Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

A GENERALIZED HOLD MODEL

Chien-Chun Chou

Department of Computer Science
Tamkang University
151 Ying-Chuan Road
Tamsui, 251, TAIWAN

Steven C. Bruell
Douglas W. Jones

Department of Computer Science
University of Iowa
Iowa City, Iowa 52242, U.S.A.

Wen Zhang

12 Flo Drive
Syosett, New York 11791, U.S.A.

ABSTRACT

The conventional hold model was first analyzed in detail by Vaucher (1977). Several problems have been encountered in applying this model to evaluate the performance of certain pending-event set implementations. For example, the conventional hold model is incapable of taking into account those performance degradation factors that are caused by changes in the size of the event set. Another problem occurs because of the fixed pattern of hold operations performed in the hold model. Finally, the conventional hold model is not suitable for evaluating the performance of concurrent (or parallel) pending event set implementations.

To solve these problems, we describe an extension to the conventional hold model that we call the generalized hold model. Our generalization possesses several interesting mathematical properties that make it useful in studying the behavior of implementations of the event set used in traditional sequential simulators. In addition, we also describe a generalized *concurrent* hold model that permits us to study the behavior of concurrent pending event set implementations. The empirical measurements made with our generalized hold model have borne out the hypotheses concerning the inaccuracies of the conventional hold model as applied to the binomial queue and calendar queue data structures.

1 INTRODUCTION

The conventional hold model was first analyzed in detail by Vaucher (1977). The overall organization of the conventional hold model consists of an initialization, a transient and a steady-state phase. In the initialization phase, a pending event set is initialized with N events. In the transient phase, a series of hold operations is performed to allow the model to reach a steady state. Finally, in the steady-state phase, another series of hold operations is performed in order that the average time taken by each hold operation can be measured. Each hold operation performs the following actions: First, the current event (say, with timestamp t_d) is dequeued from the event set – we refer to this as the *Get-Next* operation. Then, one and only one new event is scheduled with timestamp $t_d + dt$ (where dt is a time value obtained from sampling outcomes from a random variable with cumulative density function $F(x)$) – we call this the *Insert* operation.

It has been shown, for example, by Vaucher (1977) and by Kingston (1985), that as $t_d \rightarrow \infty$, a steady-state condition of the hold model is achieved. In addition, a cumulative density function $G_{hm}(x)$ (which is independent of the time t_d) can be asymptotically derived to represent the distribution function for the relative event times of the events currently in the event set. By applying this asymptotic result, $G_{hm}(x)$, and the fact that the event set size is fixed at $N - 1$ immediately before each *Insert* operation takes place, an average-case cost for the hold operation of various event set implementations can be derived as a function of $G_{hm}(x)$ and N.

Several problems have been encountered in applying the conventional hold model to evaluate the performance of certain event set implementations. In particular, the conventional hold model is incapable of taking into account those performance degradations that are caused by changes in the size of the event set. For example, as shown in the empirical comparisons done by Jones (1986), the binomial queue data structure performs well for some event set sizes and worse for others, depending on the number of one bits in the binary representation of the size. Another example is provided by the calendar queue developed by Brown (1988); here, the major cost of the implementation occurs whenever the size of the event set grows or shrinks by more than a power of two. Another problem occurs because of the fixed

pattern of *Insert* and *Get-Next* operations that results from the exclusive use of hold operations. Some pending event set implementations will deliver their best performance under sequences of hold operations.

The balance of this paper is outlined as follows: Section 2 describes our generalized hold model. The mathematical properties of this model are developed in Section 3. Section 4 provides empirical evidence that the generalized hold model is able to capture the true behavior of several pending event set implementations, whereas the conventional hold model does not do so. Section 5 extends our new model to the domain of parallel processing. This model permits us to study the behavior of concurrent event set implementations. Finally, Section 6 provides some concluding remarks.

2 GENERALIZED HOLD MODELS

Previous attempts at deriving generalized hold models can be found in Ayani (1987), Brown (1988), and Chung, Sang and Rego (1992). Of these, Brown's model is perhaps the weakest, using a random sequence of *Insert* and *Get-Next* operations to grow the event set up to a preset size, and then reversing the probabilities of *Insert* and *Get-Next* to empty the event set. In Ayani's model, after an initial event set of size N is prepared, *Insert* and *Get-Next* operations are performed randomly, with equal probability. The model used by Chung, Sang and Rego is an independently developed generalization of Ayani's; in it, the sequence of *Insert* and *Get-Next* operations is viewed as a Markov chain. For both of these models, the size of the pending event set will vary along a random walk from the desired value of N, but for large N and modest numbers of *Insert* and *Get-Next* operations, this allows for reasonably accurate and reasonably realistic measurements of hold time as a function of N.

Although random sequences of *Insert* and *Get-Next* operations are far more realistic than the fixed sequence of operations in the conventional hold model, the resulting behavior of the event set becomes difficult to model analytically. Specifically, the random walk in the size of the event set under these models is uncontrolled, and as a result, although these models allow for practical measurements, they never reach a steady state in the strict sense of the term.

Our generalized hold model is based on the behavior we observe in closed queueing network simulations. In such simulations, all events represent the end of service at some server. An end of service event may result in zero, one or two new events being scheduled: One new event is scheduled if the server that completed has a non-empty queue, and one new event is scheduled if the customer is passed to a server that had an empty queue.

In our model, we use a generalized hold operation that operates as follows: First, as in the conventional hold operation, we use *Get-Next* to remove one event notice from the pending event set. Following this, the generalized hold operation schedules from zero to two new events, where the time-stamp of each new event is independently determined as in the conventional hold operation as $t_d + dt$. The number of new events scheduled is determined by the type of the event notice.

When our generalized hold operation processes a type-1 event, it schedules exactly one new event. In this case, the new event will be a type-1 event, and the generalized hold operation will be exactly the same as a conventional hold operation.

When our generalized hold operation processes a type-2 event, it schedules two new events. One of these will be another type-2 event, and one will be a type-3 event. We introduce a new pending event set operation *Set-Insert* to allow the insertion of a set of new items in the pending event set as a single operation because for some pending event set implementations, merging a set of new items into the event set can be done significantly faster than a sequence of independent insert operations.

Finally, when our generalized hold operation encounters a type-3 event, it schedules no new events. The following section discusses the net effect of the combination of type-1, type-2 and type-3 events with our generalized hold operation.

3 MATHEMATICAL PROPERTIES

In the generalized hold model, a type-1 event always schedules another type-1 event. We can view the scheduling process for type-1 events as a *renewal process* with recurrence times $(T_1, T_2, \ldots T_i)$ being independent and identically distributed with a cumulative density function $F(x)$. Moreover, a type-2 event always schedules another type-2 event and an additional type-3 event. The scheduling process of type-2 and type-3 events is also a renewal process, where type-2 events represent renewal points. Furthermore, the recurrence time distribution of type-2 events has the same recurrence time distribution, $F(x)$, as type-1 events.

When the generalized hold model has reached a steady state so that we can ignore the specific time t_d, we can analytically determine: (1) the cumulative density function of the relative event times of events, (2) the average number of events in the event set with

event times less than x, and (3), the average event set size and the distribution of event set sizes prior to calls to *Get-Next* by the generalized hold operation.

To simplify our analysis, we assume that the pending event set is initialized with equal numbers (N) of each type of event. Let us define $G_{ghm}(x)$ as a steady state cumulative density function (cdf) which represents the number of events of either type-1, type-2 or type-3 with relative event times $\leq x$:

$$\begin{aligned} G_{ghm}(x) &= \lim_{t_d \to \infty} G_{ghm}(t_d, x) \\ &= \lim_{t_d \to \infty} P[(t_c - t_d) \leq x] \end{aligned} \quad (1)$$

Where t_c is the timestamp on the next event c. The type of an event c, can only take one of three values, so we can further represent $G_{ghm}(x)$ as

$$\begin{aligned} G_{ghm}(x) = \lim_{t_d \to \infty} \quad \{ & \quad P[(t_c - t_d) \leq x | \text{type1}] P[\text{type1}] \\ & + \quad P[(t_c - t_d) \leq x | \text{type2}] P[\text{type2}] \\ & + \quad P[(t_c - t_d) \leq x | \text{type3}] P[\text{type3}] \\ & \} \end{aligned} \quad (2)$$

As we have mentioned, the scheduling process of a type-1 event is a renewal process with recurrent time distribution $F(x)$. From renewal theory (see, for example, Ross (1983)), when $F(x)$ is a non-lattice distribution, the cdf of the relative event time (the remaining time) for type-1 events is

$$\lim_{t_d \to \infty} P[(t_c - t_d) \leq x | \text{type1}] = \frac{1}{\mu_F} \int_0^x [1 - F(t)] dt \quad (3)$$

where μ_F is the mean of $F(x)$, t_c is the timestamp of a type-1 event, and t_d is the timestamp of the current event. The scheduling process of type-2 events is also a renewal process with the same recurrent time distribution $F(x)$, so we can use equation 3 for type-2 events by the simple substitution of type-2 for type-1 throughout.

The scheduling process of type-3 events is based on each renewal of a type-2 event, so the cdf of the number of type-3 events with relative event time $\leq x$ can be derived as follows: Let $\phi(t_d)$ represent the relative event time (a random variable) for type-3 events with the current event having timestamp t_d. Now, we can write

$$\lim_{t_d \to \infty} P[(t_c - t_d) \leq x | \text{type3}] = \lim_{t_d \to \infty} P[\phi(t_d) \leq x] \quad (4)$$

Furthermore, by considering all cases at the first renewal point and by applying Laplace transforms, as well as Blackwell's Renewal Theorem (see Chou (1993)), we have

$$\lim_{t_d \to \infty} P[\phi(t_d) \leq x] = \frac{1}{\mu_F} \int_0^x [1 - F(t)] dt \quad (5)$$

By substituting (3), (4), and (5) into (2), we can obtain $G_{ghm}(x)$ and $g_{ghm}(x)$, the pdf, as

$$\begin{aligned} G_{ghm}(x) &= \frac{1}{\mu_F} \int_0^x [1 - F(t)] dt \\ &\text{and} \\ g_{ghm}(x) &= \frac{1}{\mu_F} [1 - F(x)] = \frac{1}{\mu_F} \int_x^\infty [f(u)] du \end{aligned} \quad (6)$$

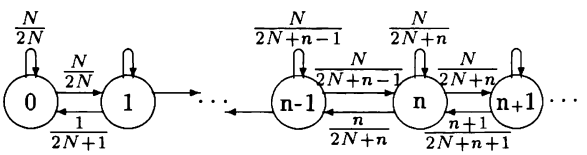
By using equations (6) and applying integration by parts (see Chou (1993)), the expected number of events with relative event times $\leq x$, i.e., the number of events in $[t_d, t_d + x]$, is

$$\begin{aligned} E[G_{ghm}(x)] &= \int_0^\infty G_{ghm}(x) f(x) dx \\ &= 1 - \frac{1}{\mu_F} \int_0^\infty [F(x) - F^2(x)] dx \end{aligned} \quad (7)$$

Remark: If the average size of the pending event set is m events at the beginning of the *Insert* operation, the expected number of events with relative event times $\leq x$ becomes

$$m - \frac{m}{\mu_F} \int_0^\infty [F(x) - F^2(x)] dx \quad (8)$$

We now consider the distribution of pending event set size. At any time t the number of type-1 and type-2 events in the event set is equal to the number of type-1 and type-2 events that the system was initialized with (2N). Since all events are independent, we can employ the Markov chain shown below, where each state corresponds to a different number of type-3 events in the event set.



Given that p_n is the probability of n type-3 events in the event set at the beginning of a generalized hold operation, that is, $p_n \leq 1$, for all $n = 0, 1, 2, \ldots$ and $p_n = 0$ for $n < 0$, and given that the summation of all p_n is 1 and the right side of equation (9) represents flows into state n, we can balance the flows into and out of each state to obtain the following recurrence relations:

$$\frac{n+N}{2N+n} p_n = \frac{n+1}{2N+n+1} p_{n+1} + \frac{N}{2N+n-1} p_{n-1} \quad (9)$$

These recurrence relations provide us with a way to compute the steady state distribution of the size of the event set. Applying z-transforms (see Chou (1993)), allows us to compute the expected size and the variance in the expected size of the event set:

$$m = 3N + \frac{1}{3} \quad \text{and} \quad \sigma^2 = N + \frac{2}{9} \tag{10}$$

Furthermore, the distribution of the number of type-3 events is

$$p_n = \frac{nN^{n-1} + 2N^n}{3e^N n!} \tag{11}$$

Note that all of the results in equations (10) and (11) are independent of the scheduling distribution F(x).

4 EMPIRICAL EVIDENCE

An Encore Multimax model 320, with fourteen NS-32532 processors and one level of cache memory was used as a testbed. For our first experiment, we implemented the generalized hold model using a linked-list data structure for the event set. With $N = 100$ initial type-1, type-2 and type-3 events, we ran the generalized hold model with $NH_t = 10,000$ generalized hold operations to reach the steady state. After every 1000 generalized hold operations, we collected a data sample by recording the relative times of every pending event. One hundred such samples were collected (30000 samples of relative event times were collected. These empirical observations were a near-perfect match to the predictions from equation (11).

For our second experiment, we ran the generalized hold model with $N = 25$, $NH_t = 10,000$ and an exponential scheduling distribution with mean 4. We recorded the size of the event set after each generalized hold operation and collected 100,000 samples. The distribution of these sample values was again an almost perfect match to the distribution predicted by equation (11).

Finally, we applied our generalized hold model to the pending event set implementations used in the empirical comparisons done by Jones (1986); these were the binomial queue, implicit heap, leftist tree, linked list, pagoda, Henriksen's implementation, skew heaps, splay tree and two-list implementation. In addition, we measured the performance of the calendar queue implementation proposed by Brown (1988). For each implementation, we measured the average time per generalized hold operation as a function of the initial size of the event set, and compared this with the time per conventional hold operation.

For most queue implementations, the our generalized hold model gave results that were indistinguishable from the results obtained using the conventional

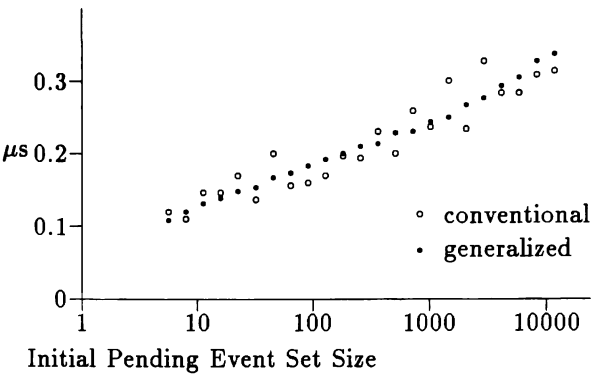


Figure 1: Binomial Queue

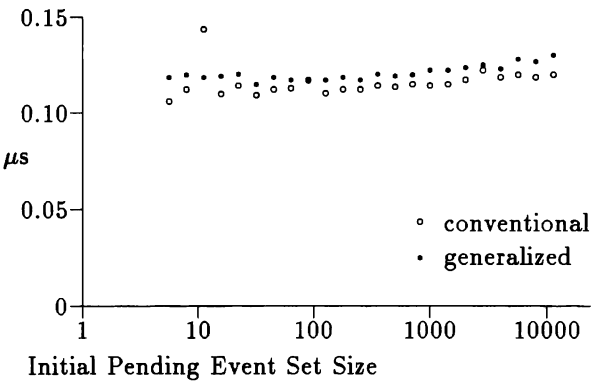


Figure 2: Calendar Queue

hold model. The exceptions were in the cases of binomial queues and calendar queues, as shown in Figures 1 and 2. Under the test load provided by our generalized hold model, the binomial queue exhibits a run-time per generalized hold operation that varies smoothly as a function of event set size. We believe that this is more likely to be a useful prediction of performance in realistic applications than the erratic behavior of the binomial queue under the conventional hold model.

In the case of the calendar queue, our measurements taken under the generalized hold model also show more averaged behavior than from the conventional hold model, but they are also systematically higher. We believe that this increase in the measured cost over the conventional hold model reflects the cost of the changes in queue size caused by our generalized hold model, and we believe that this cost will be typical of the costs incurred in real applications.

5 A CONCURRENT HOLD MODEL

As was the case with Ayani (1987) the motivation behind our work has been to develop a test model suitable for measuring the performance of pending event set implementations on multiprocessors. This has been motivated by our development of the concurrent simulation algorithm proposed by Jones (1986) and most fully explored by Chou (1993). Here, we will only briefly discuss key aspects of our concurrent hold model.

The initialization phase of our concurrent hold model is essentially the same as for the sequential version of our generalized hold model. We allocate space for a pending event set and fill it with N events from each of the three event classes, with random times selected according to the analytically determined steady state distribution of relative times. This avoids the need for a transient phase. In addition to this, we spawn $P-1$ processes so that we have a total of P processes competing for access to the pending event set.

Pseudocode for the steady state phase of one process is shown below. All P processes execute this code in parallel. The idea is that each process is to perform NH_s generalized hold operations. As described by Jones, et al. (1989), we recognize that access to the shared event set will involve critical sections, so we pre-compute all random numbers prior to each generalized hold operation, using an independent random number stream for each of the P processes.

```
steady-state phase:
    repeat NH_s times
        — computation phase
        as needed, deallocate e,
        compute random dt_1 and dt_2,
        allocate event records e_1 and e_2.
        — generalized hold operation
        Get-Next(e)
        case e.type of
        type-1: — insert a new event
            e_1.type = type-1
            e_1.time = e.time + dt_1
            Insert(e_1)
        type-2: — insert two new events
            e_1.type = type-2
            e_1.time = e.time + dt_1
            e_2.type = type-3
            e_2.time = e.time + dt_2
            Set-Insert(e_1, e_2)
        type-3: — insert no new event
        end case
    end loop
```

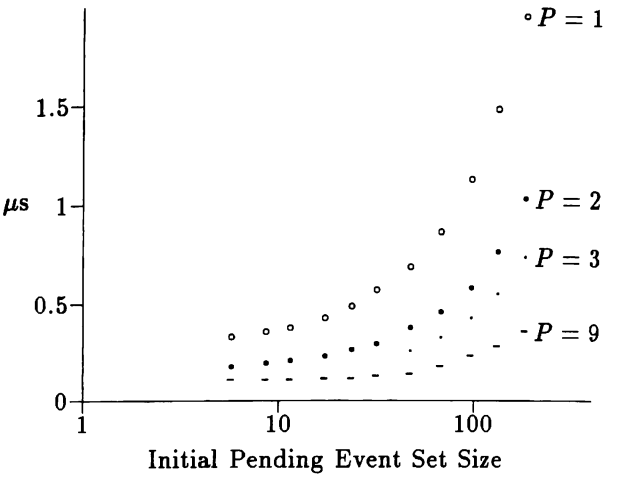


Figure 3: Concurrent Hold Model Data

We used a linked-list implementation of the concurrent pending event set with this generalized concurrent hold model; the details of this implementation have been documented by Chou (1993) and outlined by Jones, et al. (1989). Empirical results are plotted in Figure 3, where the number of processors P was varied from 1 to 8 for each of the initial pending event set sizes.

6 CONCLUSION

We have described an extension to the conventional hold model that we call the generalized hold model. Our generalized hold mode possesses several interesting mathematical properties that make it useful in studying the behavior of implementations of the pending event set used in traditional sequential simulators. In addition, we also described a generalized concurrent hold model that permits us to study the behavior of concurrent pending event set implementations.

It is important to note that our measurements have shown that the conventional hold model is quite robust. Nonetheless, there are some data structures for which the conventional hold model does not provide accurate performance predictions. It is only through the use of tools like our generalized hold model that one can discover this.

It should be straightforward to generalize on our work so that differing initial numbers of type-1 and type-2 events can be used to adjust the variance in the pending event set size. (The initial number of type-2 and type-3 events should always remain equal!) Additional work is needed to develop better imple-

mentations of the concurrent pending event set, and we would like to see theoretical analysis of the performance of algorithms for the concurrent pending event set done under our concurrent generalized hold model.

ACKNOWLEDGEMENTS

When so many people have been involved in a project, division of credit becomes a problem. D. W. Jones developed the concurrent simulation algorithm that led to the need for our concurrent hold model, and Wen Zhang developed the basic model and began the job of developing the underlying theory. Chien-Chun Chou finished the theoretical work described here, and did the empirical studies to back up the theory. S. C. Bruell provided overall supervision and a tremendous amount of helpful advice, and Chou and Jones did most of the writing.

REFERENCES

R. Ayani, 1987. Performance of Priority-Queue Implementations on Shared Memory Multiprocessor Computer Systems. Technical Report TRITA-CS-8705, Department of Telecommunication Systems - Computer Systems, The Royal Institute of Technology, Stockholm.

R. Brown, 1988. Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. *Communications of the ACM, 31* 10 (October), 1220-1227.

C. C. Chou, 1993. Parallel simulation and its performance evaluation. PhD dissertation, Technical Report 93-02, Department of Computer Science, University of Iowa, Iowa City, Iowa.

K. J. Chung, J. Sang, and V. Rego, 1992. A Performance Comparison of Simulation Calendar Algorithms: An Empirical Approach. Technical Report CSD-TR-92-060, Computer Science Department, Purdue University, West Lafayette, Indiana.

D. W. Jones, 1986. An empirical comparison of priority-queue and event-set implementations. *Communications of the ACM, 29* 4 (April), 300-311.

D. W. Jones, 1986. Concurrent simulation: An alternative to distributed simulation. In *Proceedings of the 1986 Winter Simulation Conference,* 417-423. Institute of Electrical and Electronics Engineers, San Francisco, California.

D. W. Jones, C. C. Chou, D. Renk, and S. Bruell 1989. Experience with concurrent simulation. In *Proceedings of the 1989 Winter Simulation Conference,* 756-764. Institute of Electrical and Electronics Engineers, San Francisco, California.

J. H. Kingston, 1985. Analysis of tree algorithms for the simulation event list. *Acta Informatica, 22,* 15-33.

S. M. Ross, 1983. *Stochastic Processes.* John Wiley, New York.

J. G. Vaucher, 1977. On the distribution of event times for the notices in a simulation event list. *INFOR, 15,* 2 (June), 171-182.

AUTHOR BIOGRAPHIES

CHIEN-CHUN CHOU is an Associate Professor of Computer Science and Information Engineering at Tamking University. He received his MS and PhD degrees in computer science from the University of Iowa in 1987 and 1993, respectively, and his BS degree in computer science from Chung-Yuan Christian University (Taiwan) in 1983. His research interests center on concurrent simulation and the theoretical methods applied to the analysis of parallel programs.

STEVEN C. BRUELL is an Associate Professor of Computer Science at the University of Iowa. He received his BS with honors from the University of Texas at Austin in 1973 and his MS and PhD degrees from Purdue University in 1975 and 1978, respectively. In addition to work in parallel simulation, he has worked in the area of generalized stochastic Petri nets and in the area of self stabilizing distributed systems. He has also co-authored three books.

WEN ZHANG received her MS degree in computer science from the University of Iowa in 1988. She received her BS degree in computer science from Fudan University (China) in 1985. Her research interests have centered on theoretical computer science.

DOUGLAS W. JONES is an Associate Professor of Computer Science at the University of Iowa. He received his BS degree in physics from Carnegie Mellon University in 1973, and his MS and PhD degrees in computer science from the University of Illinois in 1976 and 1980 respectively. His research interests focus on distributed simulation, pending event set data structures, and high performance simulation of computer architectures.