P-Tree Structures and Event Horizon: Efficient Event-Set Implementations

Katerina Asdre and Stavros D. Nikolopoulos
Department of Computer Science, University of Ioannina, GR-45110 Ioannina, Greece
E-mail: {katerina, stavros}@cs.uoi.gr
Received February 22, 2005; revised September 2, 2005.

Abstract This paper describes efficient data structures, namely the Indexed P-tree, Block P-tree, and Indexed-Block P-tree (or IP-tree, BP-tree, and IBP-tree, respectively, for short), for maintaining future events in a general purpose discrete event simulation system, and studies the performance of their event set algorithms under the event horizon principle. For comparison reasons, some well-known event set algorithms have been selected and studied, that is, the Dynamic-heap and the P-tree algorithms. To gain insight into the performance of the proposed event set algorithms and comparisons with the other selected algorithms, they are tested under a wide variety of conditions in an experimental way. The time needed for the execution of the Hold operation is taken as the measure for estimating the average time complexity of the algorithms. The experimental results show that the BP-tree algorithm and the IBP-tree algorithm behave very well with the event set of all the sizes and their performance is almost independent of the stochastic distributions.

Keywords discrete-event simulation, event set algorithms, hold model, event horizon, data structures, heap, P-tree, P-tree structures

1 Introduction

In a discrete event simulation system an event (or future event) is a collection of actions that are scheduled to be executed in a specific simulation time called event time. In such a system events are kept in objects known as event notices and maintained in a data structure known as event set. An event notice is represented by a record with two fields, t and a, where t is the scheduled time for its occurrence, and a is the activity which is scheduled in time t[1–3].

In a discrete-event simulation system based on the next-event time-advance approach, the next-event time-advance mechanism is responsible for the simulation clock. It initializes the simulation clock, and then determines the event time of future events. The simulation clock is then advanced to the event time of the earliest future event with the minimum event time known as next event and the system state is updated to count the occurrence of this event. When the next event occurs, it is removed from the event set and the simulation clock is advanced to the next-event time. The processing of this event may lead to the generation and scheduling of additional (new) future events. A new event is scheduled when its event time t becomes known. Then, an event notice is created and inserted into the event set in such a way that it is ensured that this event will occur at the scheduled time t. This type of simulation approach is referred to as discrete event-driven simulation.

The responsibility for the execution of these operations in a discrete event-driven simulation is due to an algorithm known as an event set algorithm (or event scheduling algorithm); that is, it

- advances the simulation clock to the next-event time.
- it is obvious that being able to repeatedly select
- the event notice from the event set that has the minimum event time is essential. If all the event notices in this event set are known in advance, and their event time remains unchanged, then the problems of determining the next event and updating the simulation clock are easily solved by sorting the event notices and retrieving them in order. In the simulation process discussed above, however, it is often necessary to insert new event notices into the event set as other events are being processed. This leads to the following set of priority queue operations:
- insert a new event notice into the event set (in a
- proper position according to its event time),
- find the event notice with the minimum event time, and
- remove the event notice with the minimum event
time from the event set.

The above priority queue operations are the most frequent operations required by a discrete event simulation system and they are involved in any event scheduling algorithms. Thus, it is clear that the main factor that affects the efficiency of an event scheduling algorithm is the structure of the event set.

The most important requirements of an event scheduling algorithm are operation speed and storage economy. Many researchers have extensively studied this field and presented both analytical and empirical results concerning the time and space performance of many event scheduling algorithms. They use different data structures for the simulation of the event sets. They are linear lists, special kinds of trees, time-indexed lists, two-level structures and many others. Moreover, they use different techniques for the operations performed by
the scheduling algorithms; (see [4–13] for an exposition of the main results).

The data structures used for the simulation of the event set can generally be classified under three types: that is, lists, tree structures and multi-lists. Lists are structures that are based on the simple linear list. They include doubly linked lists, indexed lists[14], SPEEDES Queue which is based on the event horizon technique[15–17] and many others. Trees are structures that are based on the simple binary tree, and include binary heaps[4–6,9,14], skip lists[14], and priority trees[7,14,18] which are studied here as well. Finally, multi-lists are structures that are the result of a combination of several types of lists. This is done in order to combine the merits of two structures that may not perform well when implemented separately. Such structures are the calendar queue and the SNOOPY calendar queue[13].

This paper describes efficient data structures, namely the Indexed P-tree, Block P-tree, and Indexed-Block P-tree (or IP-tree, BP-tree, and IBP-tree, respectively, for short), for the simulation event set. All the structures combine the advantages of both the P-tree and the static representation of the list. The combination of the P-tree and the list provides efficient data structures for the simulation event set in the case where the event horizon technique is applied. The main feature of each of our event set algorithms is the efficiency of the merging process in the event horizon technique; that is, the process of sorting the event notices of the secondary queue and inserting them back into the event set. We point out that, in the horizon technique the most time consuming operation performed by the event set algorithm is the merging process of the secondary queue back into the main event set.

To gain insight into the performance of the IP-tree, the BP-tree and the IBP-tree, and allow comparisons with other selected algorithms (i.e., Dynamic-heap and P-tree), they are coded and tested under a wide variety of conditions in an experimental way. The objective was to estimate the average complexity of each algorithm. For this purpose, we used a revised definition of complexity, for a given configuration of event set and a given distribution providing the scheduled time, to estimate the time expected to be needed for the execution of the Hold procedure (or Hold model).

Two main parameters affecting the execution time of the above operations are (i) the schedule time T, and (ii) the size N of the event set. The parameter T, which is given by a stochastic distribution, determines how long an event will remain in the event set. Six stochastic distributions are especially chosen. They are not only representative of typical simulation problems but also capable of showing the advantages and limitations of each algorithm. All the chosen distributions have mean 1 (see Table 1) and fall into three categories:

(A) unimodal continuous distributions,
(B) bimodal continuous distributions, and
(C) discrete distributions.

The parameter N defines the notion of the small and large event sets. In other words, it determines the size of the event set, that is, the number of event notices in the set at any time. Tests were performed with values of N from 64 (small event set) to 262,144 (large event set). This range is representative of actual simulations and the behaviour of the algorithms for N > 262144 can be extrapolated from the results.

The results of this work show that the IP-tree algorithm combines time performance, storage economy and simplicity of coding. The BP-tree and the IBP-tree algorithms outperform the IP-tree algorithm, and the BP-tree algorithm has a slightly better performance than the IBP-tree algorithm.

The paper is organized as follows. Section 2 presents the main features of the Hold model and the event horizon technique. Section 3 describes the P-tree structure on which our approach is based. The IP-tree, the BP-tree and the IBP-tree structures are described in Sections 4, 5 and 6, respectively. An experimental evaluation of the algorithms is presented in Section 7, where we also compare the performance of the algorithms. Finally, Section 8 concludes the paper with a summary of our results.

2 Hold and Event Horizon

As already mentioned, the two basic operations performed on the event set by an event set algorithm are (i) insertion of a new event notice into event set, and (ii) determination and deletion of the notice of the next event. A standard metric for comparison of the performance of an event set algorithm is the time required for a Hold operation, which combines both insertion and deletion operations[4,3,5,9]. Under the Hold model, event notices are repeatedly deleted and then re-inserted with a randomly reduced priority; this sequence of operations is known as a Hold operation. The Hold operation works as follows:

1) determine and remove the event notice with the minimum event time $T_{\text{min}}$, the current notice, from the event set;
2) increase the event time value of the current notice by T, where T is a random variate distributed according to some distribution $F(t)$, and
3) re-insert the new notice back into the event set; it now has $T = T_{\text{min}} + T$ event time.

The Hold model has two parameters: $N$, the number of notices in the event set, and $F$, the distribution used to determine the time an inserted event will occur. Thus, the model allows the average combined time for insertion and deletion to be measured as a function of the size of the event set and the stochastic distribution.

The event horizon is a fundamental concept that applies to both parallel and sequential discrete event simulations[15–17,19]. Using event horizon one can improve the performance of several event sets; that is, the priority queue data structures such as linked lists and various binary trees.