# Study of Optimised bucket widths in Calendar Queue for Discrete Event Simulator

T.Siangsukone C.Aswakul L.Wuttisittikulkij
Center of Excellence in Telecommunication Technology Department of Electrical Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, 10200, Thailand. Phone +662 218-6908 Fax +662 218-6912
E-mail todsaporn.s@student.chula.ac.th chaodit.a@chula.ac.th lunch@ee.eng.chula.ac.th

**Abstract**

In this paper, a study of optimised bucket widths in calendar queue for discrete event simulator is presented. It is shown that the performance of calendar queue can vary noticeably depending on the chosen width factor ($W_f$). Existing algorithms appears to perform reasonably well in general situations. However, under certain scenarios these algorithms may deteriorate due to the highly skewed probability distribution of incremental event time. In such scenarios, the calendar queue with optimum width factor $W_{fopt}$ has been found to accelerate up to twice as fast as existing algorithms.

Keywords : Discrete Event Simulation, Calendar Queue

## 1 Introduction

Discrete Event Simulation (DES) is a widely developed simulation method where the simulation clock is driven by events. An event is here defined as the situation that can happen in the considered system and change the system state variables. Events in DES are scheduled in a list (named Pending Event Set, PES) to occur in any future time. Typically, this list of events is implemented as a priority queue where the event priority is associated with the occurrence time of event, $t(e)$, and the event with the smallest priority is executed first.

In any DES, there are 3 main routines: (i) to find the next event from the list to be executed, (ii) to execute the event and (iii) to insert new events, being invoked from the execution, to the list. Since a DES needs to perform at least one event operation (dequeue or enqueue) in every program cycle, that is, in routines (i) and (iii), respectively, it becomes necessary that the priority queue algorithm selected for PES implementation must be able to handle all the event operations in the most efficient way.

Various priority queue algorithms (e.g. [1]-[5]) have been proposed with the aim of reducing the time complexity required for a large PES. Among these algorithms, calendar queue [5] has gained the most popularity due to its $O(1)$ time complexity, given that the calendar parameters have been properly set. In Section 2, we will illustrate the structure of calendar queue. Section 3 discusses about the effect of improperly selected calendar queue parameters. Section 4 presents the analysis of optimal bucket width for calendar queue. Experimental results of calendar queue performance at different bucket width selections are then given in Section 5. Finally, Section 6 gives a conclusion of findings and suggests some directions worthy a future investigation.

## 2 Calendar Queue [5]

A calendar queue is defined as an array of lists, each of which contains future events. To show the advantage of calendar queue algorithm, let us compare it with a linear list [10].

For time $t > 0$, let $N(t) \geq 0$ denote the number of active events being scheduled to occur in our PES. When the simulation is running in its steady state, the number $N(t)$ may be approximately constant, hence dropping the temporal argument $t$. In this case, if we schedule events by using a linear list, then $O(N)$ bound on the time taken by event operations follows because, in an enqueue operation, we may have to search all the $N$ events until we find the proper position to place the enqueued event. If $N$ is too large (e.g. when a big system is simulated), then our DES will not finish its task within an acceptable time. It is generally agreed that any algorithms that require the time complexity greater than $O(1)$ may become impractical in any large DES.

To avoid unnecessarily large time complexity, the principle of calendar queue is to partition the large list of $N$ events into $M$ shorter lists and each list is called a bucket. Bucket is a list with a specified range of admission times. Only events that occur in this range are allowed to be scheduled in the bucket. Without loss of generality, let every bucket have an equal width of $\delta$. Any event with the occurrence time $t(e)$ will be associated with the $m$-th bucket in year $y$ ($y = 0, 1, 2, \ldots$) if and only if $t(e) \in [((yM + m)\delta), (yM + m + 1)\delta)$.

Since the number of events in each of the $M$ lists is typically small, any priority queue algorithms with low overheads can be conveniently utilised for the bucket discipline operations without any significant effect on the order of calendar queue time complexity.

CP08

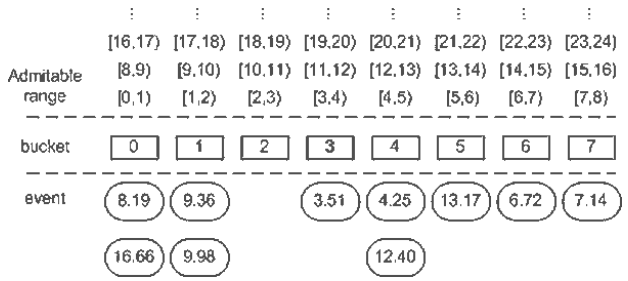


Figure 1: Calendar queue structure ($M = 8, N = 10, \delta = 1$).

Recall that there are two main operations, enqueue and dequeue, in PES management. To find the bucket number $m(e)$ to enqueue an event $e$ that occurs at time $t(e)$, we may apply

$$m(e) = \left\lfloor \frac{t(e)}{\delta} \right\rfloor \mod M \qquad (1)$$

which results in the $O(1)$ time complexity. For a dequeue operation, an index variable may be introduced to identify the most recently dequeued bucket. From this bucket onwards, the algorithm can then search for the next event to be dequeued. An example of calendar queue is illustrated in Figure 1.

In Figure 1, the most recently dequeued bucket is the third bucket. Each bucket is implemented as a linear list. Thus, a dequeue operation can be easily performed by taking out the event on top of the current bucket's list. To enqueue an event $e$ into the calendar, the target bucket is easily obtained by (1) and the event $e$ can then be inserted into the list of the target bucket. Ideally, the calendar queue requires the time complexity of $O(1)$ for both enqueue and dequeue operations if the optimal calendar parameters are selected. An analytical approach for the optimal tuning of calendar has been proposed in [9]. However, for the sake of analysability, it is resorted in [9] to a few assumptions (to be further clarified in Section 4) which may not be justified in practice. This paper is aimed at investigating the effects of such injustification and proposing an alternative way of modifying the obtained formula in [9] for practical usage of calendar.

## 3 Calendar Queue Parameters

There are two control parameters in the calendar queue. The first parameter is the number of buckets $M$. On one hand, if $M$ is too small, then the partitioned list in each bucket may become large. This can result in an increase in the time required for the bucket operations, especially when the number of events $N$ is huge. On the other hand, setting $M$ to a too large value will gain nothing of substantial advantage in terms of computational complexity. The reason is that a calendar queue with too many buckets -though decreasing the time complexity- will unavoidably increase the memory requirement. Based on rigorous empirical studies (e.g. [5] - [8]), it has been found that a plausible value of $M$ should be $M = 2N$ (whose value will also be utilised in this paper).

The second parameter is the bucket width ($\delta$). This parameter affects how a calendar queue algorithm performs in both dequeue and enqueue operations. For dequeue operations, if $\delta$ is too small, then most of the events in each bucket's list will become the events being scheduled to happen in the next years. This causes the calendar to search unnecessarily many buckets before reaching the bucket that contains the next dequeue event. However, for enqueue operations, if $\delta$ is too large, then most of the events will be placed in the current bucket (or only few next buckets), which causes a long list to occur within each bucket and leaves many other buckets mostly empty. The optimal values for bucket width will be the focus of investigations in this paper.

## 4 Calendar Queue Analysis

The analytical study of calendar queue parameters has been carried out [9] under the condition that the number of active events in the calendar is constant (e.g. at its steady state). Further, given that a dequeue event at time $t$ generates a subsequent event at time $t + \tau_i$, the incremental time $\tau_i$ is defined as an independent and identically distributed random variable with mean $\mu$. The analysis in [9] relies on the following assumptions.

[A1] The calendar queue contains an infinite number of buckets, i.e., $M = \infty$.

[A2] A direct search algorithm is applied for all dequeue operations within each bucket's list.

[A3] The processor times needed to process an empty bucket ($\tau_b$) and to process a list element ($\tau_l$) are all constant.

Based on assumptions [A1] - [A3], it is possible to derive for the optimal value of bucket width, $\delta_{opt}$, which minimises the expected time to process an event [9]:

$$\delta_{opt} = \sqrt{2\tau_b/\tau_l}\,\frac{\mu}{N} + O(N^{-3/2}) \qquad (2)$$

However, the formula (2) should only be used cautiously. Due to the limited memory resource in most computing facilities, the number of buckets ($M$) cannot be infinitely increased; hence, the invalidation of [A1]. Furthermore, instead of the direct search [A2], the linear list may be employed to achieve a better performance of the calendar queue. Regarding assumption [A3], the processing time may be more reasonably assumed dependent on the amount of memory usage and the computational requirement for generating ran-

Table 1: Distribution of $\tau_i$

| Distribution[(a)] | Expression to compute random variate[(b)] |
|---|---|
| exponential | $-\ln rand$ |
| triangular | $1.5 rand^{0.5}$ |
| negative Triangular | $3((1-rand)^{0.5}+1)$ |
| bimodal a | 0.0025rand + if $rand$ ¡ 0.5 then 0.94875 else 1.04875 |
| bimodal b | 0.8rand + if $rand$ ¡ 0.5 then 0.1 else 1.1 |

(a)Mean of $\tau_i$ is normalised to 1 in every case.
(b)Function rand returns a value uniformly distributed between 0 and 1.

dom event times according to different probability distributions.

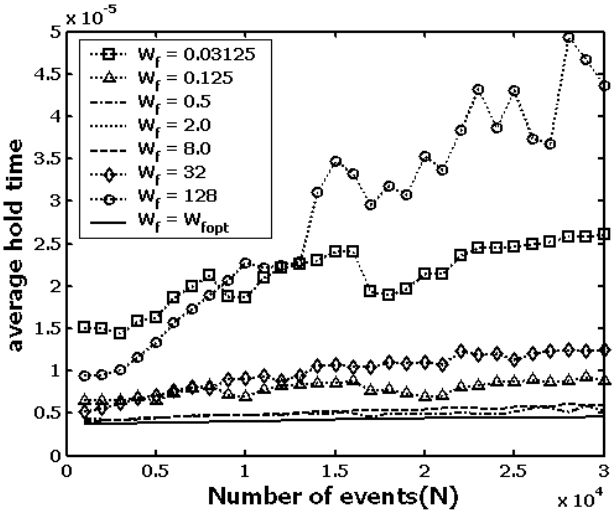


Figure 2: average time per hold operation.

## 5 Study of Optimal Bucket Width

In doubt of the justification for [A1] - [A3] , this paper proposes an alternative practical study for the optimal bucket width. Because our study focuses on a large simulation system, the number of active events ($N$) becomes unavoidably large, which lets the term $O(N^{-3/2})$ be approximately insignificant. Thus, formula (2) can be rewritten as

$$\delta_{opt} = W_f \frac{\mu}{N} \tag{3}$$

where a new parameter, called width factor ($W_f$), is here introduced to summarise all the processor time parameters. [1]

In our experiments, the widely adopted hold model (e.g. [1]-[6]) is employed to evaluate the performance of calendar queue over a wide range of $W_f$. The distributions of $\tau_i$ as detailed in Table 1 are here studied. All the experiments are performed on an INTEL P4 2.0GHz with 512Mb RAM and Microsoft Windows XP operating system.

Figure 2 plots the resultant average time per hold operation when the bucket width of calendar queue is computed from 3 with $W_f = 2^i$ $(i = -5, -4, \ldots, 7)$ at various values of $N$ when the employed distribution for $\tau_i$ is bimodal a. The figure shows that the optimal width factor $W_{fopt}$, which gives the minimum time to perform a hold operation, should be set to 2. Choosing $W_f$ too far from its optimum $W_{fopt}$ results in the complexity of calendar queue much greater than $O(1)$.

A sensitivity study to the optimality of bucket width is studied in Figure 3. It is clear that an attempt to achieve a near optimal range for $W_f$ is not too hard. The reason is that, although $W_{fopt}$ is not employed, the time to process a hold operation is still in an acceptable range of complexity for an order-of-magnitude wide range of $W_f$ around $W_{fopt}$.

Figure 4 plots the average hold time versus $W_f$ for a large $N$ = 30,000 and five different distributions of $\tau_i$. Each distribution gives the average hold time with a parabola-alike curve and reflects a different value of $W_{fopt}$. However, the obtained results from all the distributions give a wide near-optimal range of $W_f$, which always includes the value of $W_f = 2$. This finding is consistent with all other experiments not reported herein. Thus, the results suggest a practical approximation for $W_f = 2$, by which the time complexity would increase by less than 15% (based on our experiences so far) from its minimum.

Figure 5 compares the performance of calendar queue with $W_{fopt}$ and another three alternative implementations of calendar in the literature, namely,

(i) the conventional calendar queue with its fixed value of bucket width computed from the average inter-event time on the head of queue [5],

(ii) the dynamic calendar queue with its bucket width dynamically computed from the average inter-event time on the part of queue with the highest event density [7] and

(iii) SNOOPY algorithm whose implementation is based on (ii) with an addition adaptive mechanism to trace for a near-optimal bucket width [8].

Figure 5 suggests that, if one could set $W_f$ to its optimum, then we can improve the speed of existing implementations (i) - (iii) by making the optimal calendar queue algorithm upto twice as fast.

[1]Since $\tau_b$ and $\tau_l$ are not known a priori, replacing both of them by a single parameter, $W_f = \sqrt{\frac{2\tau_b}{\tau_l}}$, reduces the cardinality of solution space from 2 to 1 dimension when searching for an optimal bucket width.

CP08

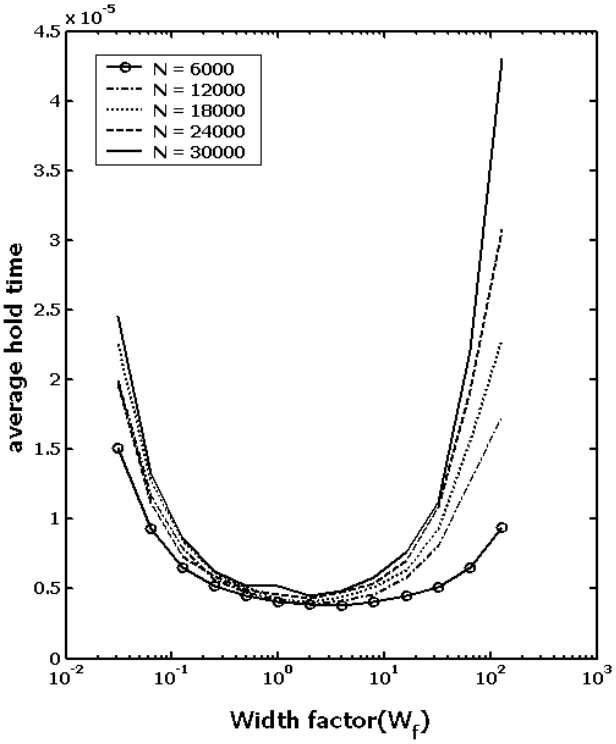


Figure 3: average time per hold operation.

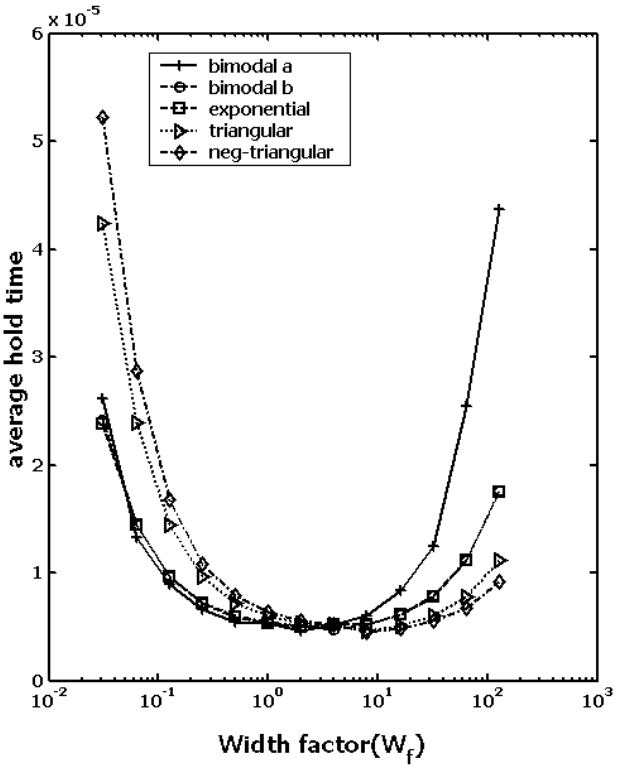


Figure 4: average time per hold operation.

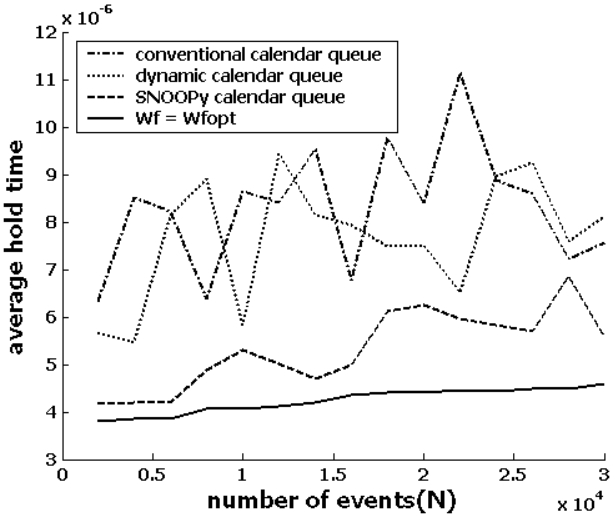


Figure 5: comparison of existing algorithm and $W_{fopt}$ algorithm.

## 6 Conclusion

Because the simulation time of DES is mostly spent on the event management operations, an efficient data structure can explicitly speed up the time needed to simulate a huge system. For this reason, the calendar queue, and other calendar-based algorithms, are the popular data structure adopted for DES implementation for their $O(1)$ time complexity.

Our experiments show that the performance of existing calendar-based algorithms may vary and can much deteriorate when the event incremental time ($\tau_i$) has a highly skewed probability distribution. However, it is here found that the calendar queue with its width factor $W_f$ set to its optimum $W_{fopt}$ can speed up the running time up to twice as fast as other existing calendar-based algorithms.

Although the selection of $W_{fopt}$ can result in the optimal speed of calendar, it is nontrivial how one may a priori set this value. This is because $W_{fopt}$ is dependent on many unknown variables (e.g. distribution of $\tau_i$). Incorporating an adaptive mechanism that can change $W_{fopt}$ dynamically and optimally in the calendar is here believed a rewarding research. However, in the meantime the near-optimum value of $W_f$, $W_f = 2$, that can be commonly use for many simulation scenarios can be adopted to achieve a fair performance of calendar queue.

## References

[1] J.H. Blackstone, C.L. Hogg, and D.T. Phillips, "A two-list synchronization procedure for discrete event simulation," *CACM*, Vol. 24, No.12, pp. 625-629, Dec. 1981.

[2] D. Davey, and J. Vaucher, "Self-optimizing partitioned sequencing sets for discrete event simulation" *infor*, Vol. 18, No. 1, pp. 41-61, Dec. 1981.

[3] W.R. Franta, and K. Maly, "An efficient data structure for the simulation event set," *CACM*, Vol. 20, No. 8, pp. 596-602, Aug. 1977.

[4] W.R. Franta, and K. Maly, "A comparison of heaps and the TL structure for the simulation event set," *CACM*, Vol. 21, No. 10, pp. 873-875, Aug. 1977.

[5] R. Brown, "Calendar Queues: A Fast 0(1) Priority Queue Implementation for the Simulation Event Set Problem," *CACM*, Vol. 31, No. 10, pp. 1220-1227, Oct. 1988.

[6] D.W. Jones, "An Empirical Comparison of Priority-queue and Event-set Implementations," *CACM*, Vol. 29, No. 4, pp. 300-311, Oct. 1986.

[7] S. Oh, and J. Ahn, "Dynamic Calendar Queue" *Inproceeding of the 32nd Annual Simulation Symposium*, 1999

[8] K.H. Tan, and L.J. Thng, "SNOOPY CALENDAR QUEUE," *Inproceeding of the 2000 Winter Simulation Conference*

[9] K.B. Erickson, R.E. Ladner, and A. LaMarca, "Optimizing Static Calendar Queue," Annual IEEE Symposium on Foundations of Computer Science, Vol. 35, pp. 732-743, 1994.

[10] D.E. Knuth, "The Art of Computer Programming. Vol.1 3rd edition., Fundamental Algorithms," Addison-Wesley, Reading, Mass., 1997. chapter 2.