D0 note 2708
James T. Linnemann
Michigan State University
January 8, 1996

More on Queueing Theory and Upgrade L2

I present analytical results for deadtime of some simple queues. This work is an extension of D0 note 1767 in that event buffers and farms are considered in a single framework. I manipulate The formulae to show how deadtime can be parameterized and controlled by adding input buffering or adding processors. I finish with some applications to the design of the D0 Upgrade L2 trigger.

1 Program Documentation

Q_SOLVE is an interactive program in D0$UTIL which can answer many queueing theory problems as fast as you can type the inputs. This document is D0$UTIL:Q_SOLVE.PS. To run the queue solver, @d0$util:q_solve to link and run, and thereafter you can $run q_solve.exe on your own copy. The variables are defined in the theory description below.

Q_SOLVE asks you (case insensitive) for values for at least 3 of the four variables P,B,u (or R) ,D. If you answer * for the value, it performs a loop over plausible values:

P: 1 to 100 by 1 for Processors

B: 0 to 50 by 1 for Buffers

R: 0 to 2.5 by .05 (units of fraction of capacity/processor)

D: 0 to 50

If you answer ? for the value, it will solve for that parameter.

If you give an explicit value, it will attempt the solution with that value.

Enter u or R as a decimal fraction eg 1.03 or .06.

Enter D as percent, eg 5 means 5% and .1 means 0.1% .

The program is an endless loop, so control-C when you have had enough.

Output is to the screen and to the file FOR020.DAT; output is supressed if the deadtime is $< .005\%$.

The solution for u uses the CERNLIB routine RZERO C205. The value of u should be accurate to $|u - u_{true}| < .001(1 + u)$.

To help building intuition of the behavior of the approximate solution given in Equation (28) below, both the exact and approximate solutions are reported by the program.

2 Simple Queueing Theory For Triggering and DAQ

This note presents some basic results in queueing theory and shows how close to nominal capacity one can operate for a given deadtime for triggers and for front-end data buffering. These techniques are behind the full simulations of the data acquisition system (of many coupled, parallel queues) performed by Jay Wightman, Marvin Johnson, and others. This note analyzes the results for deadtime from one kind of queue which has a relevance to us and applies them.

2.1 Basics of Queueing Theory

Queueing theory is a branch of probability theory which can be used to relate traffic through a data acquisition to the deadtime incurred. I looked the main results up in a book [1]. Queueing theory typically puts much more emphasis on quantities like queue length and average waiting time than deadtime, but that is often all we need. An important characteristic of the typical solutions is that much or all of the dependence on input rate and processing time can be summarized by their product

$$\boxed{u = InputRate(Hz) \times MeanProcessingTime} \tag{1}$$

I present the results on deadtime for the most important queue for the purposes of a HEP experiment: a buffered multi-server queue. I will use it to model the upgrade L2 trigger . This queue contains as special cases a single-server buffered system (eg Front-end "Double Buffering") and an unbuffered multi-server queue (eg the L3 farm). I will analyze in detail the solution I looked up, in order to extract the main dependences on the number of buffers and the number of processors.

A Queue is just a series of customers (events) waiting in line to be served. The server is what processes the data. In the front-end data acqusition system, the server is the digitization; in L2 or L3 it is a processing node. A Buffer is a waiting spot where a customer can stand before interacting with the server.

```
Buffer system:  -input-> [ ][ ] S        S = Server; [ ] = Buffer
```

This shows a server with 2 buffers in front of it; this system could have three customers in the system before the next customer is turned away (causing deadtime). Although there are only 2 buffers, you will often hear this described as a triple-buffered system because it could hold 3 events. That's because we describe the location of the data being processed by the server as a buffer. A newly arriving event goes directly to the server if it isn't busy; otherwise it goes to the empty buffer nearest the server. If all buffers are full, the event is lost and deadtime is incurred.

The system I am considering is more general: it can have several processors instead of a single server:

```
                          |S
-input->  [ ][ ]....[ ]|S
                          |S
```

Here is a queue with 3 equivalent servers and some number of input buffers. An incoming event goes to any empty server. If all servers are busy, the event goes into one of the B FIFO buffers. If all buffers are already occupied, the event is lost, incurring deadtime. This is a model of the L3 system; it includes the buffering in the VBD's which allow some stacking of the events at the input.

In all of these queues, the time to switch between buffers or servers is ignored.

2.2 Definition of Basic Variables

Queue Notation:

Queues with infinite sources of events and emptied by First In, First Out, are described by a notation I/S/c/K

where I,S describe input and service time distributions and can take one of the following values:

I,S	D	Deterministic, constant time between events
	M	exponential time distribution between events (Poisson)
		or exponential distribution of processing time
	GI,G	general, unspecified arrival, service time distribution
c	is the number of servers (processors)	
		(if you look at the program, I use P)
K	is the maximum number of customers (events) in the queue	
$B = K - c$	is the number of extra buffers in the system	

Other Variables:

$\lambda =$ Input Rate (Hz) to the queue $= 1/ < T_{arrival} >$
$\mu =$ Processing Rate in Hz $= 1/ < T_{process} >$ for a single processor
The traffic intensity is measured in Erlangs, a dimensionless unit:

$$\boxed{u = \lambda/\mu =< T_{process/server} > / < T_{arrival} >} \tag{2}$$

D = probability of the queueing system being full, ie. the fractional deadtime. (D is usually written as p_K, the probability that the queue is full)

The natural queueing theory variables involves the *input* rate λ, not the throughput rate. However we often measure the throughput rate:

$$\lambda_a = \lambda_{actual} = \lambda * (1 - D) \tag{3}$$

Notice that if you measure the deadtime and the throughput of a simple queue, you can measure the service time by

$$< T_{process} >= u/\lambda = u(1 - D)/\lambda_a \tag{4}$$

if you know the relation between D and u.

2.3 Interpretation of u

Recall $u =< T_{process/server} > / < T_{arrival} >$ = the ratio of the time it takes to handle one event to the time between events. It measures how much trouble you're in if you try to handle the data stream with a single processor. Obviously, if $u > 1$, you will need to have more processors to handle the data stream. In fact, u is the mean number of events which arrive during the processing of an average event, or in a deadtimeless system, the mean number of events being processed in the system at any given time.

Define the *capacity* of a processor as $\mu = 1/ < T_{process/server} >$, the maximum number of Hertz it could handle if it was fully occupied. A queue with c identical servers then has a capacity $c\mu$. So we see that $u = \lambda/\mu$ is the ratio of the *input* rate attempted to the single server maximum rate, or the input rate to the queue in units of a single server processing power. For a multiple-server queue, the variable

$$R = u/c = \lambda/(c\mu) \tag{5}$$

gives the input rate in units of the maximum capacity for the whole system of servers, or equivalently, how much trouble each of the individual servers are in trying to handle their slice of the data stream.

$R = u/c > 1$ means that you are attempting to drive the system beyond its maximum capacity, which you won't be able to do because of deadtime.

Loosely, u measures how hard you drive a queue compared to the capabilities of a single server, while R measures the fraction of system capacity you are trying to use. Turning that around, R would give, if not for deadtime (events that never get to processors), the fraction of your system's resources that are occupied on average.

The actual server fractional utilization (fraction of capacity of the server(s) used) is

$$\rho = \lambda_{actual}/(c\mu) = u(1-D)/c = R(1-D) \tag{6}$$

So ρ measures actual (output, or throughput) rate in units of the maximum capacity, so $\rho = 1$ is the limit. Another interpretation of ρ is that it is the average fraction of the servers which are busy. $R = u/c$ is the same as ρ *without* the deadtime correction. Loosely, ρ measures the fraction of the queue's total capacity which you succeeded in using, while $R = u/c$ measures how much you tried to use. $R = 1$ means that the *input* rate is such that it would saturate the capacity were it not for deadtime.

2.4 The Wall

You will see that the solutions for deadtime are expressed in terms of u and R, rather than ρ, which is what can actually be measured. There much less of a "wall" in deadtime vs u, because deadtime changes smoothly even as you adjust the *input* rate through $R = 1$, while it is impossible to exceed $\rho = 1$. The "wall" of deadtime vs rate occurs because what you measure is the actual rate the system achieves, $\lambda_a = \lambda(1-D)$, with service time fixed. To change ρ, the experimentally accessible variable, by a given amount, you have to make a significantly larger change in R. Thus the curve is much steeper near $\rho = 1$ than it is for R near 1.

Further, the expression for $D(\rho)$ is more complicated than for $D(R)$ because $D(\rho) = D(R(1-D))$ involves D on both sides of a highly nonlinear equation. The effect is more simply expressed in terms of the cause alone (R), than a mixture of cause and effect (ρ).

2.5 Combining Deadtimes

If the dead fractions from two sources is independent, then

$$D = D_1 + D_2 - D_1 D_2 \tag{7}$$

2.6 Limitations of these calculations

Our system has external sources of deadtime (eg MRBS LOSS). These results consider only steady state characteristics of the queues, and thus the input rates should be the *instantaneous* rates while we are not suffering MRBS LOSS deadtime. This is only a fair approximation. The time to empty the L3 system is $1/\mu$, about .2 sec, so the L3 queue is largely emptied during the .4 sec MRBS loss period. The remaining cycle time is about 2.1 sec. Under heavy loading, the time t(fill) is about t(empty). It must take several time constants to return to steady state, so the system does not reach steady state early in the remaining 2.1 sec of the cycle. Thus the appropriate rate for L3 purposes is somewhere between instantaneous and average; using instantaneous rates gives conservative deadtime estimates.

For L3, the buffering in the system tends to even out the arrival time distribution so it is closer to equally-spaced than Poisson. This would also tend to make deadtimes smaller than these estimates. The queueing program RESQ which Jay Wightman and Marvin Johnson have been using takes this into account correctly. I believe this correction is small, since I succeeded in reproducing a RESQ curve for a L3 + DAQ system from a curve of deadtime vs input Hz. For the earlier parts of the data acquisition system, only the instantaneous rate is relevant, since the system is idle during MRBS LOSS but loads very quickly (say 4 buffers/λ_{L3} = 40ms).

3 Expressions for Deadtime

The results I show are for a M/M/c/K system, ie Poisson arrival and processing time, c servers, with B=(K-c) additional buffers. The Poisson arrival strictly applies only for pieces of DAQ very early in the system, such as the front end, L1, and maybe L2. Because of the early buffering, later stages (such as writing to data cables, or L3) will see something closer to deterministic arrival times, and will not have to be so highly overdesigned. In general, wider fluctuations in arrival give larger deadtimes for the same u. A fully deterministic system with equally spaced arrivals (a pipeline) can run at full utilization without deadtime.

The solutions I show are the steady state values, and ignore such issues as startup transients, settling time, oscillations (possible when $R > 1$) etc.

The probability that there are n events in the queue can be written in the form

$$p_n = P_n p_0 \tag{8}$$

where p_0 is a normalization factor such that

$$\sum_{n=0}^{K} p_n = 1 \tag{9}$$

This gives

$$1/p_0 = \sum_{n=0}^{K} P_n \tag{10}$$

Deadtime occurs when the system is full, ie $D = p_K$. The solution turns out to be

$$D = \frac{u^c}{c!} R^B p_0 \tag{11}$$

where

$$\boxed{R = u/c} \tag{12}$$

$$1/p_0 = \sum_{l=0}^{c} \frac{u^l}{l!} + \frac{u^c}{c!} \sum_{j=1}^{B} R^j \tag{13}$$

The actual form used in the programs is based on dividing numerator and denominator by $\frac{u^c}{c!}$. This ensures that if any term in the series becomes large, the calculation can be terminated since the deadtime is small.

$$\boxed{D = \frac{R^B}{\sum_{m=0}^{c} \frac{c!}{(c-m)!u^m} + \sum_{j=1}^{B} R^j}} \tag{14}$$

3.1 Analysis of the deadtime formula

The first sum in (13) is associated with the multiple processors, while the second concerns the buffering. I will examine these sums to obtain approximations. Since one purposely designs the system to have low deadtime, the approximations usually need only be valid for small D. This occurs for $R < 1$ and $c > 1$ or $B > 0$, or R small.

Observe that the sum in the multiprocessor part is a truncation of the series for e^u. Define θ as the ratio of the sum to the exponential,

$$\theta(c, u) = e^{-u} \sum_{l=0}^{c} \frac{u^l}{l!} \tag{15}$$

Numerical investigations with Mathematica show that

$$.5 < \theta < 1 \text{ for } R < 1, c < 1000 \tag{16}$$

Obviously $\theta \to 1$ as $c \to \infty$ or as $R \to 0$. But $\theta \to 0$ as $R \to \infty$.

Define b as the buffering sum:

$$b(B, R) = \sum_{j=1}^{B} R^j \tag{17}$$

Since b lies in the range $0 \leq RB$ for $R < 1$, define a factor ϕ

$$\phi(B, R) = \frac{b(B, R)}{R} = \sum_{j=1}^{B} R^{j-1} \tag{18}$$

where ϕ lies in the range $1toB$ for $R < 1$, and ϕ increases with R. For $R \neq 1$, we can also sum the geometric series for b to obtain an exact expression.

$$b(R, B) = \frac{R(1 - R^B)}{1 - R} \text{ or } \quad b = B \text{ for R} = 1 \tag{19}$$

We can now write the deadtime expression as

$$D = \frac{\frac{u^c}{c!} R^B}{e^u \theta(c, R) + \frac{u^c}{c!} R \phi(B, R)} \tag{20}$$

Identify a factor D_c, which is related to the deadtime for an unbuffered set of processors

$$D_c(c, u) = \frac{e^{-u} u^c}{c!} \tag{21}$$

Now analyze D_c to understand its limiting behavior. Applying the trusty Stirling expansion to $c!$,

$$c! \approx c^c e^{-c} \sqrt{2\pi c} \tag{22}$$

which is already accurate to 8% for $c = 1$,

$$D_c(c, u) \approx \frac{e^{-u} u^c}{c^c e^{-c}\sqrt{2\pi c}} \tag{23}$$

Using $R = u/c$,

$$D_c(c, R) \approx \frac{R^c e^{c(1-R)}}{\sqrt{2\pi c}} \tag{24}$$

By considering the sign of $1 - R + \log R$, it can be shown that

$$D_c(R < 1) < 1/\sqrt{2\pi c} < 1 \tag{25}$$

Now returning to the full deadtime expression, we find

$$\boxed{D = \frac{D_c R^B}{\theta + D_c b} = \frac{D_c R^B}{\theta + D_c R\phi}} \tag{26}$$

If we use the approximate form for D_c, and recall $K = c + B$, the number of events held in the system, we find

$$D(R < 1) \approx \frac{R^K e^{c(1-R)}}{\sqrt{2\pi c}(\theta + D_c R\phi)} \tag{27}$$

A crude calculator approximation which is usually good to a factor of 2 comes from taking $\phi = B$ and $\theta = 1$

$$\boxed{\frac{1}{D(R < 1.2)} \approx \frac{\sqrt{2\pi c}}{R^K e^{c(1-R)}} + \frac{B}{R^{(B-1)}}} \tag{28}$$

$$\boxed{D(R > 1.2) \approx 1 - 1/R} \tag{29}$$

A very important point is that for moderate R, the dominant dependence is $D \propto R^K$, that is, the total number of the events held by the queue largely controls the deadtime for reasonable operating conditions ($R < 1$). Both approximations tend to get over-optimistic for $R \geq 1$. Allowing for a possible factor of 2 error in the deadtime is safe, but actually too pessimistic as long as $R < 1$. Equation (28) is reported by the Q_SOLVE program. The expression for $R > 1.2$ is actually a limiting form independent of the particular queue type for $R \gg 1$ (it comes from taking the $\rho = 1$ limit of (6); this expression underestimates the actual deadtime.

An even cruder approximation

$$D(R < 1) \approx \frac{R^K e^{c(1-R)}}{1.9\sqrt{c}} \tag{30}$$

neglects some of the benefits of buffers at moderate to large R so typically it's too high by factors of 1.5 to 3. So don't trust it unless it says you're safe.

3.2 Dependence of Deadtime on Loading

First consider lightly loaded systems, R small. Then b is also small and $\theta = 1$, so in the light loading limit,

$$D(R \ll 1) \approx D_c R^B \approx \frac{R^K e^c}{\sqrt{2\pi c}} < R^B \sqrt{\frac{2}{\pi c}} \tag{31}$$

So for modest loading, the dominant dependence is $D \propto R^K$, depending on the total number of events the system can hold.

Table 1: Single Processor with Buffers
Rmax (fraction of nominal capacity)
for Dead Fraction =

B buffers	.01%	.1%	1%	5%	10%
0	.0001	.0010	.0101	.0526	.1111
1	.0101	.0321	.1057	.2572	.3935
2	.0472	.1037	.2354	.4416	.6021
3	.1028	.1873	.3520	.5767	.7394
4	.1643	.2673	.4475	.6739	.8314

It's also handy to know the situation at $R = 1$

$$D(R = 1) = \frac{1}{\theta/D_c + B} < \frac{1}{\sqrt{\pi c/2} + B} \tag{32}$$

What happens if the system is driven very hard? This means that you will force the system to $\rho = 1$. From the definition of ρ (6), we can solve for $D(R)$ at $\rho = 1$ and find

$$D(R \gg 1) \approx 1 - 1/R \tag{33}$$

This does *not* drop out nicely from the approximate forms such as (27), which is unsuitable for large R because both θ and D_c become small, forcing a re-evaluation of their limit.

4 Results for Particular Queues

4.1 Results for a Single Processor

In this case $R = u/1 = u$ and the exact form is

$$D(c = 1) = \frac{1}{\sum_{m=0}^{B+1} R^{-m}} = \frac{(1-R)R^{(B+1)}}{(1 - R^{(B+2)})} \tag{34}$$

where the second expression is valid for $R \neq 1$. The approximate form, while not far off, does not simplify nicely, though it does extract the R^K dependence. The single server shows fast powerlaw decreases in deadtime as buffers are added to a lightly loaded system, added, but gains come much more slowly for heavier loading.

To obtain low deadtime with a single server, one must design quite conservatively: only a small fraction of the capacity may be used unless one buffers *very* heavily. Ethernet comes to mind, although there the limits are more in the waiting time than the deadtime. In Table 1 I show the maximum value of u consistent with various deadtime goals as a function of the number of buffers.

Finally, the unbuffered single processor is also an interesting case:

$$D(c = 1, B = 0) = \frac{R}{1+R} \tag{35}$$

which can be re-expressed as a formula for the deadtime:

$$D(c = 1, B = 0) = R(1 - D) = \rho = \text{Throughput Hz} \times <!T_{processing}!> \tag{36}$$

4.1.1 Latency for a Single Server

The latency is commonly referred to as the time that it takes for an event to be processed through a queue and server. It includes not only the service time by the server, but also the time spent waiting in the queue. In the steady state, the latency denoted by $<T_l>$ is given by

$$<T_l>= L/\lambda_a \tag{37}$$

where L is the average number of events being served, or waiting in buffers, and λ_a is the deadtime-corrected arrival rate. For the single server queue,

$$<T_l>= \frac{u}{\lambda_a}\frac{1-u^K(K+1-Ku)}{(1-u)(1-u^K)} \tag{38}$$

If the deadtime is small, then u^K is small and the deadtime correction to λ is small and (using $u = \lambda <<t>>$)

$$<T_l>\approx \frac{<<t>>}{(1-u)} \tag{39}$$

4.2 Results for an Unbuffered Farm

For an unbuffered system, $B = 0$ (one approximation to the the L3 farm),

$$D(c, R, B=0) = \frac{1}{\sum_{m=0}^{c}(c!/m!)u^{(m-c)}} = \frac{D_c}{\theta} \approx \frac{R^c e^{c(1-R)}}{\sqrt{2\pi c}} = \frac{R^c e^c e^{-u}}{\sqrt{2\pi c}} \tag{40}$$

The first, exact, expression is valid for an M/G/c/c system, ie Poisson arrival time distribution, and *any* service time distribution (only the mean time matters), c servers and no additional buffering. The deadtime expression sometimes is called the Erlang B function, $B(u, c) = D(u, c, B = 0)$. Recalling that θ lies between .5 and 1, we see the justification for the notation D_c, as it gives an approximation of the deadtime for unbuffered systems. If the number of processors is fixed, $D_c(R) \propto R^c$; this is relevant to understanding an existing system. To choose c in face of a given expected load $u = Rc$, the main dependence is R^c for R small, but $1/\sqrt{c}$ for R near 1.

To evaluate the effect of losing a processing node under a fixed load u, I calculate the ratio $D(u, c-1)/D(u, c)$ exactly and find

$$\boxed{D(u, c-1, B=0) = \frac{1}{R}\frac{D(c)}{(1-D(c))}} \tag{41}$$

This says that you absorb a familiar factor of $1/R$ in deadtime by losing one of your nodes, and things are worse if you are already heavily loaded. A numerical comparison shows that you change the deadtime more by adding a processor than by adding a buffer even when buffers are already in place. This is because adding a processor does two things at once: besides simply adding another slot for an event (which a buffer also does), it adds capacity to the system, lowering the processor utilization fraction $R = u/c$ (u is fixed by the external rate and the speed of the processors).

For fixed capacity $c\mu$, if there are no other constraints , it is better to have more small nodes, since one can get closer to the theoretical capacity. However, $c - 1$ buffers in front of a single processor has a smaller deadtime than c processors each c times slower than the single processor (the c processors would operate at the same R as the single processor).

Table 2: Processors without extra Buffers
Rmax (fraction of nominal capacity)
for Dead Fraction =

c nodes	.01%	.1%	1%	2%	5%	10%
1	.0001	.0010	.0101	.0204	.0526	.1111
2	.0071	.0229	.0763	.1117	.1907	.2977
5	.0904	.1524	.2722	.3314	.4437	.5762
10	.2260	.3092	.4461	.5084	.6216	.7511
15	.3187	.4051	.5405	.6006	.7088	.8323
16	.3337	.4201	.5547	.6143	.7215	.8438
25	.4352	.5188	.6450	.7002	.7994	.9133
45	.5573	.6321	.7429	.7915	.8789	.9814
50	.5773	.6502	.7580	.8051	.8907	.9912

These are instances of what queueing theory tells you repeatedly: if you have resources, it is far better to pool them than to specialize them.

Since the Level 3 system consists of many nodes, it can be run much closer to full capacity than the DAQ system, which typically can hold fewer events (K much smaller). Table 2 gives some numbers for how close.

4.3 Multiple Processors with Shared Buffers

Now, fix the number of processors and add buffers. There are really only two cases, where you need more buffers, and where you don't. If you didn't need the buffers, it's because D_c is already small in (26). In the denominator, only the $\theta(R < 1) \approx 1$ term is important. If you have the sense to keep $R < 1$, the deadtime decreases on adding buffers as a power of R^B. This is the typical numerical situation: from (25), D_c is large only for $R > 1$. If you really needed the buffers, the effect is much the same, but now both terms in the denominator contribute and, since R is no longer small, the buffering gains more slowly in a heavily loaded system.

It is useful to examine the consequences of a buffer failing. The external load is constant, so u and R are constant and we find

$$\boxed{D(u, B-1, c) = \frac{1}{R} \frac{D(B)}{(1 - D(B))}} \tag{42}$$

Remarkably, the expression for $D(B-1)/D(B)$ is precisely of the same form as for $D(c-1, B=0)/D(c, B=0)$. If the initial deadtime is the same, then the effect of losing one node from an $unbuffered$ farm is the same as of losing one buffer of a buffered single processor or multiprocessor system. However, it is still worse to lose a processor than a buffer from a given system, since you both lose event buffering and raise the load on the remaining processors. Sad to say, the expression for $D(c-1)/D(c)$ is complex for $B > 0$.

Let us look at the consequences of having two types of farm nodes which handle different types of events. Suppose we specify that the system has deadtime no worse than 1%. I will compare the split system with a single system of the same number of total nodes. Using the deadtime combination formula (7), and imagining half of the event stream running into each sub-farm, I will require .5% for each sub-farm. I will ignore the cross term when combining the deadtimes of the two farms because I have specified a small deadtime for each sub-farm. The results are shown in Table 3.

Table 3

Maximum Loading Allowed for Multiple Processors with Shared Buffers

1% deadtime total or .5% in each of two sub-farms

c	B	K	Utot	Rmax	
					Initially unbuffered system
10	0	10	4.5	.45	original farm needs 10 nodes w/o buffers
					specs are 1% with 10 nodes
					Lack of buffers for 10 => overbuild by *2
5	21	26	4.5	.90	1/2 the processors in 1 farm for same u
					needs quite heavy buffering
					Split the original 10 nodes
5	0	10	2.3	.23	not OK: allowed u down 50% if farm is split
8	0	16	4.5	.28	add lots of processors to get down to .5% deadtime:
					need to load more lightly for same total u
16	0	16	8.8	.55	but same processors in 1 farm would raise u 95%
5	3	16	4.5	.45	or add 3 buffers per sub-farm added to recover
10	6	32	7.2	.72	but same buffers in 1 farm raises u by 60%
					Buffered Farm
10	10	20	8.1	.81	original farm with 10 buffers and 10 nodes
					=> the extra buffers allow 80% more u 1
5	5	20	5.7	.57	lose 30% u by splitting farm
7	5	24	8.1	.58	need to add 2 nodes/farm to recover u
14	10	24	11.5	.82	but put in 1 farm raise total u 40%
5	15	40	8.1	.81	or MUCH heavier buffering required when farm split
10	30	40	9.5	.95	adding those buffers to farm would raise u 17%

Table 4
L2 options for u = .5
with 1% deadtime

c	B	K	u	R	D(%)
4	0	4	.5	.13	.16
3	1	4	.5	.17	.2
2	2	4	.5	.25	.47
1	5	6	.5	.5	.79

Table 5
u for various scaled-up L2 options
with 1% deadtime

c	B	K	u	R	D(%)	u/.5
12	0	12	5.9	.5	1	x 12
9	3	12	5.4	.6	1	11
6	6	12	4.1	.68	1	8
3	15	18	2.6	.87	1	5

From the table we see that if the number of nodes is rather small, say 10 or so, then one must waste about half the processors as buffers. Splitting capacity into separate farms requires substantial further safety factors, or heavy buffering. Without this safety factor, the number of buffers doubles (or worse). Splitting the processing power across two farmlets raises the number of buffers even more.

Once the system is heavily buffered, it can run fairly close to its maximum capacity. Thus splitting it does not cost quite as much, nor does joining the two equivalent systems gain as much.

These calculations were done assuming that all buffers are shared among any of the processors of a farmlet. If this is not the case and buffers are associated with particular nodes, the deadtime will be worse than those given here, and a RESQ simulation should be performed to see how much worse.

Another useful study is to see how strongly the required number of event buffers and processors depend on the estimated time to complete an algorithm. Suppose (taking the present L1.5 EM algorithm, and the present C40 processors as a benchmark), the estimated time for servicing an L2 request is 50μsec. If we ran the full 10 KHz through such an algorithm, that would give $u = .5$. In Table 4, I consider various combinations of processors and buffers with the goal of achieving 1% deadtime. This system needs front end buffers to hold 4-6 events.

Suppose I triple the number of buffers or processors in the system. I might do this for conservatism's sake since the algorithms are not yet well-specified, or I might use the factor of 3 to represent the possibility of 3 candidates or algorithms per event. For each of the the 4 acceptable solutions, I examine how much capacity I get for 1% deadtime in Table 5.

The scaled-up systems would require buffering for 12 (or 18) events in the front ends. Economics might lead one to choose among the acceptable options. Obviously, the requirements

Table 6
L2 options for u = 1.0
with 1% deadtime

c	B	K	u	R	D(%)
5	0	5	1	.2	.3
4	1	5	1	.25	.4
3	2	5	1	.33	.7
2	4	6	1	.5	.9
1	98	99	1	1	1.0 too many event buffers

Table 7
u for various scaled-up L2 options
with 1% deadtime

c	B	K	u	R	D(%)	u/1.0
15	0	15	8.1	.54	1	8.1
12	3	15	8	.67	1	8
9	6	15	6.4	.71	1	6.4
6	12	18	4.9	.82	1	4.9
3	15	18	2.5	.85	1	didn't really triple buffers: stopped at K=18

go up if substantially longer algorithms are common enough to drive up the mean time substantially.

Finally, let's see how things change if I go through the same procedure assuming that the base algorithm took twice as long (perhaps a more realistic figure). Then the base load is $u = 1.0$; results are shown in Table 6. The base system has not changed too much: front end buffers for 5-6 events should suffice.

Again, tripling the system yields the options in Table 7.

This design seems reasonably robust in scale as long as the processors chosen can handle all the events. The "tripled" design has an achievable throughput capacity which differs only by 25% or so when starting from an initial capacity which differs by a factor of 2. The number of events to be buffered in the front end changes only mildly from 12-15 to 15-18.

References

[1] Probability, Statistics, and Queueing Theory by Arnold O. Allen (Academic Press, 1978). This is a very clear and well-written book, but the author's sense of humor occasionally veers to the juvenile and sexist.

[2] Simple Queueing Theory for Level 2 and DAQ, D0 Note 1767, J. Linnemann; or D0$UTIL:q_solve.doc