Analysis of a Programmed Backoff Method for Parallel Processing on Ethernets

Norman Matloff

Department of Computer Science
University of California at Davis
Davis, CA 95616
matloff@cs.ucdavis.edu

Abstract. In many parallel processing applications, task times have relatively little variability. Accordingly, many nodes will complete a task at approximately the same time. If the application is run on an Ethernet, the near-simultaneity of the task completion times implies that when the tasks attempt to communicate with some central task manager, they will bump into each other. This in turn can cause a major slowdown in communication, as the Ethernet hardware generates unnecessarily long backoff times. The work here will analyze a solution to this problem.

Key words and phrases: Ethernet collisions; task times; programmed backoff.

1 Introduction

On an Ethernet,[1], if during a transmission one or more nodes produce frames to send and test the line, they then attempt to send as soon as the current transmission ends. If more than one node is involved, the nodes collide, generate a random backoff time, and then try sending again.

Now suppose we have a parallel processing application running on a homogeneous set of workstations connected by an Ethernet, and consider task rendezvous frames sent on it. For example, in a message-passing paradigm, we might have root-finding program [2]. Here, a function is known to have a single root in a given interval, which the program finds (to the desired level of accuracy) in a parallel iterative procedure.[2] In any given iteration, the current interval to be searched is divided into n subintervals, where n is the total number of machines. Each "worker" node inspects its assigned subinterval, and then reports to a "manager" node whether the given function has a sign change in that subinterval. Only one of these subintervals will experience such a change, and it will then become the new interval. The manager will broadcast the values of the endpoints of the new interval to the workers, so that they can divide it into

[1] The material here will also apply to other carrier sense multiple access/collision detect (CSMA/CD) local area networks, but for simplicity we concentrate here on Ethernets.

[2] We assume here that the evaluation of the function is lengthy enough to make a parallel search worthwhile. For instance, the function may itself be evaluated through a time-consuming numerical solution of a differential equation.

new subintervals, and so on. Under a shared memory paradigm (in this case distributed shared memory), barrier operations would produce a similar pattern.

A problem which arises here is that in many applications task times (including communication delays) have small degrees of variability [1]. For instance, in the root-finding example above, the function-evaluation times should be fairly uniform. As another example, the mean run time for a Heapsort of r items is O(r log r), while the standard deviation is only O($\sqrt{r}$) [5]; the larger the problem, the smaller the standard deviation is relative to the mean.

In applications in which the tasks at several nodes finish approximately simultaneously, the task rendezvous operations will cause collisions on the Ethernet. The random backoffs which result will then slow down the application. In this context the random backoffs produced by the Ethernet hardware are typically much longer than need be. In [4], an approach to solving this problem was proposed, called *programmed backoff*. Suppose n nodes are currently processing tasks, with the task at node k completing at time T_k. At that time, the software running at node k will produce its own backoff, delaying $k\delta$ time before sending its rendezvous frame to the manager node. The goal is that by having the software produce a small, deterministic backoff we can avoid unnecessarily long backoff times produced by the Ethernet hardware.

Under the programmed backoff procedure, collisions are still possible. At that point, the Ethernet hardware will take over anyway. But hopefully this will be a relatively rare event.

In the work here, we present some theoretical models of the effectiveness of programmed backoff. We are particularly interested in the effects of varying the internode backoff spacing δ, for different task distributions.

2 Investigation

2.1 Analytical

Let f denote the probability density function of each T_k. As our first measure of the effectiveness of programmed backoff, let us determine the expected number of first-round collisions. To this end, let 1_{ij} equal 1 if node i and j collide in the first round, 0 otherwise. Then the total number of first-round collisions is

$$N = \sum_{i}^{n} \sum_{j>i}^{n} 1_{ij}$$

Let τ denote the time to transmit a task rendezvous frame. This typically will be much smaller than task times, since the frame will usually contain very little data (such as a 0-1 variable in the root-finding example, indicating whether this node's subinterval produces a sign change for the function). Let U_k denote the actual time at which transmission begins for node k, i.e.

$$U_k = T_k + k\delta$$

and assume the U_k (equivalently, the T_k) to be independent. Then

$$E(N) = \sum_{i}^{n} \sum_{j \neq i}^{n} P(|U_i - U_j| < \tau) \tag{1}$$

where

$$P(|U_i - U_j| < \tau) = \int \int_{|s-t|<\tau} f(s) f(t - \delta(j-i)) ds dt \tag{2}$$

Equation (1) suggests that E(N) is $O(n^2)$ in magnitude. This suggests that the benefits of programmed backoff grow rapidly with the system size n, speculation which will be confirmed below.

We can get a lower bound on the quantity in (2) as follows.

Lemma: Suppose X and Y are continuous[3] independent random variables with the same variance γ^2 and with EY = EX + d. Then

$$P(|X - Y| > b) \leq \frac{2\gamma^2 + d^2}{b^2} \tag{3}$$

Proof: First define Z to be X - (Y - d), and thus write $E[(X-Y)^2]$ as $E[(Z-d)^2]$. Then the latter quantity will be equal to $2\gamma^2 + d^2$, since Z will have mean 0 and variance $2\gamma^2$. Then letting g denote the density of X-Y, we have

$$2\gamma^2 + d^2 = E[(X-Y)^2] \geq \int_{|u|>b} u^2 g(u) du \geq b^2 P(|X - Y| > b),$$

yielding the result.

Now taking X and Y to be U_i and U_j, respectively, we have that

$$P(|U_i - U_j| < \tau) \geq 1 - \frac{2\sigma^2 + (j-i)^2 \delta^2}{\tau^2}$$

where σ^2 is the variance associated with the density f.

2.2 Simulation

At this point we turn to simulation. Taking as our criterion the expected time η until all n nodes have successfully transmitted a message, our interest will center on the following questions:

- How much of an improvement can programmed backoff bring over simply letting the Ethernet hardware manage transmission?
- With all other factors fixed, how does the optimal value of δ vary with the system size n?

[3] Actually, this condition is could be dropped.

– Let c be a *scale parameter* for a family of density functions for the task times. That is

$$f(t) = \frac{1}{c} h[c(t-q)]$$

for some function h and some constant q. As c increases, we get densities which have similar shapes but are more disperse, and σ will be proportional to c.
It is of interest to investigate how the optimal value of δ varies as c increases.

In the simulations the task time distribution was first taken to be the uniform density U(1-c,1+c). Thus we have a family of distributions centered around a mean of 1.0, with c playing the role of a scale parameter as described above.

Frame transmission time, τ, was assumed to be considerably smaller than 1.0, the mean task time. Specifically, in all the simulations presented here, τ was taken to be 0.1. This is a practical assumption, since otherwise the communication overhead (even without collisions) would be too high for effective speedup due to parallelism. Note also that in many applications of the type we have discussed here, the task rendezvous message is very short. For example, in the root-finding application cited earlier, the message information consists merely of 1 or 0, indicated whether or not a sign change was found in the node's assigned subinterval. (However, the minimum length of an Ethernet frame is 64 bytes [6].) Ethernet hardware backoff was modeled according to the usual binary exponential scheme [6].

Intuitively the quantity (2) will typically be a decreasing function of δ. On the other hand, as δ increases we are adding more delay "at the front end," adding an increasing component to η. Thus we might expect that the graph of η as a function of δ is roughly U-shaped, and this will be seen to be the case.

We begin with a simulation for a small value of c, 0.1, presented in Figure 1 for system sizes 32, 64 and 128. Here we have the near-simulataneity in task time completion which formed the fundamental motivation for our work, so it is not surprising that programmed backoff is shown to be capable of strong speedups in the task rendezvous process, of sizes 292%, 439% and 619% respectively. Note too that the larger the system, the greater the benefit obtainable from programmed backoff.

The optimal value of δ is seen to be relatively constant as a function of n (though showing a slight decreasing trend). The near-constancy makes some sense when viewed in the following context: If the task times were completely constant, the optimal value of δ would be τ; this value would result in a schedule under which the (i+1)st node started transmitting immediately after th i-th.

This reasoning would not apply to the case c = 0.8, shown in Figure 2. Here there is much more variation in task times, and accordingly the speedups in this case, are somewhat more moderate: 158%, 324% and 507%. Yet it is interesting to find that the optimal values of δ are similar to those in the previous case.

As noted above, if the task times were completely constant, the optimal value of δ would be τ. Thus we would expect the optimal δ to be just slightly more than τ in settings with nearly-constant task times. Preliminary simulations conducted by the author for values of τ smaller than 0.1 (not included here) seem to confirm this. Moreover, typically the user can find the value of τ *a priori*, since it is a known function of Ethernet parameters and the user's message length.

However, even with c = 0.8 the task-time distribution has a fairly small standard deviation, so next we turned to the family of exponential distributions, with the parameter c being the mean of the distribution.[4] Figures 3 and 4 correspond to c = 0.1 and c = 0.8, respectively. The results are similar to those of Figures 1 and 2. However, the results for c = 10.0, shown in Figure 5, are quite different. Here task times have enough variation that programmed backoff simply produces superfluous delay over what is needed to avoid backoffs produced by the Ethernet cards.

3 Discussion and Conclusions

We have constructed a theoretical model of the effects of small variability in task times in parallel processing on Ethernets. The model suggests that overall task rendezvous time will be on the order $O(n^2)$, and we have derived a lower bound based on the standard deviation of the task times.

As a potential solution to this problem, we have found that programmed backoff can produce very large speedups in cases in which the task time distribution has a small standard deviation. In addition, the optimal value of δ in such cases appears to be rather insensitive to type of distribution, and appears to be typically about 10-20% larger than the transmission time for a task-rendezvous message.

A number of other approaches to the Ethernet backoff problem may be effective. A tree-based barrier [7, p. 247] imposes a partial ordering among the nodes regarding the sequence in which they send barrier messages, thus preventing most collisions. It has recently come to the author's attention that the Genoa Active Message Machine (GAMMA) [3] has now taken this idea a step further, imposing a linear ordering among the nodes. Note, howeer, that these methods are easiest to implement in applications in which there is only one set of nodes which will be involved in barriers, and that set is fixed throughout the program. It may not be possible to implement such "ordered barrier access sequence" methods in full generality.

References

1. Vikram S. Adve and Mary K. Vernon, "The Influence of Random Delays on Parallel Execution Times," *Proceedings of the 1993 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, May 1993, pp. 61-73.
2. S.G. Akl. *"The Design and Analysis of Parallel Algorithms"*, Prentice Hall, Inc, 1989.
3. G. Ciola, G. Ciaccio and L. Mancini. *GAMMA Project: Genoa Active Message MAchine*, Web page http://www.disi.unige.it/project/gamma.
4. Gregory Davies and Norman Matloff. "Network-Specific Performance Enhancements for PVM," *Proceedings of the Fourth IEEE International Symposium on High-Performance Distributed Computing*, August 1995, pp205-210.
5. G. Gonnet. *Handbook of Algorithms and Data Structures*, Addison-Wesley, 1984.
6. Gilbert Held. *Ethernet Networks* (second edition), John Wiley, 1996.
7. G. Wilson. *Practical Parallel Programming*, MIT Press, 1995.

[4] It is thus not a scale parameter in the sense defined earlier, and in fact the mean equals the standard deviation in this distribution.

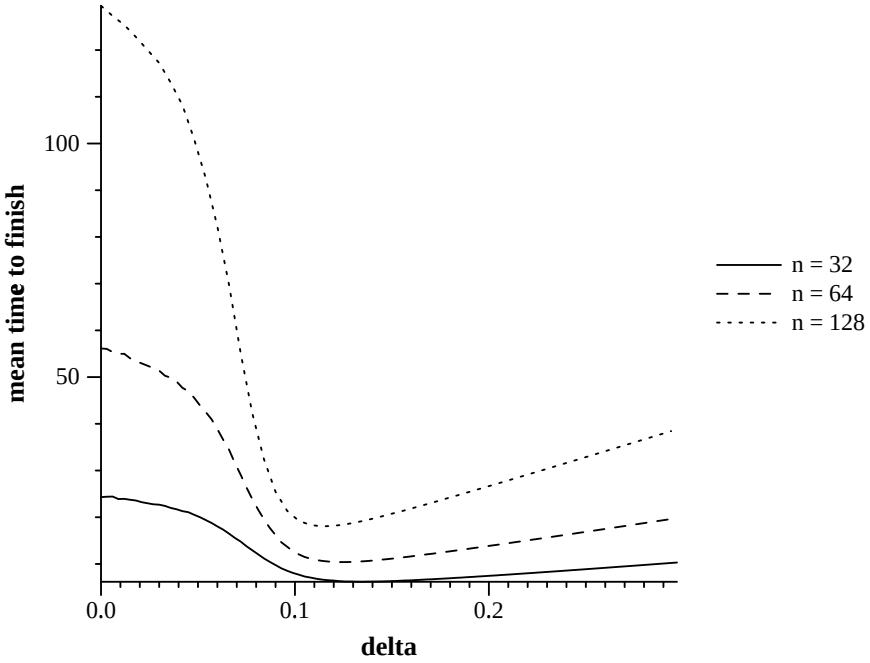

Figure 1

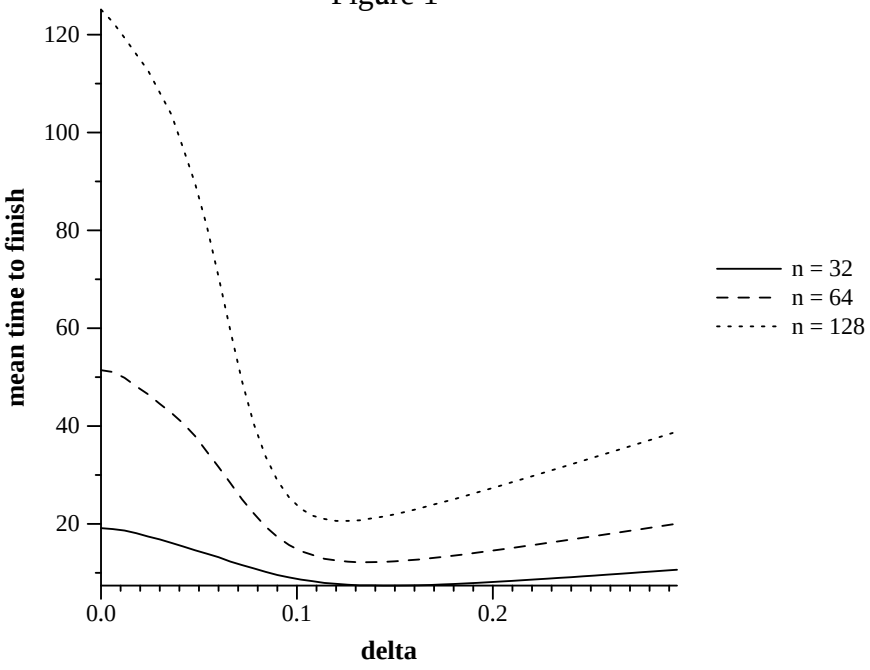

Figure 2

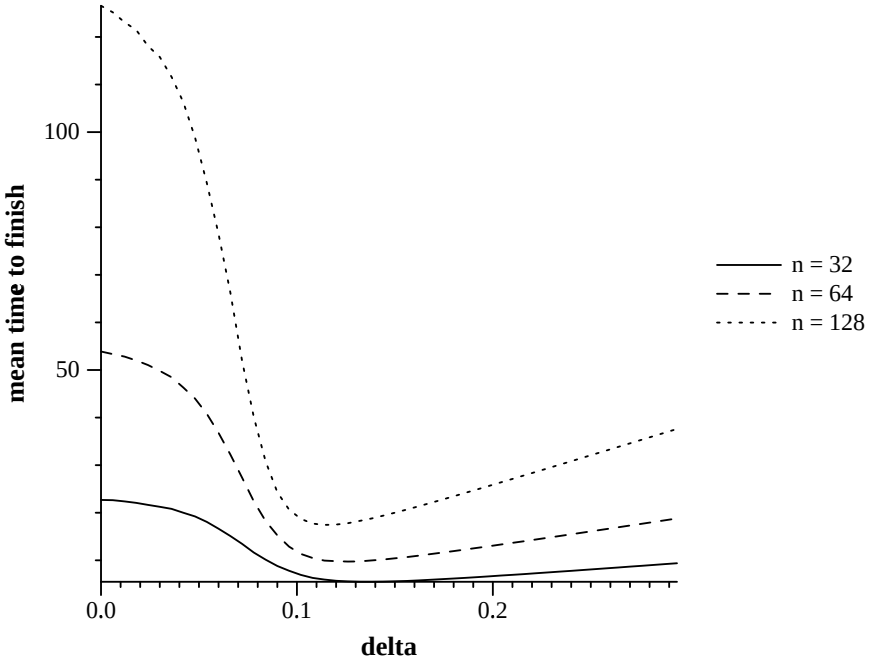

Figure 3

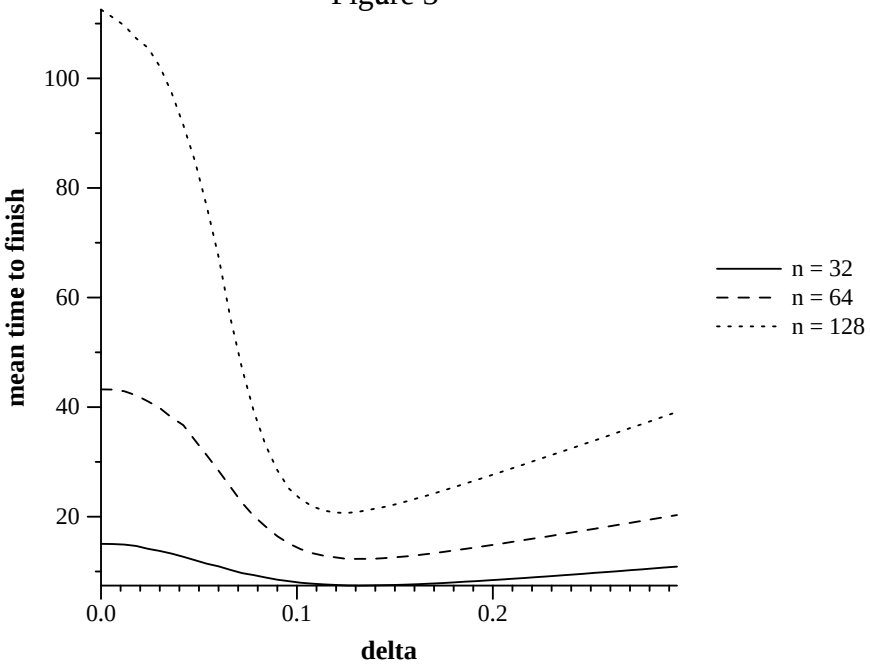

Figure 4

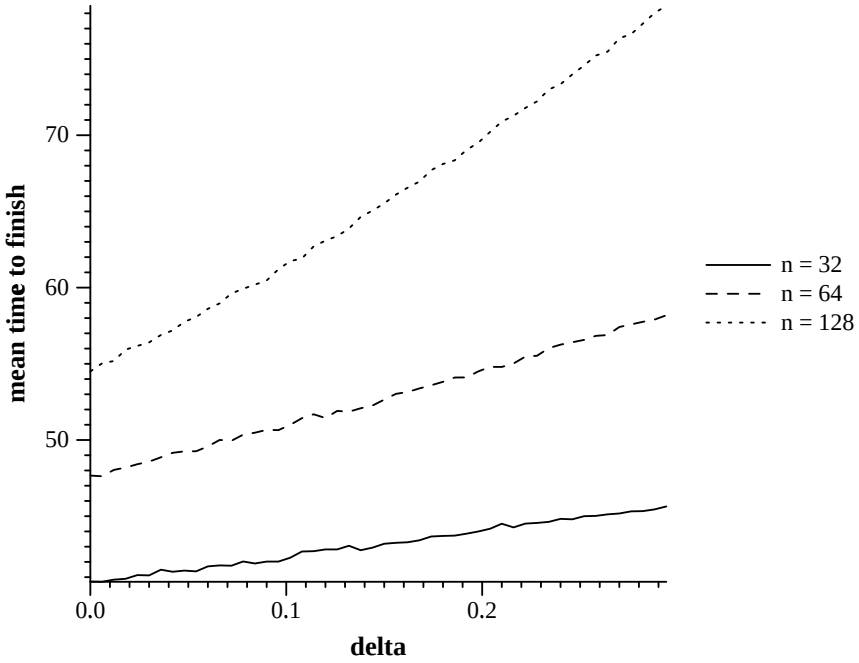

Figure 5