Group Commit Timers and High-Volume Transaction Systems

Pat Helland
Harald Sammer
Jim Lyon
Richard Carr
Phil Garrett
TANDEM COMPUTERS INCORPORATED

Andreas Reuter
UNIVERSITY OF STUTTGART

Technical Report 88.1
March 1988
Part Number 12522

Group Commit Timers and High-Volume Transaction Systems

Technical Report 88.1
March 1988
Part Number 12522

Pat Helland
Harald Sammer
Jim Lyon
Richard Carr
Phil Garrett
TANDEM COMPUTERS INCORPORATED

Andreas Reuter
UNIVERSITY OF STUTTGART

Contents

1. Introduction 1
2. Group Commit and Timers 1
3. Our Model 2
4. Motivation 3
5. Response Time With Zero Timers 4
5.1 Calculating C0 5
6. Response Time With Non-Zero Timers 8
6.1 Calculating Cd 8
7.1 Minimizing Response Time 10
7.2 Computing the Optimal Timer Delay (D) 11
8. Calculating A From System Statistics 12
9. The Effect of Group Commit Timers on Throughput 13
10. Testimonial 16
11. Future Research 16
11.1 Shepherds 17
11.2 A Minor Anomaly 18
12. Conclusions 19

Appendix
A. Hieroglyphics 20
B. References 28

When "Group Commit Timers and High-Volume Transaction Systems" was first published in the proceedings of the Second International Workshop on High-Performance Transaction Systems (Asilomar, California, September 1987), the graphs in Figures 4 and 5 were accidentally switched. The authors apologize for the error. In this publication, the graphs are correct.

GROUP COMMIT TIMERS AND HIGH VOLUME TRANSACTION SYSTEMS

PAT HELLAND, HARALD SAMMER, JIM LYON, RICHARD CARR, AND PHIL GARRETT
Tandem Computers, Inc.

ANDREAS REUTER
University of Stuttgart.

1) Introduction

In the summer of 1986, Tandem released the B30 version of its Transaction Monitoring Facility (TMF). This release included a GROUP COMMIT mechanism. It also included a primitive mechanism to gate the commit writes using manually adjusted TIMERS[1]. Subsequent experiments on large benchmarks have shown that timers reduce the overhead of transactions running on high volume systems. This reduced overhead increases system throughput while meeting response time constraints.

While other systems have used Group Commit Timers[2], we have investigated the fact that various timer values are appropriate for various system loads. This paper discusses TIMERs and describes how a transaction's commit overhead is reduced by making it delay. It turns out that the system can set the timer dynamically to minimize average response time. Calculations for the optimal TIMER DELAY are presented. Finally, some directions for future research are described.

2) Group Commit and Timers

Group Commit refers to the technique used in high volume transaction systems where many transactions are committed with a single disk I/O to the log. That single I/O may contain commit records for several transactions.

Group Commit has been in use for a number of years now [Gawlick] . Without group commit, the transaction rate of a single-log system would be limited by the number of transfers per second that the log could support. Group Commit allows transaction systems with disk-based logs to break through that barrier of roughly 30 transaction per second.

When Group Commit Timers are in use, the system behaves as follows:

-- When a transaction needs to write a commit record it will wait.

-- When the Group Commit Timer pops, all of the transactions waiting to get a commit record written will have a record written as a part of a single disk I/O to the log disk.

We also define special behavior when the Group Commit Timer value is 0. This is:

-- If a transaction needs to write a commit record and there is no log I/O outstanding, then the commit record is immediately written to the log.

-- If a log I/O completes and other transactions are waiting to commit, then commit records for all waiting transactions are written using a single I/O.

When non-zero Group Commit Timers are in effect, the system awakens periodically and writes commit records for all waiting transactions using a single I/O. The result is that commit writes occur at scheduled intervals. Non-zero Group Commit Timers are a lot like people waiting on the street corner for a bus that arrives every five minutes. When the Group Commit Timer is zero, the bus driver will leave when the first passenger arrives.

3) Our Model

We have chosen to examine the effect of Group Commit Timers on CPU response time. One could justifiably argue that this is inadequate because disk I/O and the effects of queuing for disk I/O have not been included. This has been done for the following reasons:

- Since we have a single writer of the Log and the Log is kept on a separate disk from the database, queuing for I/O on the Log disk is negligible.
- We believe that Group Commit Timers will not significantly affect the queuing for disk at a fixed transaction rate.
- The response times that we use are for comparison purposes only. We intend to minimize the time that a transaction spends queued for the CPU and utilizing the CPU.
- Most importantly, we were getting a headache understanding this much.

Fundamentally, we are assuming that Group Commit Timers do not affect the I/O component of the response time. We hope to see further extensions of this work which incorporate disks and disk queuing into the model.

4) Motivation

Timers cause individual transactions to be delayed. Intuitively, this also implies an increase in the transaction's response time. Surprisingly, timers can reduce the average transaction's CPU response time by reducing the per-transaction overhead.

To understand this, one must realize that transactions spend much of their time waiting in the CPU queue. By reducing the total demand for CPU resources, we reduce the length of the queue. The time that a transaction waits in the queue can be dramatically reduced if the CPU is heavily loaded.

Group Commit Timers cause the number of transactions in each Group Commit Buffer to increase. The number of Group Commit writes drops. This decreases both the average CPU cost for each transaction and the total CPU work performed by the system. As the total system CPU work drops, so does the queue length for the CPU and the average response time for a transaction. If the savings in CPU queuing exceeds the average wait time for the Group Commit Timer, then Timers improve the transaction's response time.

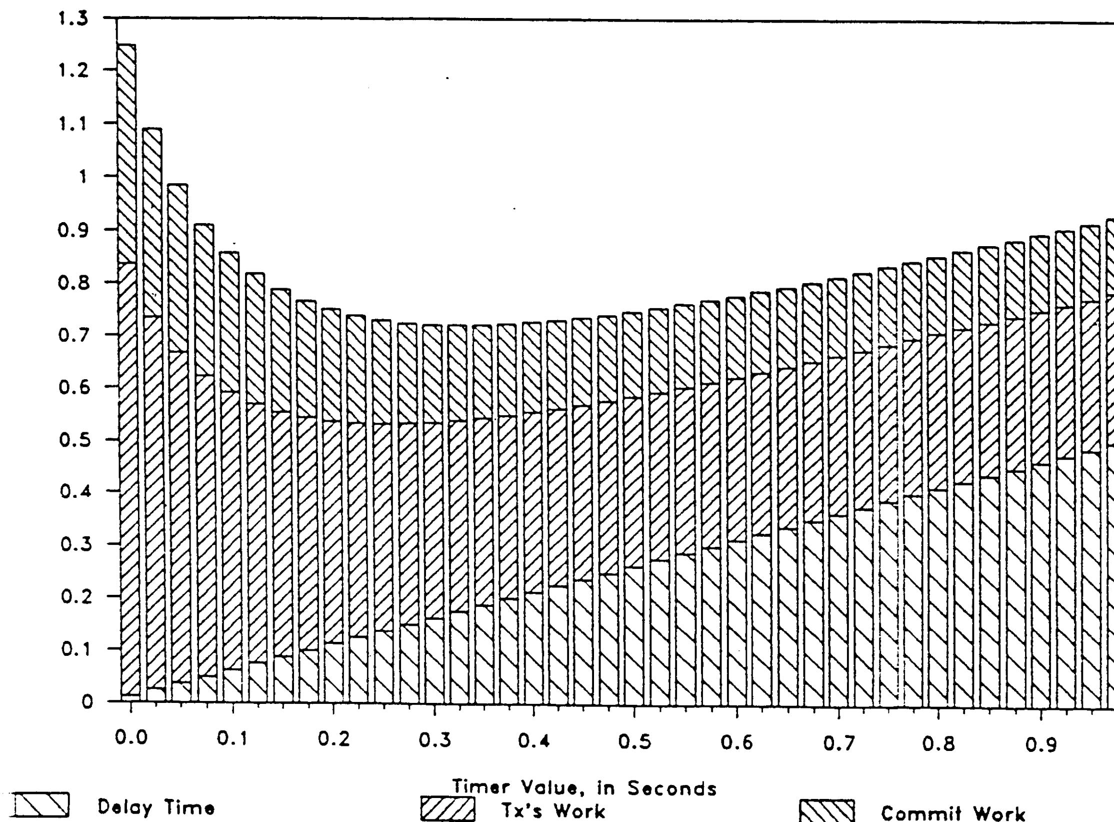


Fig. 1 - **Response Time vs. Timer Value**

Note that as the timer values increase, the time spent for the transactions work (including CPU queuing) drops.

The remainder of this paper is organized as follows:

Section 5 describes the technique for calculating the average response time of a transaction when the Group Commit Timer is zero.

Section 6 describes how transaction response times are calculated in a system in which timer values are non-zero.

Section 7 discusses a technique by which timer values may be automatically determined as a function of measurable system parameters.

Section 8 describes how to calculate all of the necessary parameters for the equations from the system. This allows the system to plug in values to the described equations and set the timers dynamically.

Section 9 presents some predictions of the effect of Group Commit Timers on various transaction loads..

Section 10 consists of a brief description of the use of the Group Commit Timers at Tandem.

Section 11 discusses areas in which we feel further research may be beneficial.

Finally, Appendix A contains all of the mathematics necessary to derive the formulas presented. If you trust our math, the appendix may be skipped.

5) Response Time With Zero Timers

Before examining the expected response times for transactions when timers are non-zero, it is important to analyze the behavior of transactions when the Group Commit Timers are zero. In this discussion, as in the rest of the paper, we examine the total time spent by the transaction either executing in the CPU or waiting in the CPU queue. The time for the physical I/Os to the database has been omitted. We are assuming that Group Commit Timers do not affect the I/O time.

Let's divide the transaction into two parts to examine the CPU costs associated with it. We choose to consider the transaction's CPU cost as the sum of:

A - *The CPU cost of a transaction (excluding COMMIT).*

B - *The CPU cost for one transaction's COMMIT (i.e. the cost to write a Group Commit Buffer).*

Both A and B are measured in seconds.

Define:

C_0 - *The average number of transactions in a Group Commit Buffer when a timer value of zero is used.*

T - *The transaction rate (transactions per second).*

U_0 - *The CPU utilization of a system when no timers are used.*

R_0 - *The average transaction's CPU response time on a system when the timer has a value of zero.*

We can now calculate CPU utilization by determining the amount of work for each transaction times the number of transactions per second. The utilization is dimensionless.

$$U_0 = (A + B / C_0) * T$$

Next, assuming M/M/1 queuing gives us a response time for a transaction of:

$$\text{CPUResponseTime} = \frac{(\text{CPU Work})}{(1 - \text{utilization})} \quad \text{seconds.}$$

This means that the transaction's average CPU response time (that is, its response time exclusive of I/O) will be:

$$R_0 = \frac{(A + B)}{(1 - U_0)} \quad \text{seconds.}$$

This can be rewritten as:

$$R_0 = \frac{(A + B)}{(1 - T(A + B / C_0))} \quad \text{seconds.}$$

5.1) Calculating C_0

Unless B is small with respect to A, the calculation for the average transactions response time will be highly dependent on C_0 (the fullness of the Group Commit Buffer). To understand the calculation of C_0, we must review what can cause a Group Commit Buffer to be written.

There are two reasons why a Group Commit buffer would be written when the timer is zero:

-- If a transaction is ready to commit and no I/O is in progress.

-- If an I/O completes and a transaction is waiting to commit.

Let's define the following terms:

L - *The amount of time it takes a Log I/O to complete. This time is wall clock time rather than CPU time. We are not including queuing for the disk.*

W_t - *The total writes to the log per second.*

W_i - *The immediate writes to the log per second. These are the writes caused by the transaction arriving and finding no I/O to the log in progress.*

W_d - *The delayed writes to the log per second. These are the writes caused by an I/O completing with one or more transactions waiting.*

C_{0d} - *The average number of transactions in a delayed write buffer.*

Note that:

$$W_t = W_i + W_d$$

Now, we remember that each immediate write will have one transaction in each buffer. Each delayed write will have C_{0d} transactions in each buffer. So, now we can state that:

$$C_0 = \frac{W_i + W_d C_{0d}}{W_t}$$

After much gnashing of teeth, this becomes:

$$C_0 = LT + e^{-LT}$$ (see Appendix A.1)

As you can see, when the timer is zero, the number of transactions in a Group Commit buffer is a function of the transaction rate and the time a write to the log takes. It is completely independent of any other factors. For this reason, it is possible to pick a reasonable time for the log write and chart C_0 as a function of transaction rate. See figure 2.

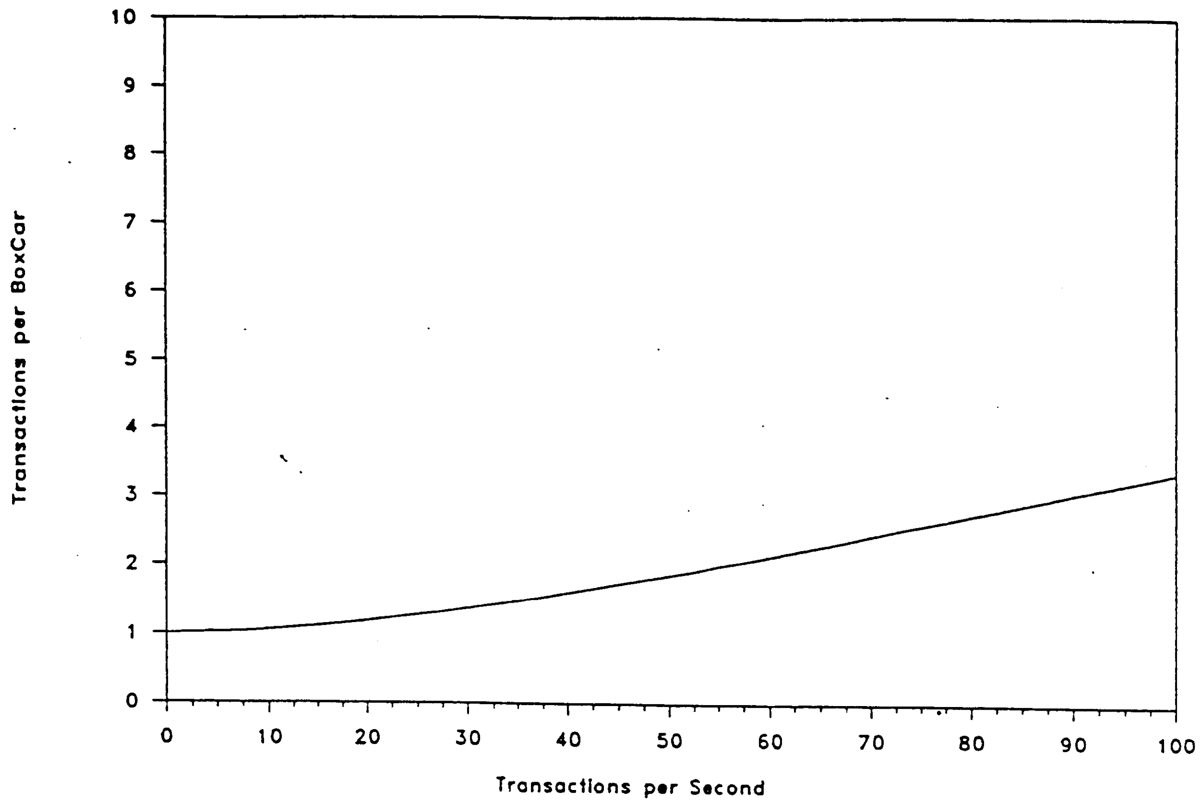


Fig. 2 - Zero-Timer Group Commit Buffer Size with L = .033 seconds

6) Response Time With Non-Zero Timers

The calculation of the system utilization and transaction response time follows the same pattern for systems using non-zero timers as it does for systems with zero timers. There are two differences: we expect the utilization to be different and the response time with non-zero timers must include an expected wait for the timer to pop. Let's define:

D - *The timer delay (in seconds). Every D seconds, the timer will pop and a Group Commit Buffer write will be attempted.*

C_d - *The average number of transactions in a Group Commit Buffer when timers are non-zero.*

U_d - *The CPU utilization when timers are non-zero.*

R_d - *The average transaction's CPU response time when timers are non-zero.*

We can now state that:

$$U_d = (A + B / C_d) * T$$

$$R_d = \frac{(A + B)}{(1 - U_d)} + D/2 \qquad \text{seconds.}$$

In a system using non-zero Group Commit Timers, there is only one reason for writing a Group Commit Buffer: the timer pops, and at least one transaction needs a commit record written.

We must determine C_d to be able to evaluate this.

6.1) Calculating C_d

To evaluate C_d, we must first discuss the conditions under which a Group Commit buffer will be written. When the timer pops (every D seconds), the Group Commit buffer will be written to the log if one or more transactions are waiting to write a commit record. All of the waiting transactions will be included in the buffer. We are assuming that a full buffer is so unusual a condition as to be negligible.

What we must be careful to consider, however, is the probability that the timer will pop and there will be no transactions waiting to be committed. The Group Commit buffer will not be written under those circumstances.

To evaluate this probability, we use a Poisson distribution. We represent the probability that exactly *n* transactions are waiting for a commit record to be written as:

$$P_{TD}(n)$$

where TD is the transaction rate (T) times the delay interval (D). This represents the expected mean number of transactions.

We can now describe the expected number of records written in a Group Commit buffer as:

$$\sum_{i=1}^{\infty} iP_{TD}(i)$$

The probability that a Group Commit write will occur is the same as the probability that at least one transaction is ready to commit when the timer pops. This is:

$$\sum_{i=1}^{\infty} P_{TD}(i)$$

So, the average number of transactions in a Group Commit buffer will be:

$$C_d = \frac{\text{Expected records per write}}{\text{Probability that a write occurs}}$$

Which is:

$$C_d = \frac{\sum_{i=1}^{\infty} iP_{TD}(i)}{\sum_{i=1}^{\infty} P_{TD}(i)}$$

Which boils down to become:

$$C_d = \frac{TD}{1 - e^{-TD}}$$

(see Appendix A.2)

Now that we know how to calculate the average response time, what do we do with it? Our goal is to employ the timer mechanism in such a way that it minimizes the average transaction's response time. If we can derive a formula based on information available in the system, we can construct an algorithm to set the timers dynamically, based upon the system load.

7.1) Minimizing Response Time

The average response time for a transaction (excluding I/O) when timers are in use is:

$$R_d = \frac{(A + B)}{(1 - U_d)} + D/2 \text{ seconds.}$$

Our goal is to rearrange this formula to be a function of:

T - *The transaction rate of the system. This we can directly measure as the system operates.*

B - *The cost of writing a Group Commit Buffer. This is a constant cost on a given system. Its value is determined by using performance measuring tools.*

A - *The average transaction's CPU cost (excluding the cost of commit). This we can calculate as a function of B, T, measured CPU utilization, and measured rate of writing the Group Commit Buffer.*

D - *The Group Commit Timer Delay.*

Once we have this formula, we then assume that B, A, and T are constants with respect to R_d. The calculation of R_d can then be viewed as a function in terms of D. The minimum value of R_d can then be found by taking its derivative and solving for:

$$R_d' = 0$$

This strategy gives us:

$$R_d = \frac{DA + DB}{D - TAD - B + Be^{-TD}} + D/2$$

(see Appendix A.3)

$$R_d' = \frac{B(A + B)(TDe^{-TD} - 1 + e^{-TD})}{(D - TAD - B + Be^{-TD})^2} + 1/2$$

(see Appendix A.4)

If we assume that $R_d' = 0$, we can then derive that:

$$D = \frac{B(1 - e^{-TD}) + \sqrt{(2B(A + B)(1 - e^{-TD} - TDe^{-TD}))}}{1 - TA}$$

(see Appendix A.5)

This is the formula that defines the proper value for a Group Commit Timer.

7.2) Computing the Optimal Timer Delay (D)

As we have just seen, D may be expressed as:

$$D = \frac{B(1 - e^{-TD}) + \sqrt{(2B(A + B)(1 - e^{-TD} - TDe^{-TD}))}}{1 - TA}$$

Unfortunately, this solution for D involves D in the right half side of the equation. We have chosen to determine a method to guess at an initial value for D and then refine the guess to be within acceptable limits.

When we solved for D, we did so assuming that $R_d' = 0$. We then rearranged the function to arrive at a value for D. In doing so, we really arrived at a different function Y which we know has the same value for D at the point Y = 0. It is important to realize that the function Y is not really the same as R_d'. We have chosen as Y the function:

$$Y = 2B(A+B)(TDe^{-TD} - 1 + e^{-TD}) + (D - TAD - B + Be^{-TD})^2$$

(see Appendix A.6)

Once we have an initial guess for the value of D, we will use the function Y to find a better guess. The Newton-Raphson method[PRESS] extrapolates to find the next estimate of the root that we are searching for. This is done by geometrically extending the tangent line at the current point $Y(D_i)$ until it crosses zero. This then becomes our next guess $Y(D_{i+1})$.

So, to iterate, we have the equation:

$$D_{i+1} = D_i - \frac{Y(D_i)}{Y'(D_i)}$$

To do this we need Y'. We have concluded that this is:

$$Y' = 2((1 - TA)D - B(1 - e^{-TD}))(1 - TA - TBe^{-TD}) - 2B(A + B)T^2De^{-TD}$$

(see Appendix A.6)

For our initial guess, we will assume that TD is large. This causes e^{-TD} to approach zero. We will take as our guess the value of D when e^{-TD} is zero. This gives us:

$$D_0 = \frac{B + \sqrt{2B(A + B)}}{1 - TA}$$

So, to calculate the optimal timer values, we will first calculate D_0 using the above formula. We then calculate successive D_i values for a few iterations. In most cases, this converges very rapidly.

8) Calculating A From System Statistics

The preceding section derives a formula for the timer delay value which is a function of B, T, and A. B is a constant (for a given release of the operating system). T is measurable from the system. It remains to describe how A can be calculated dynamically.

Define the following terms:

R - *The frequency with which the Group Commit Buffer is written. The system can trivially maintain statistics to provide this.*

U_{sys} - *The total system work. This is represented by the CPU utilization. Again, the system can trivially maintain statistics to provide this.*

As discussed above, all of the work for transactions running on the system may be considered to be divisible into two parts: A and B. B is the cost of writing a commit buffer. A is the remaining cost of the transaction.

This means that we can consider the work performed by the system to have two components:

$$U_{sys} = (R * B) + \text{Other transaction work}$$

The other transaction work may be averaged among the transactions processed during the sample interval. This implies that:

$$\text{Other transaction work} = (A * T)$$

This leaves us with:

$$U_{sys} = (R * B) + (A * T)$$

Where R, U_{SYS}, and T are measurable and B is a constant. Rearranging things leaves us with:

$$A = \frac{U_{sys} - (R * B)}{T}$$

9) The Effect of Group Commit Timers on Throughput

To learn about the effects of Group Commit timers, we set up some spread sheets to plot throughput verses response time for a number of artificial transactions. In the first, we assumed that:

A = 100 Milliseconds

B = 50 Milliseconds

L = 33 Milliseconds

This nicely models a machine in which a relatively low transaction rate is achieved on a single processor. This gave us the following results:

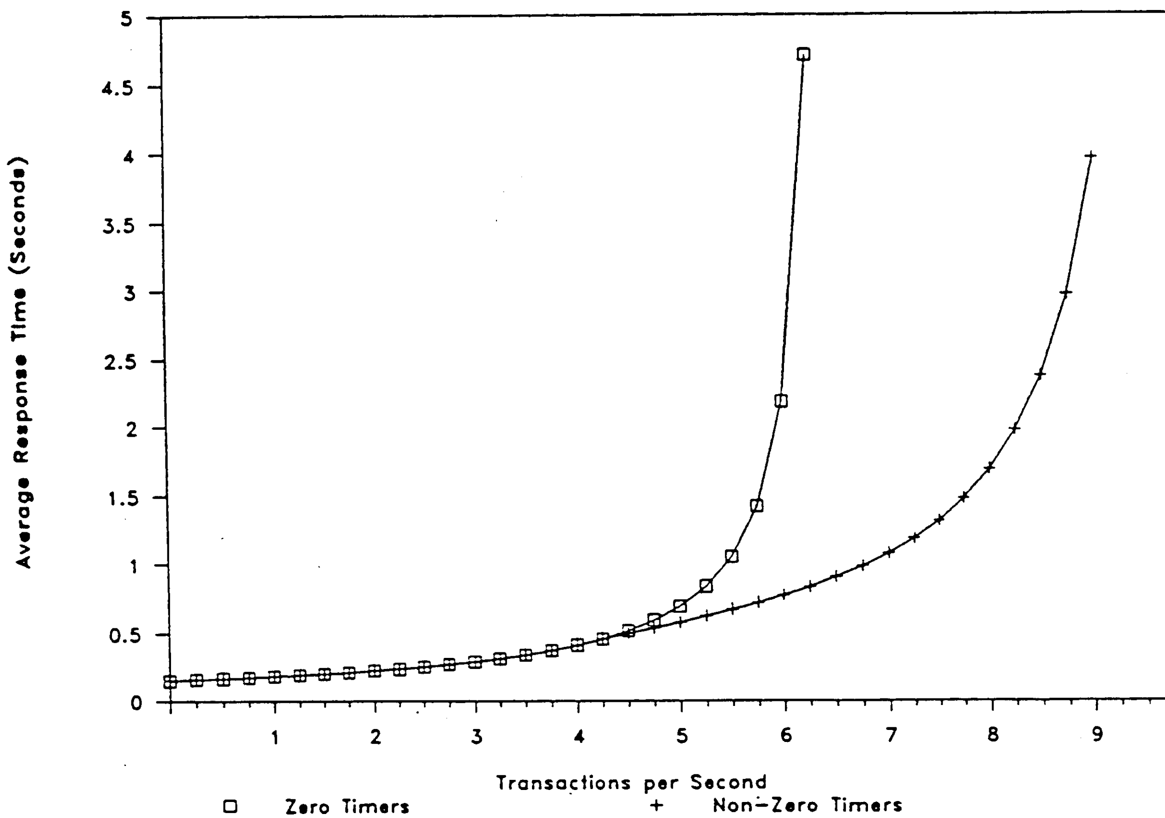


Fig. 3 - System Throughput vs. Response Time -- Trial 1

Next, we tried to analyze what would happen if the clock rate of the machine was 10 times as great. This would give us:

A = 10 Milliseconds

B = 5 Milliseconds

L = 33 Milliseconds (remember, L is wall clock time)

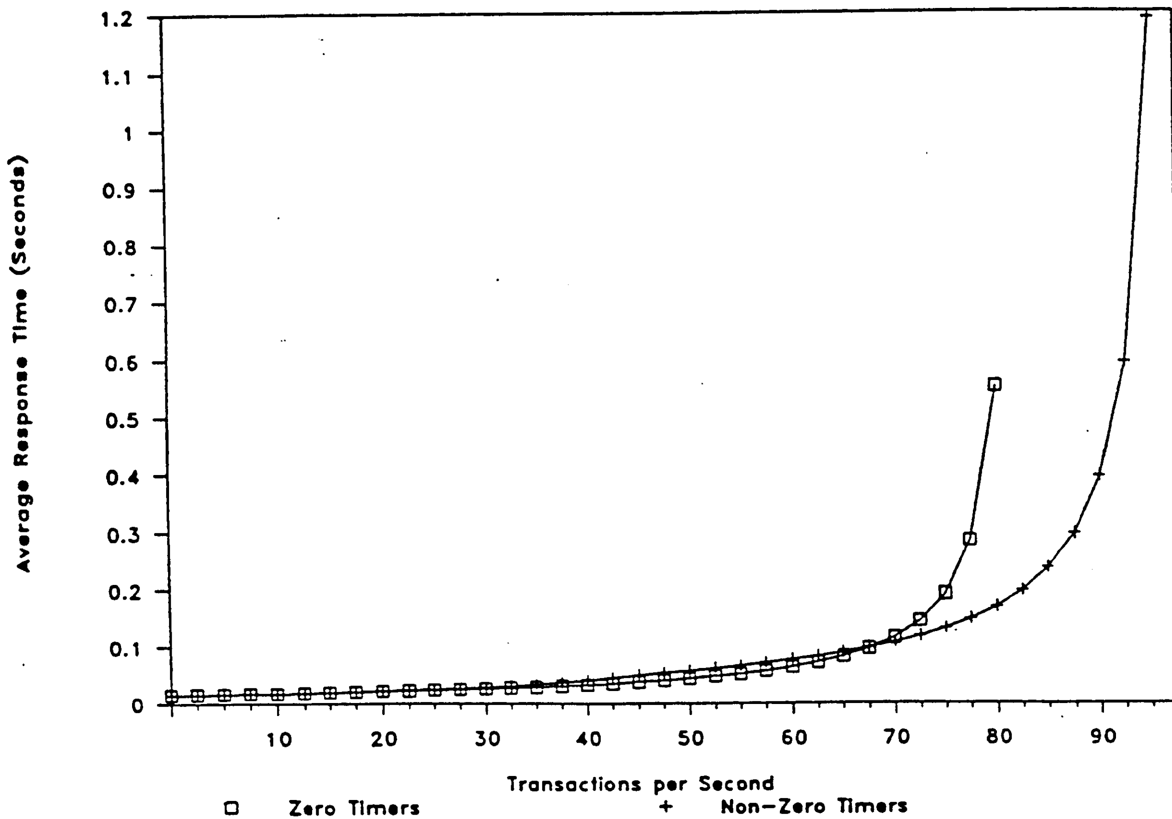


Fig. 4 - System Throughput vs. Response Time -- Trial 2

Notice that the dramatic results are a consequence of the ratio between A and B (which is what we experience on our machine). If you change this ratio because you have extra good logging or you have larger transactions, the effect is present, but not as pronounced. Let's try:

A = 100 Milliseconds

B = 10 Milliseconds

L = 33 Milliseconds (remember, L is wall clock time)

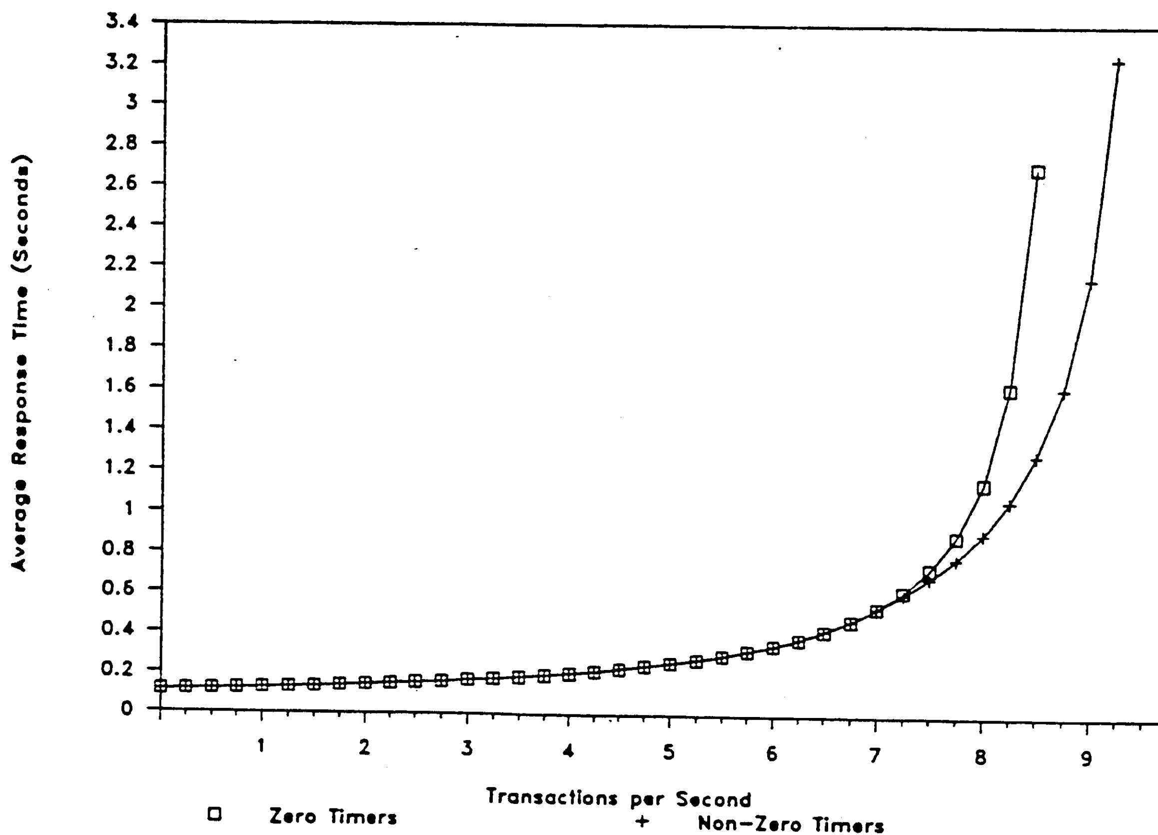


Fig. 5 - System Throughput vs. Response Time -- Trial 3

10) Testimonial

In the Spring of 1987, Tandem gathered 32 of VLX processors together to perform a large benchmark as a prelude to the announcement of our NonStop SQL product. When we started the benchmark (which consisted of DebitCredit[3] transactions with 2 second response time), we were not using the Group Commit Timers. Running with timer values of zero, we were able to process 165 transactions per second.

After performing some "back-of-the-envelope" calculations, the Group Commit Timer was set. By using the Group Commit Timer (and the Audit Flush Timer discussed below in section 11), we were able to process 208 transactions per second with 2 second.

11) Future Research

A number of other applications for dynamic timers have been discussed amongst our cohorts. The idea of Group Commit Timers is an approach that is not specifically related to Group Commit. The idea works whenever you have multiple activities which could be accomplished more efficiently if bundled together into one unit of work.

Some ideas that have come to mind include:

- <u>TIMERS AND MESSAGE BASED SYSTEMS</u>

  The Tandem system is a distributed multi-processor system. The various CPUs communicate with each other by sending messages. Currently, somewhere in excess of 20% of our machine cycles are spent sending messages.

  The overhead for sending a message has two components: a "per-message" overhead and a "per-byte" overhead. If we could bundle multiple messages into a single transmission, some of the "per-message" overhead cost could be amortized over the messages that were bundled.

- <u>TIMERS FOR DATA COMMUNICATION</u>

  Most of the data communications protocols have large overheads associated with each transmission. By bundling multiple messages going in the same direction, into a single transmission, many CPU cycles could be saved.

  As with Group Commit, the savings associated with increased sharing of the same work could reduce the response time of the work enough to justify hanging around for a while to see if anyone else wants to hitch a ride.

- AUDIT FLUSH TIMERS

  The Tandem system is a distributed system in which the before and after audit images for database updates are frequently generated and buffered in a CPU that is not physically connected to the disk holding the log[4,5]. To commit a transaction, this audit must be transmitted to one of the CPUs that is physically connected to the Log disk.

  We have found that significant savings may be obtained by using timers to delay the transmission of the audit images for a while. When timers are used here, the buffer that is sent will contain the audit images for more transactions that are ready to commit.

- LOCK RELEASE TIMERS

  This was an idea that didn't work. In the Tandem system, the record locks on the database are maintained in the process that manages the disk containing the records being locked. To release the locks, the system must dispatch the disk process. We tried to see if by delaying the dispatch of the disk process, we could cause the disk process to release the locks for multiple transactions at the same time.

  Well, it would release the locks for multiple transactions at the same time, but the CPU savings weren't worth the increased lock contention. Oh, well.

11.1) Shepherds

As discussed above, the Tandem system is already employing two timers that affect the life of a transaction. We are considering the possibility of adding more.

When one function controlled by a timer may be begun as soon as another function controlled by a timer completes, a token (or ***Shepherd***) may be passed as each function is processed. Even if the second function (e.g. the Group Commit buffer flush) must wait for a bunch of completions of the first function (e.g. Audit Flush from all of the constituent processors), the shepherd may be used.

The basic advantage to this approach is that you can get savings by sharing the work of all of the various functions while only delaying for the processing of the first function. Once the first function starts, you, your other transactions, and the shepherd will get immediate service from the remaining functions.

Our guesstimates indicate that shepherds come out dead even with multiple timers on our system in which two levels of timers are employed. Once we start bundling more of the system functions into activities that can be simultaneously performed for multiple transactions, shepherds will be looked at even more closely.

11.2) A Minor Anomaly

When we went to model the number of transactions in a Group Commit buffer when non-zero timers are used, we found that the number was worse than the zero timer cases in a very small range (this effect is visible in Figure 6). The problem occurs at low transaction rates when timers are first enabled and the timer value is still smaller than L (the wall clock time to do a log I/O). At this point, our Response Time model ceases to be valid.

We decide not to worry about this at this time and chose, instead, to restrict the system to behave as if the timer value was zero when the calculated optimal timer value was smaller than L.

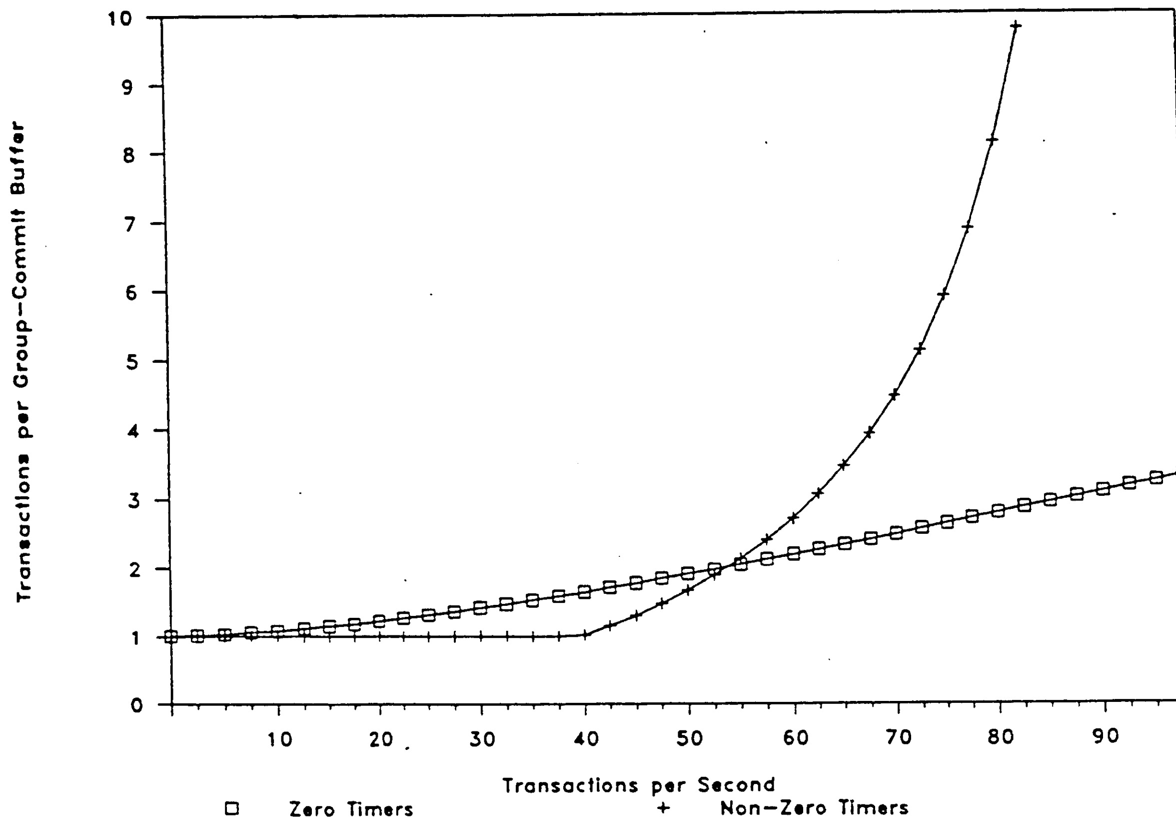


Fig. 6 - Non-Zero Timer vs Zero Timer Buffer Sizes

This is assuming the following values:
L - .033 seconds
A - .010 seconds
B - .005 seconds

12) Conclusions

Group Commit timers have been shown to provide very noticeable increases in throughput while still meeting response time objectives. They offer a surprising gain for a modest amount of implementation.

We feel quite strongly that timers are important. The arbitrary selection of a specific value for the timer could harm some transaction mixes. An example of this would be a system running serialized batch transactions. The use of a non-zero timer will only add to the response time. To expect a system manager to wisely select an appropriate timer value is also untenable. For these reasons, it seems clear that the system should dynamically calculate an appropriate timer value based on the system load.

While more work is needed to perfect the model, we feel comfortable using these formula in our production software to calculate the Group Commit Timer value.

A) Appendix A: Hieroglyphics

This appendix fills in all of the "leaps of faith" that have been made in the rest of the paper.

A.1) Calculate C_0

As described in section 5.1, the formula for the expected Group Commit Buffer population (C_0) when no timers are used is:

$$C_0 = \frac{W_i + W_d C_{0d}}{W_t}$$

A.1.1) Calculate C_{0d}

The calculation for C_{0d} is almost identical to the calculation for C_d. The only difference is that the time to accumulate transactions is not D, but L (the service time on a log I/O). Following the same arguments as those presented for C_d, we have:

$$C_{0d} = \frac{\text{Expected records per delayed write}}{\text{Probability that a delayed write occurs}}$$

Which is:

$$C_{0d} = \frac{\sum_{i=1}^{\infty} iP_{LT}(i)}{\sum_{i=1}^{\infty} P_{LT}(i)}$$

Note that:

$$\sum_{i=1}^{\infty} iP_{LT}(i) = \sum_{i=0}^{\infty} iP_{LT}(i)$$

This is because when i = 0, $0P_i(LT)$ equals 0. But note that:

$$\sum_{i=0}^{\infty} iP_{LT}(i) \quad = LT$$

Hence:

$$\sum_{i=1}^{\infty} iP_{LT}(i) \quad = LT$$

Now we observe that:

$$\sum_{i=1}^{\infty} P_{LT}(i) \quad = 1 - P_{LT}(0)$$

This means that:

$$C_{0d} = \frac{LT}{1 - P_{LT}(0)}$$

A.1.2) Calculate W_d

The number of delayed writes to the disk is somewhat complicated to compute. A delayed write occurs whenever a log write finishes and 1 or more transactions is waiting to commit. These transactions must have arrived while the log I/O was outstanding (or in the time interval L). The probability that one or more transactions is waiting can then be expressed as:

$$(1 - P_{LT}(0))$$

This means that the number of delayed writes is:

$$\begin{aligned} W_d &= W_t\,(1 - P_{LT}(0)) \\ &= (W_i + W_d)\,(1 - P_{LT}(0)) \\ &= W_i(1 - P_{LT}(0)) + W_d(1 - P_{LT}(0)) \\ &= W_i(1 - P_{LT}(0)) + W_d - W_dP_{LT}(0) \end{aligned}$$

So:

$$W_dP_{LT}(0) \quad = W_i(1 - P_{LT}(0))$$

$$W_d \quad = \frac{W_i(1 - P_{LT}(0))}{P_{LT}(0)}$$

A.1.3) Calculate W_t

Let's start out with:

$$W_t = W_i + W_d$$

$$= W_i + \frac{W_i(1 - P_{LT}(0))}{P_{LT}(0)}$$

$$= \frac{W_i P_{LT}(0)}{P_{LT}(0)} + \frac{W_i(1 - P_{LT}(0))}{P_{LT}(0)}$$

Giving us:

$$W_t = W_i \frac{1}{P_{LT}(0)}$$

Which means that:

$$W_i = W_t P_{LT}(0)$$

A.1.4) Put C_0 Together

Now that we know both C_{0d} and W_d, we can calculate that:

$$W_d C_{0d} = W_t(1 - P_{LT}(0)) \frac{LT}{(1 - P_{LT}(0))}$$

$$= W_t LT$$

Next, we recall that

$$C_0 = \frac{W_i + W_d C_{0d}}{W_t}$$

Substituting gives us:

$$C_0 = \frac{W_t P_{LT}(0) + W_t LT}{W_t}$$

$$= P_{LT}(0) + LT$$

Now, we have to look in our textbook[PRESS] to tell us the value of the Poisson Probability Function. According to the magic formula in the book,

$$P_x(0) = e^{-x}$$

Hence,

$$P_{LT}(0) = e^{-LT}$$

And we get:

$$C_0 = LT + e^{-LT}$$

A.2) Calculate C_d

The calculation for C_d is almost identical to the calculation for C_{0d}. The only difference is that the time to accumulate transactions is D when calculating C_d and L (the service time on a log I/O) when calculating C_{0d}. For this reason, the discussion here and in section 6.1 is almost identical to the discussion in sections 5.1 and A.1.1.

In section 6.1, we determined that:

$$C_t = \frac{\sum_{i=1}^{\infty} iP_{TD}(i)}{\sum_{i=1}^{\infty} P_{TD}(i)}$$

Note that:

$$\sum_{i=1}^{\infty} iP_{TD}(i) = \sum_{i=0}^{\infty} iP_{TD}(i)$$

This is because when $i = 0$, $0P_i(TD)$ equals 0. But note that

$$\sum_{i=0}^{\infty} iP_{TD}(i) = TD$$

Hence:

$$\sum_{i=1}^{\infty} iP_{TD}(i) = TD$$

Now we observe that:

$$\sum_{i=1}^{\infty} P_{TD}(i) = 1 - P_{TD}(0)$$

This means that:

$$C_d = \frac{TD}{1 - P_{TD}(0)}$$

$$C_d = \frac{TD}{1 - e^{-TD}}$$

A.3) Calculate R_d

As described in section 6, R_d is:

$$R_d = \frac{(A + B)}{(1 - U_d)} + D/2$$

Substituting for U_d (again described in section 6) we have:

$$R_d = \frac{(A + B)}{1 - T(A+B/C_d)} + D/2$$

$$= \frac{(A + B)}{1 - TA - TB/C_d} + D/2$$

$$= \frac{C_dA + C_dB}{C_d - TAC_d - TB} + D/2$$

Substituting for C_d (described in section 6.1) yields:

$$R_d = \frac{ATD/(1 - e^{-TD}) + BTD/(1 - e^{-TD})}{TD/(1 - e^{-TD}) - T^2AD/(1 - e^{-TD}) - TB} + D/2$$

Multiplying above and below by $(1 - e^{-TD})$ yields:

$$R_d = \frac{ATD + BTD}{TD - T^2AD - TB + TBe^{-TD}} + D/2$$

$$= \frac{AD + BD}{D - TAD - B + Be^{-TD}} + D/2$$

A.4) **Calculate R_d'**

Let's make the following assignments:

$$f = AD + BD$$

$$g = D - TAD - B + Be^{-TD}$$

Then we have:

$$R_d' = \frac{f'g - fg'}{g^2} + 1/2$$

$$f' = A + B$$

$$g' = 1 - TA - TBe^{-TD}$$

$$f'g = (A + B)(D - TAD - B + Be^{-TD})$$

$$= AD - TA^2D - AB + ABe^{-TD} + BD - TABD - B^2 + B^2e^{-TD}$$

$$fg' = (DA + DB)(1 - TA - BTe^{-TD})$$

$$= AD - TA^2D - ABTDe^{-TD} + BD - TABD - B^2TDe^{-TD}$$

So, when we combine f'g and fg' we get:

$$f'g - fg' = AD - TA^2D - AB + ABe^{-TD} + BD - TABD - B^2 + B^2e^{-TD}$$

$$- AD + TA^2D + ABTDe^{-TD} - BD + TABD + B^2TDe^{-TD}$$

$$= - AB - B^2 + ABe^{-TD} + B^2e^{-TD} + ABTDe^{-TD} + B^2TDe^{-TD}$$

$$= (A + B)(-B + Be^{-TD} + TBDe^{-TD})$$

$$= B(A + B)(TDe^{-TD} - 1 + e^{-TD})$$

Now, we can assemble R_d' as:

$$R_d' = \frac{f'g - fg'}{g^2} + 1/2$$

$$= \frac{B(A+B)(TDe^{-TD} - 1 + e^{-TD})}{(D - TAD - B + Be^{-TD})^2} + 1/2$$

A.5) Solve for D When ($R_d' = 0$)

Now, we are going to attempt to solve for $R_d' = 0$. We do this by assuming that $R_d' = 0$ and munching around with the equation.

$$R_d' = \frac{B(A+B)(TDe^{-TD} - 1 + e^{-TD})}{(D - TAD - B + Be^{-TD})^2} + 1/2$$

$$0 = \frac{B(A+B)(TDe^{-TD} - 1 + e^{-TD})}{(D - TAD - B + Be^{-TD})^2} + 1/2$$

So this gives us:

$$-1/2 = \frac{B(A+B)(TDe^{-TD} - 1 + e^{-TD})}{(D - TAD - B + Be^{-TD})^2}$$

$$(D - TAD - B + Be^{-TD})^2 = -2B(A+B)(TDe^{-TD} - 1 + e^{-TD})$$

$$(D(1 - TA) - B(1 - e^{-TD}))^2 = 2B(A+B)(1 - e^{-TD} - TDe^{-TD})$$

Taking the square root of both sides we have:

$$D(1-TA) - B(1-e^{-TD}) = \sqrt{(2B(A+B)(1 - e^{-TD} - TDe^{-TD})}$$

$$D(1-TA) = B(1-e^{-TD}) + \sqrt{(2B(A+B)(1 - e^{-TD} - TDe^{-TD})}$$

$$D = \frac{B(1 - e^{-TD}) + \sqrt{(2B(A+B)(1 - e^{-TD} - TDe^{-TD}))}}{1 - TA}$$

A.6) Calculating Y and Y'

We take the following equation from section A.5 above.

$$(D - TAD - B + Be^{-TD})^2 = - 2B(A + B)(TDe^{-TD} -1 + e^{-TD})$$

Modifying this slightly we have:

$$0 = (D - TAD - B + Be^{-TD})^2 + 2B(A + B)(TDe^{-TD} -1 + e^{-TD})$$

Let's define a function Y as:

$$Y = (D - TAD - B(1 - e^{-TD})^2 + 2B(A + B)(TDe^{-TD} -1 + e^{-TD})$$

The function Y has a value of 0 exactly when $R_d' = 0$. Therefore, if we find the value for D that causes Y to equal 0, we will have found the value for D which causes R_d' to equal 0. This value for D is the optimal value for the timer.

So, as mentioned above in section 7.2, we need to find Y' to use the Newton-Raphson method to approximate where Y = 0. Let's rearrange Y slightly to get:

$$Y = 2B(A + B)(TDe^{-TD} - 1 + e^{-TD}) + ((1 - TA)D - B(1 - e^{-TD}))$$

And we get:

$$Y' = 2B(A + B)(Te^{-TD} - T^2De^{-TD} -Te^{-TD}) + 2((1 - TA)D - B(1 - e^{-TD}))(1 - TA - TBe^{-TD})$$

$$Y' = 2((1 - TA)D - B(1 - e^{-TD})) (1 - TA - TBe^{-TD}) - 2B(A + B)T^2De^{-TD}$$

B) References

[1] HELLAND, PAT. "The Transaction Monitoring Facility (TMF)", IEEE Database Engineering, June 1985.

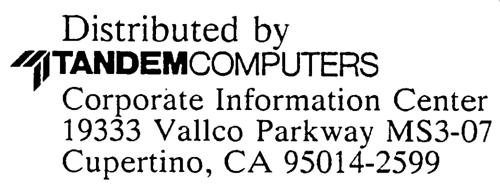
[2] GAWLICK, DIETER, AND KINKADE, DAVID, "Varieties of Concurrency Control in IMS/VS Fast Path",IEEE Database Engineering, June 1985.

[3] ANON, ET AL, "A Measure of Transaction Processing Power", Tandem Technical Report TR 85.2, February 1985.

[4] BORR, ANDREA, "Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing", Proc. Seventh International Conference on Very Large Data Bases, September 1981.

[5] BORR, ANDREA, Robustness to Crash in a Distributed Database: A Non Shared-Memory Multi-Process Approach", Proc. Tenth International Conference on Very Large Data Bases, August 1984.

Distributed by



TANDEMCOMPUTERS

Corporate Information Center
19333 Vallco Parkway MS3-07
Cupertino, CA 95014-2599