely involves finding the roots of a cubic equation. This approximate solution accurately captures the qualitative behavior of the model. Further, it also provides quantitative results that are fairly close to the exact values (as determined by simulation). Taking advantage of the simplicity of the approximate solution, we derive interesting corollaries, some of which are unique to the database environment. For example, we show that the absolute values of certain database parameters play a role in determining system performance, unlike the corresponding classical real-time systems where performance is determined solely by *normalized* quantities.

2 Model and Notation

We consider a system where transaction arrivals are Poisson with rate A, transaction data processing times are exponentially distributed with mean $1/\mu$, and transaction laxities are (independently) exponentially distributed with mean $1/\gamma$ ($A, \mu, \gamma > 0$). We assume that the database is large, that it is accessed uniformly, and that each transaction atomically requests its entire data set (i.e. static locking or predeclaration [9]). We also assume that each transaction requests J data objects and that J is much smaller than N, the database size (this is usually true in practice).

The database scheduler queues and processes transactions in arrival order. A transaction is allowed access to its data only if it has no data conflicts with currently executing transactions ***and*** if all transactions that arrived prior to it have either been successful or been killed. While this type of ***fcfs*** policy is not typical of real-time systems, there are database situations, however, where this policy may be used due to fairness requirements. A practical example is that of brokers submitting real-time buy and sell orders in a stock exchange, wherein ***fcfs*** processing may be used to maintain fairness among brokers. In addition, a ***fcfs*** policy provides a baseline against which more sophisticated real-time scheduling disciplines can be evaluated.

In the subsequent discussions, we use α ($0 \leq \alpha \leq 1$) to denote the steady-state fraction of input transactions that are killed. To succinctly characterize our system configuration, we use the queueing-theoretic notation $M/M/N_J/M$, where the first M denotes the Poisson transaction arrival process, the second M denotes the exponential transaction service time distribution, N_J denotes the number of servers, and the last M denotes the exponential transaction laxity distribution. In typical queueing systems, the number of servers is usually constant. However, in the database environment, the number of "servers", that is, the number of transactions that can be simultaneously processed, is ***variable*** depending on N, J, and the current sequence of transaction data requests (i.e. the level of data contention). We therefore use the notation N_J for the server descriptor, thereby highlighting the variability in the number of servers. With this convention, our real-time database model is represented by a $M/M/N_J/M$ queueing system, and our goal is to characterize the α behavior of this system.

3 Related Work

There is an extensive literature on the analysis of queueing systems with deadlines. In particular, queueing systems such as $M/M/1/M$ and $M/M/m/M$ have been solved exactly with respect to the α metric [2, 11]. However, these results are applicable only to systems with a constant number of servers. They are not useful for determining the performance of our queueing model since the number of servers in the database is variable, as explained in Section 2.

Database systems where queueing is not allowed were considered in [8, 7]. In these systems, a transaction that cannot receive service as soon as it arrives is immediately killed (equivalently, all transactions have zero laxity). The exact solution for this model was shown to be quite complex in [8] and approximations to the solution for large databases were presented in [7, 8]. In our model, where queueing is included, the situation becomes more complicated, especially since the number of servers is variable.

The performance of locking protocols in database systems has been extensively analyzed. However, virtually all of these studies (e.g. [9, 10]) have been been made in the context of conventional database systems where transactions do not have service deadlines. Therefore, their results are not directly applicable to the real-time environment.

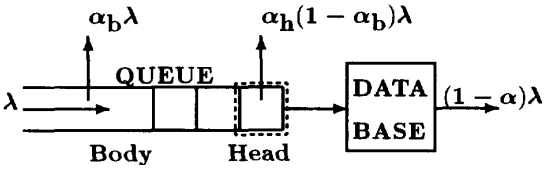

Figure 4.1: Queueing Model

4 Analysis of $M/M/N_J/M$

In this section, we present an approximate solution to the $M/M/N_J/M$ queueing system described in Section 2. Our solution is in two parts: First, we characterize α_h, the steady-state fraction of transactions that successfully manage to reach the ***head*** of the transaction wait queue but are killed while waiting for their data conflicts to disappear. Next, we compute α_b, the steady-state fraction of transactions that are killed before they reach the head of the queue, that is, while they are in the ***body*** of the queue. These quantities are related to the overall a of the system by the following equation (derived by elementary flow analysis of Figure 4.1 which shows the queueing model)

$$1-\alpha = (1-\alpha_h)(1-\alpha_b) \tag{1}$$

Therefore, if we are able to separately compute the "head-of-queue" and "body-of-queue" performance statistics, we can then easily derive the overall system performance. Our motivation for taking this two-step approach is to ***decouple*** the data conflict analysis from the queueing analysis and thereby simplify the performance modeling.

In the following derivations, we refer to $p = \lambda/\mu$ as the system offered load, and to $6 = \mu/\gamma$ as the normalized mean laxity (following the terminology of [11]). Further, we refer to $\xi = J/N$ as the database access ratio. For ease of explanation, we initially derive results for the case where transactions access their data objects only in write (exclusive lock) mode. Later, in Section 7, these results are extended to the situation where data is accessed in both read and write modes.

4.1 Head-Of-Queue Performance

In this section, we compute α_h $(0 \leq \alpha_h \leq 1)$, the probability that a transaction which has successfully managed to reach the head of the queue is killed while waiting in this position.

Lemma 1 *The value of α_h is approximately given by*

$$\alpha_h = A(1-\alpha) \tag{2}$$

where the coefficient $A = \frac{\rho\xi_J}{1+\delta}$.

Proof: Consider a transaction that reaches the head of the queue when k database objects are currently locked and finds that some of the data objects it requires are in this locked set (i.e. the transaction has data conflicts). The probability that this transaction is killed while waiting for the conflicting locks to be released is given by

$$\alpha_{h|k} = \sum_{i=1}^{J} P_{con|k,i}\, P_{exp|i} \tag{3}$$

where $P_{con|k,i}$ is the probability that the transaction conflicts on i of its requested J objects, and $P_{exp|i}$ is the probability that the transaction's laxity expires before these i objects are released.

We approximately model the head waiter's request of J data items from the N database objects as J samplings with replacement, that is, as a sequence of Bernoulli trials. In this situation, the probability of exactly i conflicts is given by

$$P_{con|k,i} = \binom{J}{i} \left(\frac{k}{N}\right)^{i} \left(1-\frac{k}{N}\right) \tag{4}$$

since the probability of requesting an already locked item is $\frac{k}{N}$.

We next compute $P_{exp|i}$, which is the probability that the head waiter's laxity expires before all of its i conflicting locks are released. Due to the assumption of uniform access to the database and since $J \ll N$, the probability of having more than one conflict with the same transaction is small. We therefore assume that each of the i conflicts occurs with a different transaction. The cumulative distribution of the maximum of i identically-distributed exponential variables with parameter μ is given by

$$F_{max(i)}(t) = (1-e^{-\mu t})^{i}$$

Only values of t that are greater than the remaining laxity of the waiting transaction have to be considered and since the expression $e^{-\mu t}$ tends to 0 with increasing t, we make the approximation that

$$F_{max(i)}(t) \approx (1-ie^{-\mu t})$$

Since transaction laxities are exponentially distributed (with parameter γ), and by virtue of the memoryless property of exponential distributions, we obtain

$$P_{exp|i} = \int_0^\infty ie^{-\mu t}\,\gamma e^{-\gamma t}\,dt = \frac{i\gamma}{\mu+\gamma} = \frac{i}{1+\delta} \tag{5}$$

Substituting the above results in Equation 3 gives

$$\begin{aligned}\alpha_{h|k} &= \sum_{i=1}^{J} \binom{J}{i} \left(\frac{k}{N}\right)^{i} \left(1-\frac{k}{N}\right)^{J-i} \left(\frac{i}{1+\delta}\right) \\ &= \frac{kJ}{N(1+\delta)} \sum_{i=1}^{J} \binom{J-1}{i-1} \left(\frac{k}{N}\right)^{i-1} \left(1-\frac{k}{N}\right)^{J-i} \\ &= k\frac{\xi}{1+\delta} \qquad (6)\end{aligned}$$

since the second summation is identically equal to 1

We now go on to compute α_h, the *unconditional* probability that a head-of-queue waiter is killed. Using P_k to denote the probability of k objects being locked, α_h can be expressed as

$$\alpha_h = \sum_k \alpha_{h|k} P_k = \sum_k k\frac{\xi}{1+\delta} P_k = \frac{\xi}{1+\delta} E(k) \qquad (7)$$

Here, $E(k)$ is the average number of locked objects and is easily computed using Little's formula [6]. The rate at which transactions obtain locks is $\lambda(1-\alpha)J$ and locks are held for a mean duration of $1/\mu$. It therefore follows from Little's formula that

$$E(k) = \frac{\lambda(1-\alpha)J}{\mu} = pJ(1-a) \qquad (8)$$

Combining Equations 7 and 8, we finally obtain

$$\alpha_h = \frac{\xi}{1+\delta}\rho J(1-\alpha) = \frac{\rho\xi J}{1+\delta}(1-\alpha)$$

□

Note that in the above derivation, a series of approximations were made to obtain a simple expression for α_h. The expression is asymptotically exact as $N \to \infty$.

4.2 Body-Of-Queue Performance

In this section, we compute α_b ($0 \leq \alpha_b \leq 1$), the steady-state probability that a transaction in the queue is killed before reaching the head of the queue, that is, while it is in the body of the queue.

Lemma 2 *The value of α_b is a unique root of the cubic equation*

$$\alpha_b^3 + B\alpha_b^2 + C\alpha_b + D = 0 \qquad (9)$$

where the coefficients B, C, and D, are given by

$$\begin{aligned}B &= \frac{1}{\rho\delta}\left(1+\frac{1}{1+\delta}\right) - \left(1+\frac{6}{1+\delta}\right) \\ C &= \left(1-\frac{2}{1+\delta}\right) - \frac{1}{\rho\delta}\left(1+\frac{1}{1+\delta}\right) - \frac{1}{\rho^2\xi J}\left(1+\frac{2}{\delta}\right) \\ D &= \frac{1}{1+\delta}\end{aligned}$$

Over the range of valid parametric values, the equation has exactly one root on [0,1] – this is the required root.

Proof: A detailed proof of this lemma is given in the Appendix. Here, we will sketch the outline of the proof. The basic idea behind our solution is to treat the transaction wait queue *itself* as an $M/G/1$ system with the head of queue position playing the role of the "server". That is, we treat the wait queue as being composed of a (pseudo)server and a secondary queue. As shown in the Appendix, it is possible to express the "service-time" distribution of this system (i.e. the distribution of the time spent at the head-of-queue position) with the following equation

$$f_h(t) = (1-E)u_0(t) + E(\mu+\gamma)e^{-(\mu+\gamma)t} \qquad (10)$$

where $E = \rho\xi J(1-\alpha_b)(1-a)$ and $u_0(t)$ is the impulse function [6].

From Equation 10, we infer that a fraction $(1-E)$ of the input transactions have a service time of zero while the remainder have an exponentially distributed service time with parameter $(\mu+\gamma)$. The transactions that have a service time of zero are those that are killed before they reach the head of the queue and those that immediately enter the database on reaching the head of the queue. The remaining transactions either enter the database after waiting for some time at the head of the queue or are killed during their wait at the head of the queue.

In [6], formulas for computing the waiting time distribution of M/G/1 queues are given in terms of the service-time distribution. Substituting the service-time distribution from Equation 10 in these formulas, the cumulative distribution function of the waiting time in the body of the queue works out to

$$F_w(t) = 1 - Ge^{-(\mu+\gamma)(1-G)t} \qquad (11)$$

where $G = \frac{\rho^2\delta\xi J}{1+\delta}(1-\alpha_b)(1-a)$.

Recall that α_b is (by definition) the fraction of transactions that are killed because their laxity is smaller than their waiting time in the body of the queue. Therefore,

$$\alpha_b = \int_0^\infty (1-F_w(t))\,\gamma e^{-\gamma t}\,dt = \frac{G}{2+6-G(1+6)} \qquad (12)$$

After substituting for G, the above equation expresses α_b in terms of the system input and output parameters. Using this equation in conjunction with Equations 1 and 2, and after some algebraic manipulations, we finally arrive at the cubic equation described in the lemma. The proof that this equation has only a single root in [0,1] is given in [4].

□

An important point to note here is that the above derivation is approximate. This is because the $M/G/1$ queueing results that were used in the derivation assume independence between the task arrival process and the service time distribution. In our case, however, the head-of-queue "service-time" distribution (Equation 10) is *dependent* on the task arrival process since it involves terms (e.g. p) that are a function of the arrival process.

4.3 System Performance

In this section, we combine the results derived above for the head-of-queue and body-of-queue statistics to compute a ($0 \leq a \leq 1$), the overall fraction of killed transactions.

Theorem 1 *For the $M/M/N_J/M$ system, the steady-state fraction of transactions that are killed is approximately given by*

$$\alpha = 1 - \frac{(1-\alpha_b)}{1 + A(1-\alpha_b)} \qquad (13)$$

where α_b is obtained from Lemma 2, and $A = \frac{\rho \xi J}{1+\delta}$ is the coefficient derived in Lemma 1.

Proof: The above expression for a is obtained by combining Equations 1 and 2.

□

From the above results, we observe that the performance of an $M/M/N_J/M$ real-time database system is determined by $p, 6, \xi$ and J. This is in contrast to classical $M/M/1/M$ real-time systems where the system performance is dependent only on p and 6 [11].

5 Quality of Approximations

In this section, we compare the accuracy of the approximate analysis with respect to the exact solution, as determined by simulation' of the queueing system. In Figures 5.1 through 5.4, we plot a, the fraction of killed transactions, as a function of p, the system load, for different combinations of 6 (the normalized mean laxity) and ξ (the database access ratio). The transaction size, J, is set to 10 in these experiments, and all data objects are requested in write (exclusive lock) mode. Four different values of ξ, which span the range from a large-sized database to an extremely small database were considered. The chosen ξ values were 0.0001, 0.001, 0.01 and 0.1, which correspond to database sizes of 100000, 10000, 1000, and 100 respectively. Note that while $t \ll 1$ was assumed in the analysis, the performance for larger values of ξ was also evaluated in order to observe at what stage the analysis broke down when the assumptions were not satisfied.

For each of the ξ settings, we evaluated the α performance for three values of 6, the normalized laxity. The selected 6 values were 0.1, 1.0 and 10.0, thus covering a spectrum of transaction slack times ($\delta = 0.1$ corresponds to transaction laxities being small compared to processing times, $6 = 1.0$ makes the laxities comparable to processing times, and $6 = 10.0$ results in laxities that are much greater than processing times).

In Figure 5.1, which captures the large database situation, we observe that under light loads, the analytical solution (solid lines) provides an excellent approximation to the exact solution (broken lines) for all the laxities. At heavier loads, the quantitative matching deteriorates to some extent (for the large laxity case), but the qualitative agreement is maintained throughout the entire loading range. This experiment confirms that, for large databases, the simple cubic approximation is a good estimator of system performance.

The above experiment is repeated for progressively decreasing database sizes in Figures 5.2 through 5.4. From these figures, it is clear that the approximations provide reasonably accurate performance predictions until ξ goes above 0.01. Further, even when ξ is as large as 0.1 (Figure 5.4), the *qualitative* agreement between the analysis and the exact solution remains very close. Therefore, although our analytical solution is heavily based on the assumption that the database is large, it captures system performance trends for smaller-sized databases as well.

'All α simulation results in this paper show mean values that have relative half-widths about the mean of less than 5% at the 95% confidence level.

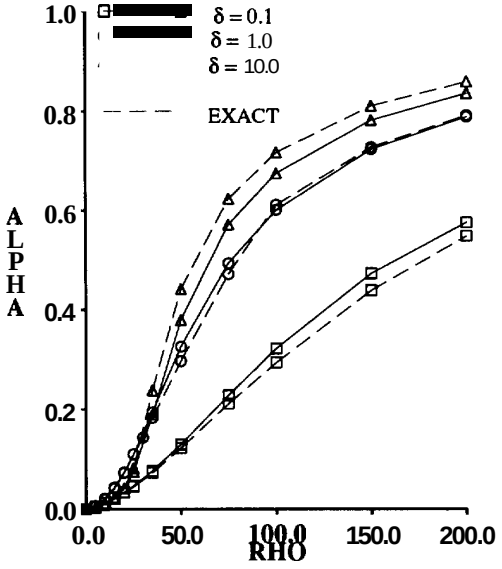

Figure 5.1: $\xi = 0.0001$

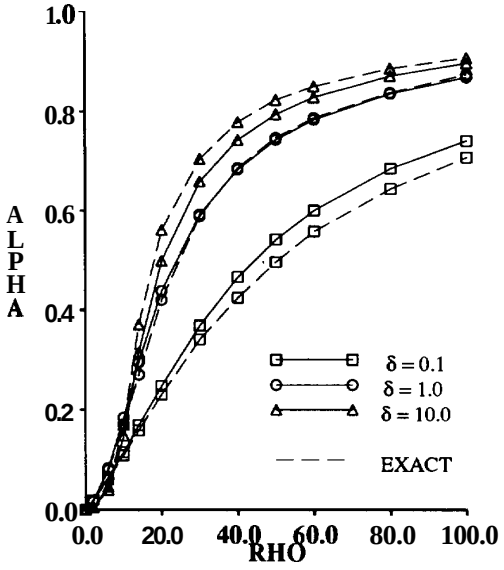

Figure 5.2: $\xi = 0.001$

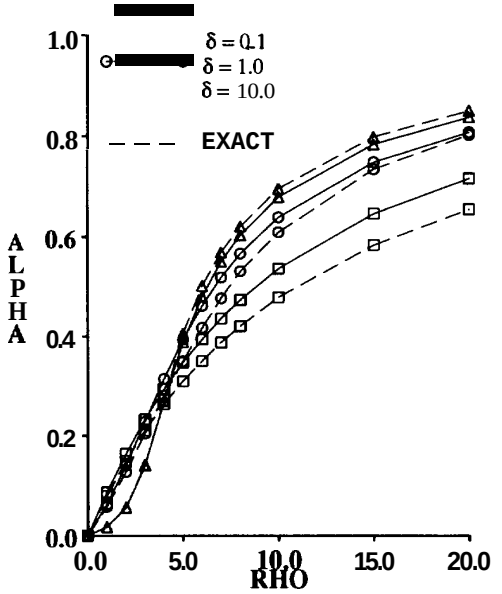

Figure 5.3: $\xi = 0.01$

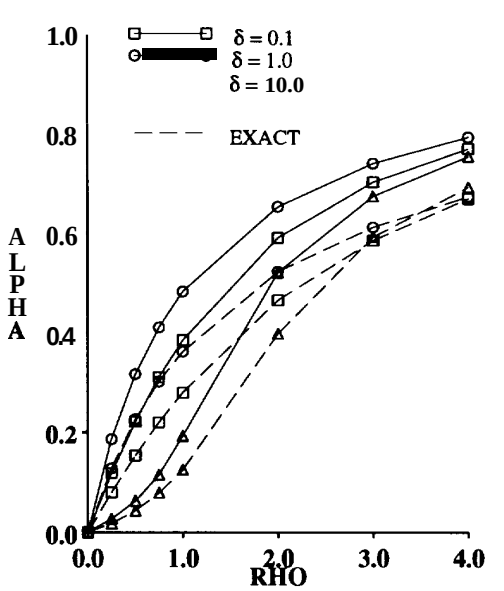

Figure 5.4: $\xi = 0.1$

6 Observations

In this section, we derive interesting corollaries from the **a** solution constructed in Section **4.**

6.1 Extreme Laxity Cases

We consider two extreme cases here, one where the laxity tends to 0, and the other where the laxity tends to ∞, keeping the remaining workload and system parameters fixed. When laxity tends to 0, transaction wait queues do not form and $\alpha_b \rightarrow 0$. Substituting $\delta = 0$ and $\alpha_b = 0$ in Equation 13 gives

$$\alpha_{\delta=0} = \alpha_h = 1 - \frac{1}{1+\rho\xi J} \qquad (14)$$

Conversely, when laxity tends to ∞, it is clear from Equation 2 that $\alpha_h \rightarrow 0$. Substituting $\delta \rightarrow \infty$ in the equation for α_b (Equation **9)** and simplifying, we obtain

$$\alpha_{\delta\rightarrow\infty} = \alpha_b = \begin{cases} \frac{}{\rho\sqrt{\xi J}} & \text{if } \rho > 1/\sqrt{\xi J} \\ & \text{otherwise} \end{cases} \qquad 15)$$

This equation shows that when transactions are willing to wait almost indefinitely to obtain service, they do not get killed unless the system offered load is greater than $1/\sqrt{\xi J}$. From Equation 8, this critical system load corresponds to the average number of locked database objects being $\sqrt{N}$.

6.2 Performance Crossover

An interesting feature of Figures 5.1 and 5.2 is that the large laxity (δ = 10.0) performance is worse than the small laxity (δ = 0.1) performance over virtually the entire loading range. Further, in Figure 5.3, a performance ***crossover*** (at $\rho = 4.0$) is clearly observed between the large laxity and the small laxity performances (the crossover occurs in Figures 5.1 and 5.2 also but is not clear due to the scale of the graph). This means that under light loads, large laxity results in improved performance, whereas under heavy loads, it is the other way around. Therefore, there is a critical loading point after which increased laxity can ***degrade*** performance. This may appear counter-intuitive since the expectation is that an increase in laxity should result in better performance, **as** observed in the corresponding classical real-time systems [11]. The reason for the difference in the database context is that transactions do not ask for generic servers, but for servers with "identity" (i.e. for specific data objects). As a result, transactions get queued up behind transactions that develop data conflicts and increased laxities result in longer queues and more conflicts. Under heavy loads, the queues become long enough that more and more transactions are killed while waiting in the queue although they have been provided with greater laxity. In short, the increased willingness to wait on the part of individual transactions is more than outweighed by the increased system queueing times that result from this willingness to wait.

6.3 Crossover Point

In this subsection, we compute the crossover loading point beyond which the **a** performance with $\delta \rightarrow \infty$ becomes worse than that with $\delta = 0$. By equating the α results obtained for $\delta = 0$ and $\delta \rightarrow \infty$, and after some algebraic manipulations, we obtain

$$\alpha_{crossover} = \begin{cases} \sqrt{\xi J} & \text{if } \sqrt{\xi J} < 1 \\ 1.0 & \text{otherwise} \end{cases} \qquad (16)$$

From this expression for $\alpha_{crossover}$, it is clear that with decreasing ξ (the database access ratio), the crossover occurs at lower and lower values of α. For example, with $\xi = 0.001$ and J = 10, the $\alpha_{crossover}$ evaluates to 0.1. This means that from the system perspective, for loading levels that result in a kill fraction greater than 0.1, a workload of transactions that are willing to wait almost indefinitely is more difficult to handle than a workload of transactions that find only immediate service acceptable.

6.4 Performance Bounds

By evaluating the partial derivative of **a** w.r.t. δ in Equation 13, the following corollary is obtained (the proof is provided in [4]):

Corollary 1 ***The α performance under light loads ($\rho \rightarrow 0$) is a decreasing function of δ. conversely, under heavy loads, ($\rho \rightarrow \infty$), the α performance is an increasing function*** of δ.

From this corollary, we infer that $\delta \rightarrow \infty$ provides the lower bound on **a** under light loads, and the upper bound on **a** under heavy loads. Conversely, $\delta = 0$ provides the upper bound on **a** under light loads, and the lower bound on **a** under heavy loads. Of course, since these bounds are derived from the approximations, they aren't exact numerical bounds on the queueing model itself; however, they serve to indicate the ***conditions*** under which these bounds would be achieved for the original system. In short, Equations **14** and 15 provide estimates of the **a** performance range in the light-load and heavy-load regions, respectively.

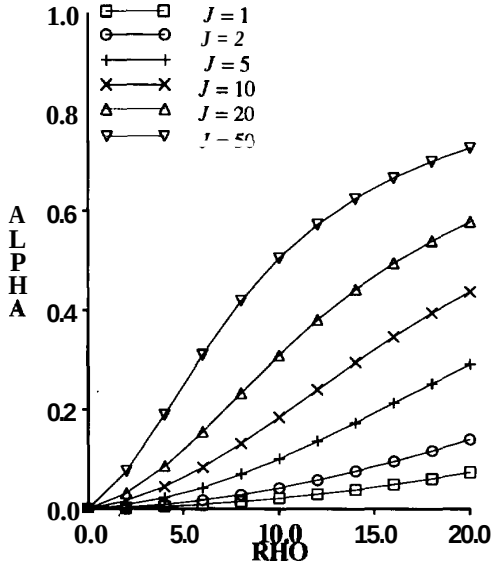

Figure 6.1: Effect of J (δ=1.0)

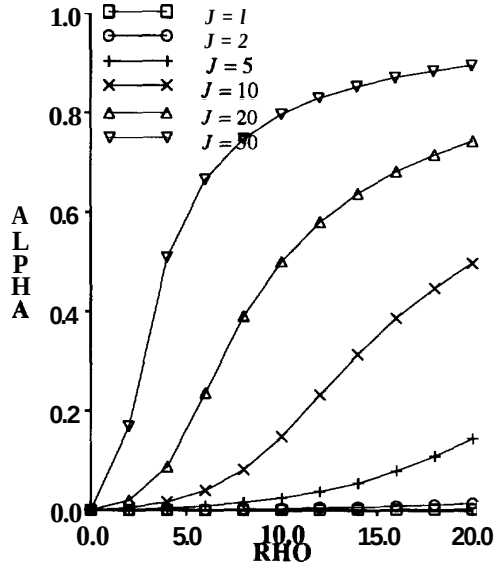

Figure 6.2: Effect of J (δ=10.0)

6.5 Effect of Transaction Size

We show in Figure **6.1** and Figure **6.2** the effect of varying the transaction size while keeping the database access ratio fixed (i.e., the database size is scaled in proportion to the transaction size). For this experiment, we set $\xi = 0.001$ and graph α as a function of ρ for different values of **J**, the transaction size. In Figure **6.1**, δ is set to **1.0** and in Figure 6.2, δ is set to 10.0. It is clear from these figures that the absolute value of the transaction size plays a significant role in determining system performance. This is in contrast to the classical $M/M/1$ and $M/M/1/M$ systems where performance is determined solely by *normalized* quantities [11].

7 Read and Write Access

In the derivations of Section **4,** it was assumed that transactions accessed all their data objects in write (exclusive lock) mode. The following lemma extends this analysis to include read (shared lock) data accesses (the proof is given in [4]).

Lemma 3 *Let each transaction request a fraction* ω $(0 \leq \omega \leq 1)$ *of its J data objects in write mode and the remainder in read mode. Then, Lemmas 1 and 2 and Theorem 1 apply on exactly the same form except that* ξ *is to be replaced by* $\xi\omega(2-w)$ *in all the equations.*

When $\omega = 1$ (all data items requested in write mode), the expression $\xi\omega(2-w)$ reduces to ξ, as should be expected. Conversely, when $w = 0$ (all data items requested in read mode), the expression $\xi\omega(2-\omega)$ reduces to 0. Substituting this value in the α solution results in $\alpha = 0$. This is as expected since no data conflicts occur when data is accessed only in shared mode, that is, the database behaves like an "infinite server".

8 Conclusions

In this paper, we have attempted a preliminary analytical study on the performance of real-time database systems with respect to the number of missed transaction deadlines. Our goal was to provide insight into RTDBS behavioral characteristics, rather than to quantify actual system performance. To this end, we modeled the real-time database as an an $M/M/N_J/M$ queueing system and developed an approximate closed-form solution for computing the fraction of killed transactions in this system. The solution is based on decoupling the queueing analysis from the database conflict analysis and then treating the transaction wait queue itself as an $M/G/1$ system.

The solution only requires finding the roots of a cubic equation, unlike typical Markovian models where the computational complexity is often a function of the parameter values. Due to its simplicity, the approximate solution provided us with insight into the sensitivity of system performance to workload parameters and also yielded conditions under which performance limits would be reached.

Our study showed that, for medium and large-sized databases, the approximate analysis provides extremely good qualitative agreement with the corresponding simulation-derived exact results. In addition, the quantitative results are also fairly accurate, especially under light loads. For small-sized databases, the qualitative matching was retained although there was considerable deterioration in quantitative accuracy under heavy loads.

Our experiments showed that the absolute value of transaction size, independent of its relation to database size, plays a significant role in determining system performance. Therefore, we recommend that designers of real-time database applications should try to minimize the size of their transactions. Our results also showed that unlike classical real-time systems, where increased task laxity usually results in improved performance, increased transaction laxity worsens performance under heavy loads. We provided a quantitative characterization of the loading level beyond which increased laxity results in degraded performance. We also showed that laxity tending to infinity provides the best performance under light loads, while laxity tending to zero is the best under heavy loads.

In our model, the transaction scheduler used a *fcfs* processing policy. Different performance behaviors may show up for prioritized scheduling disciplines. However, it is our view that the approximate analysis *methodology* described here can be successfully used to analyze these other cases also and we have had some preliminary encouraging results in this regard in our ongoing research. Some of the other assumptions in our model were that transaction laxities and processing times are exponentially distributed. We are currently working on extending the analysis to deterministic distributions of laxities and processing times.

References

[1] Abbott, R., and Garcia-Molina, H., "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Trans. on Database Systems,* 17(3), September 1992.

[2] Haugen, R., and Skogan, E., "Queueing Systems with Stochastic Time Out", *IEEE Trans. on Communications,* 28(12), December 1980.

[3] Haritsa, J., "Transaction Scheduling in Firm Real-Time Database Systems," *Ph.D. Thesis,* Computer Sciences Dept., Univ. of Wisconsin, Madison, August 1991.

[4] Haritsa, J., "Performance Analysis of Real-Time Database Systems," *Technical Report 92-96,* Systems Research Center, Univ. of Maryland, College Park, September 1992.

[5] Huang, J., "Real-Time Transaction Processing: Design, Implementation, and Performance Evaluation," *Ph.D. Thesis,* Computer and Information Science Dept., Univ. of Massachusetts, Amherst, May 1991.

[6] Kleinrock, L., "Queueing Systems," *Volume 1: Theory,* John Wiley & Sons, 1975.

[7] Lavenberg, S., "A Simple Analysis of Exclusive and Shared Lock Contention in a Database System," *Proc. of 4th ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems,* May 1984.

[8] Mitra, D., and Weinberger, P., '(Some Results on Database Locking: Solutions, Computational Algorithms and Asymptotics," *Mathematical Computer Performance and Reliability,* Iazeolla, G., Courtois, P., and Hordijk, A. (eds.), North-Holland., 1984.

[9] Tay, Y., Goodman, N., and Suri, R., "Locking Performance in Centralized Databases," *ACM Trans. on Database Systems,* Dec. 1985.

[10] Thomasian, A., and Ryu, I., "A recursive solution method to analyze the performance of static locking systems,'' *IEEE Trans. on Software Engineering,* Oct. 1989.

[11] Zhao, W., and Stankovic, A., "Performance Analysis of FCFS and Improved FCFS Scheduling Algorithms for Dynamic Real-Time Computer Systems," *Proc. of 10th IEEE Real-Time Systems Symposium,* December 1989.

Appendix

We present here the detailed proof for Lemma **2.** The first step is to compute $f_h(t)$, the service time distribution of transactions at the head-of-queue (pseudo)server. Both transactions that are killed before they reach the head of the queue and transactions that immediately enter the database on reaching the head of the queue have an effective service time of zero. Denoting the service time random variable by x_h, we have

$$f_h(0) = P(x_h = 0) = (\alpha_b + (1-\alpha_b) * (1 - P_{con}))\, u_0(t) \tag{17}$$

where P_{con} is the probability that a transaction at the head of the queue has to wait due to data conflict and $u_0(t)$ is the impulse function. **A** quick way to compute P_{con} is to realize that it is equivalent to the head-of-queue kill fraction in a system where tasks have zero laxity. Using the result in Equation **2,** we have

$$P_{con} = \alpha_{h|\delta=0} = \rho\xi J(1-\alpha)$$

Substituting this expression for P_{con} in Equation **17** and simplifying, we obtain

$$f_h(0) = (1-E)\, u_0(t) \tag{18}$$

where $E = \rho\xi J(1-\alpha_b)(1-a)$.

The transactions that do not fall into the above categories either gain entry into the database before their laxity expires or are killed while positioned at the head of the queue. The service time distribution for a transaction with remaining laxity l is given by

$$f_{h|l} = \begin{cases} \mu e^{-\mu t} & 0 < t < l \\ e^{-\mu l}\, u_0(t-l) & t \geq l \end{cases} \tag{19}$$

where the first equation corresponds to the case where the transaction's data conflict disappears before its laxity expires, and the second equation corresponds to the case where the transaction is killed.

Therefore, the unconditional pdf of the service time distribution when the service time is greater than 0 is

$$\begin{aligned} f_h(t>0) &= \int_t^\infty (\mu e^{-\mu t} + e^{-\mu l}\, u_0(t-l))\, \gamma e^{-\gamma l}\, dl \\ &= (\mu+\gamma)\, e^{-(\mu+\gamma)t} \end{aligned} \tag{20}$$

Combining the expressions in Equations **18** and **20,** the complete service time pdf is given by

$$f_h(t) = (1-E)u_0(t) + E(\mu+\gamma)e^{-(\mu+\gamma)t} \tag{21}$$

Then, using the well-known $M/G/1$ results [6], we obtain the corresponding waiting-time distribution to be

$$w(t) = (1-\rho_h)u_0(t) + (1-\rho_h)\lambda E e^{-(\mu+\gamma-\lambda E)t} \tag{22}$$

where ρ_h is the "utilization" of the head-of-queue server. Consequently, the CDF of the waiting time is given by

$$\begin{aligned} F_w(t) &= \int_0^t w(t)\, dt \\ &= (1-\rho_h)(1 + \frac{\lambda E}{\mu+\gamma-\lambda E}(1 - e^{-(\mu+\gamma-\lambda E)t})) \end{aligned}$$

The ρ_h parameter is easily computed as

$$\rho_h = \lambda \bar{x}_h = \lambda \frac{E}{\mu+\gamma}$$

from the distribution given in Equation **21.**

Substituting this value of ρ_h in the above equation for $F_w(t)$ and simplifying, we have

$$F_w(t) = 1 - Ge^{-(\mu+\gamma)(1-G)t} \tag{23}$$

where $G = \frac{\rho^2\delta\xi J}{1+\delta}(1-\alpha_b)(1-a)$

Recall that α_b is the probability of a transaction being killed due to its laxity being smaller than its wait time in the body of the queue. Therefore,

$$\alpha_b = \int_0^\infty (1 - F_w(t))\, \gamma e^{-\gamma t}\, dt$$

Substituting for $F_w(t)$ from Equation **23** and evaluating the integral, the above expression reduces to

$$\alpha_b = \frac{G}{2+\delta - G(1+\delta)} \tag{24}$$

Substituting for G in this equation, and then solving for α_b, we have

$$a;\, + P\alpha_b + 1 = 0 \tag{25}$$

where $P = 2 - (2+\delta)(\frac{1}{1+\delta} + \frac{1}{\rho^2\delta\xi J(1-\alpha)})$.

From the flow equation (Equation *1)* and from Equation **2,** we can express a in terms of α_b as

$$\alpha = 1 - \frac{(1-\alpha_b)}{1 + A(1-\alpha_b)} \tag{26}$$

Substituting this expression for a in Equation **25** and making algebraic manipulations, we finally obtain

$$\alpha_b^3 + B\alpha_b^2 + C\alpha_b + D = 0$$

where B, C, and D are the coefficients given in Equation **9.**

□