INVESTIGATION OF EVENT SET ALGORITHMS

Gábor Lencse
Department of Telecommunications
Technical University of Budapest
Irinyi ut 42
H-1117 Budapest, Hungary
E-mail: Gabor.Lencse@hit.bme.hu

ABSTRACT

Seven data structures are examined by simulation and compared to find out which one should be used to implement the Future Event Set of an event driven discrete event simulator. The number of key comparisons and pointer references as well as the CPU time are measured. The effect of the different CPU architectures is examined. Various parameters of the model are changed to determine the characteristics of the studied event set algorithms.
We conclude that both the processor type and the distribution of the delay of the new events influence which data structure produces the best time properties. Though balanced trees proved to be quite good, heap and especially skip list was found to be significantly better on modern processor architectures.

INTRODUCTION

Discrete event simulation is a powerful method in the performance analysis of networks, multiprocessor systems, digital circuits and other computer systems. In the case of the event-driven approach, the events (the state changes of the system) are stored in the Future Event Set (FES). The data structure and the algorithms used for storing the elements of the FES influence the speed of the simulator to a great extent.
The most common data structures have already been studied and their general performance characteristics have been determined. The cost of different operations on the given data structure (e. g. the number of key comparisons necessary) is mostly expressed as the function of the elements currently stored in the FES. Most investigations on data structures and algorithms assume that the keys are random (with uniform distribution), the number of search operations is much higher than the number of insertions or deletions and the element to be searched or deleted is also randomly chosen (Aho et al. 1975; Knuth 1973; Wirth 1976). Nevertheless, in the case of the FES, the keys are not uniformly distributed, but show a rising tendency; in the vast majority of cases, the first element is deleted and a new one is inserted. Sometimes randomly chosen elements are deleted, but no other search operations are used. Reeves (Reeves 1984) examined variants of lists and heaps, but the different kinds of tree structures and the skip list (Pugh 1990) require further investigation.
This paper deals with the experimental investigation of the performance of different data structures as they are used for storing the elements of the Future Event Set of an event-driven discrete event simulator. The examined data structures are: ordered single-linked list, binary tree, AVL-tree, B-tree, 2-3-tree, heap and skip list (Aho et al. 1975; Knuth 1973; Pugh 1990; Wirth 1976).
This topic was identified as being of importance to implementing event-driven discrete event simulators.

THE MODEL USED FOR INVESTIGATION

To construct a model suitable for our study, we must start from the algorithm of the event-driven discrete event simulation. Let us see this algorithm first:

```
initialize, insert certain events into the FES;
repeat
  remove the first event from the FES;
  NOW:=the time of the event removed from the FES;
  process the event, during this insert some event(s) into the FES
        if necessary;
until (FES is empty) or (NOW>limit) or (for other reason, we must
        stop)
```

Here, 'first event' means the event with the smallest time stamp.
Our model for the FES must be an abstract data structure with the insert and first operations that insert a new event into the FES and removes the earliest one from the FES, respectively.
In addition to these operations, a third one is required: sometimes it is necessary to delete an arbitrary event from the FES. (This is used for example for handling time-out.)
As we do not take measurements on an existing simulator that is simulating a given system, but examine how the different data structures behave if we use them to implement the FES, we must make the following assumptions:

1. during its processing, an event may generate one or more new events
2. the time interval between the time stamp of the current event and that of a newly generated event follows a certain probability distribution
3. the choice of the event to be deleted from the FES is random with uniform distribution

To judge the behaviour of the different event set algorithms, the following characteristics are measured:

- number of key comparisons necessary

- number of pointer references necessary
- CPU time used

Of course, the last one depends on the architecture of the computer used for performing the simulation.

THE PARAMETERS OF THE MODEL

The State of the FES

According to the first assumption, we must take into consideration whether an event generates exactly one event or more. We must examine both the steady state (number of events in the FES is constant) and the transient one (number of events in the FES is increasing). In the latter case, let all events generate *two* new ones.

The Number of Events in the FES

The aim of this study is to determine and compare especially the *asymptotic* behaviour of the different event set algorithms, so a sequence of measurements is necessary using different values for n that denotes the number of events stored in the FES. Let n take the values 10, 21, 46, 100, 215, 464, 1000, 2154, 4641, 10000. The numbers were chosen so that the difference between the logarithm of the successive values be approximately the same constant.

The Number of Simulation Steps

This way, $2n$ insertions and n first operations are performed in the transient state. Let m denote the number of steps (first and insert operations) in the steady state. This number must be large enough to ensure the accuracy of the measurements.

The distribution of the delay

The probability distribution of the time interval between the current event and the newly generated one (mentioned in the second assumption) was represented by the following distributions:

- uniform distribution in the [0, 1] interval
- exponential distribution with $\lambda=1$ intensity
- normal distribution: Expected value=1, Standard deviation=0.3
- normal distribution: Expected value=1, Standard deviation=0.1

(where the normal distributions are truncated to $t \geq 0$)

The number of deletions

Let us use deletions (see assumption 3) as a given percentage of all the removal of events from the FES. The values used were 0%, 5%, 10%, 20%, 50%. Let us call them *delete percentage*s. (The 0% means that all removals are first operations and the 50% means that the number of random deletions equals with the number of first operations.)

The type of the processor

As it was mentioned before, the time of the operations depends on computer architecture. For this reason, three different processors were used. (DEC ALPHA, i80846-DX, i80486-DLC. For more details see the next section.) On a processor without hardware floating point support, the number of key comparisons dominates. However when the hardware support speeds up the floating point operations, the number of pointer references and other instructions necessary for the maintenance of the data structure may influence the time of the event set operations more than in the first case.

Data structures studied

The names of the seven data structures used for storing the elements of the FES are listed here:

- ordered single-linked list
- binary tree
- AVL-tree
- B-tree
- 2-3-tree
- heap
- skip list

Six out of the seven parameters described in the previous sub-sections (with the exception of m, the number of simulation steps in the steady state) are orthogonal, which means that measurements must be performed for all possible combinations of their values. This way, we get a huge quantity of results.

INVESTIGATION BY SIMULATION

To measure the before mentioned performance characteristics of the seven data structures, several series of very simple event driven discrete event simulations were performed. Events were inserted into the FES and they were removed by first or delete operations. The algorithm of the discrete event simulation shown at the beginning of the paper was modified as follows:

The removal of the element is done by delete operation with the probability of *delete percentage* and by first with (1-*delete percentage*) probability. "NOW" is updated only in the case of first. After every removal *two* new events are inserted with the time stamp of NOW+random(actual_distribution) in transient state and only *one* in steady state.

The parameters were changed across the series of simulation runs and 100 experiments were executed with all the parameter combinations used. Average and standard deviation of the number of key comparisons and pointer references as well as that of the CPU time used were calculated. The following formula of empirical deviation was used:

$$\sigma = \sqrt{\frac{\sum(x^2) - N \cdot \bar{x}^2}{N-1}}$$

Where N=100 is the number of experiments and $\bar{x}$ denotes the average value of x.

The rest of this section describes further details necessary to reproduce the simulation.

All the programs were written in the C++ language taking advantage of both the incomplete Boolean evaluation and operator overloading. The standard 'double' type of C++ was used to store the keys of the events.

The simulation was run on three different computers.

One of them was a DEC ALPHA workstation (150MHz, Evolution 4 processor, 512 KB cache, 64MB RAM, DEC 2000, model 300, DEC OSF/1 V3.0 operating system). The programs were compiled with gcc (v2.6) using all optimizations (-O3 flag).

The second one was an Intel 80486 DX2 (66MHz, 256KB cache, 20MB RAM, DOS 6.2). The Borland C++ 3.1 compiler was used and the compilation was optimized for the speed of the code. The memory model was "Compact".

The third one was an Intel compatible (Texas) 80486 DLC (40MHz, 128KB cache, 4MB RAM, no mathematical co-processor, DOS 6.2). The compiler and its options were the same as in the second case except that the floating point emulation was turned on as this third processor (unlike the first two) does not have hardware floating point support.

Unfortunately, the clocks of these machines have the resolution of 1ms, 55ms, 55ms, respectively, so it was necessary to repeat the experiments within one time-measuring interval in the case of the transient state for the small values of n (number of events loaded into the FES) to obtain acceptable accuracy.

The number of steps in steady state (earlier denoted by m) was chosen to be large enough (10,000, 10,000 and 1000 for the three processors, respectively) to ensure the satisfactory precision of the time-measuring.

The random number generator was taken from (Jain 1991) to ensure the quality of the random numbers used for the simulation.

The algorithms of the data structures were taken from the sources below:

AVL-tree	(Wirth 1976)
B-tree	(Wirth 1976)
heap	(Gonnet 1984)
skip list[1]	(Pugh 1990)

As the 2-3-tree is the special case of the B-tree, the code was written on the basis of the B-tree applying a number of simplifications. Binary tree and list were found to be trivial and implemented by the author.

Heap is usually implemented in the way that its elements are stored in an array. However, this slows down its operations when an element has to be moved. For this reason, only pointers to the elements (the events) were stored in the array.

The parameter of the B-tree is 20, that is, minimum 20, maximum 40 keys can be stored in a node. And again, pointers to event entries were stored in the array of the node.

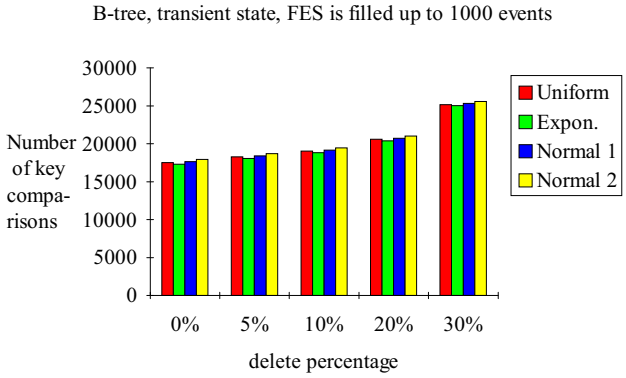
DISCUSSION OF THE RESULTS[2]

Distribution Dependence

The quantity of the results is so large that some reductions seem to be necessary. Let us examine first whether the performance of the different event set algorithms depends on the distribution used. (If not, then we can reduce the data to be examined by 75%.)

The number of key comparisons is suitable for this purpose as it is not architecture-dependent so it is enough to examine a data structure only one processor.

The AVL-tree, the B-tree and the 2-3-tree show very similar behaviour: their behaviour does not depend on the distribution used, and the number of key comparisons is rising as a function of delete percentage. The graph below shows the results for the B-tree only as the figure would be very similar for the other two trees.



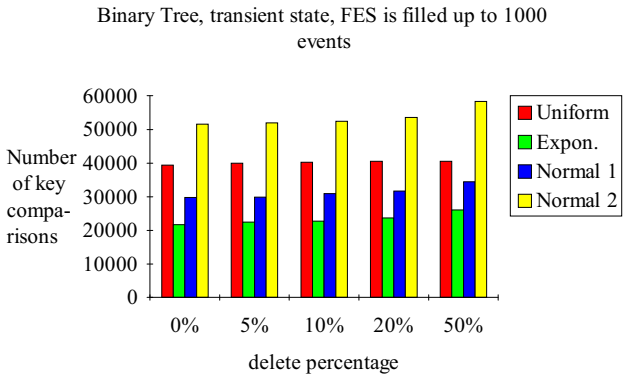
This distribution independence is not surprising, because all the three trees are well balanced, and the effect of delete percentage can be explained by the fact that the first operation does not require key comparisons while the delete does.

The binary tree (see on the next page) shows distribution dependence. This data structure produces its best results in the case of the exponential distribution. If we compare the results of the two normal distributions, the one with smaller deviation produces worse results than the other one. (An intuitive explanation is that the first operation consumes the binary tree from its left side and insertion builds it on the right side. This is even more true if we use normal distribution with small deviation as it is nearer to the deterministic scheduling of the

[1]The source code was downloaded via anonymous ftp from ftp.cs.umd.edu/pub/skipLists

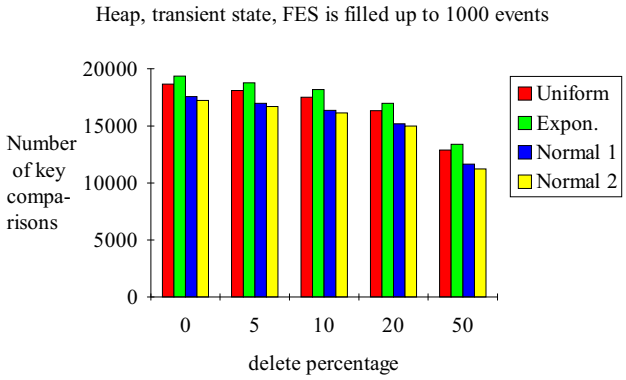
[2]If you are interested to get all the results, please send an e-mail to the author (Gabor.Lencse@hit.bme.hu).

new events using constant delay. The exponential distribution produces values near zero with higher probability so it also builds the left side of the tree.).



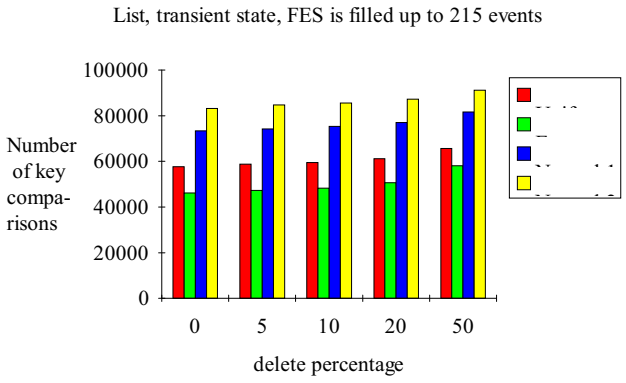
Another important observation is that the delete percentage has much less influence on the number of key comparisons here than we saw earlier.

The behaviour of the heap is shown in the graph below:



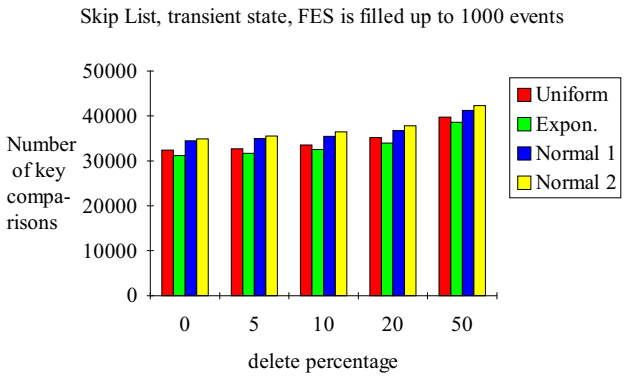
It has a slight distribution dependence, but it is small so we can neglect it. Unlike all the others, heap requires less key comparisons as the delete percentage increases.

The simulation with the list was so slow that it could not be run by n=1000, so now we shall use the results for n=215. (These values should not be compared to the result of the other data structures with n=1000.)



Its distribution dependence is similar to that of the binary tree, and the number of key comparisons increases slightly in the function of the delete percentage as well.

Finally, let us look at the skip list:



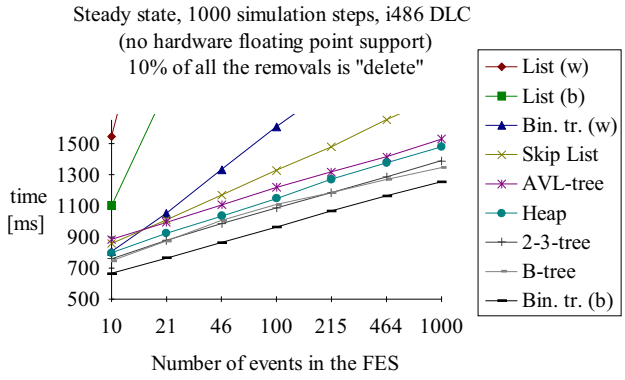
The distribution dependence is negligible but the number of key comparisons shows a certain increase as the delete percentage grows.

Comparison of the performance of the data structures

As discussed above, both the distribution and the delete percentage may influence the performance of the data structures examined. This can make the comparison very hard unless we find a reasonable compromise. Let us use a 10% delete percentage - that seems to be quite realistic. The solution to the problem of distribution dependence is to use two graphs of the data structures the performance of which is highly influenced by the distribution used, the binary tree and the list. One of them shows their best result and the other the worst one, denoted by "b" and "w", respectively. Uniform distribution is used for all the other 5 data structures.

The following 3 figures show the time characteristics of the data structures.

First, we examine the case where the processor does not support the floating point operations:

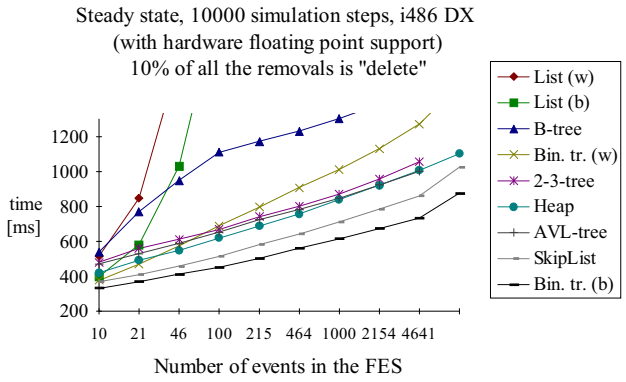


(Note: the horizontal axis has a logarithmic scale.)

The well-balanced data structures (2-3-tree, B-tree and AVL-tree) performed well because they efficiently minimize the number of key comparisons that dominate the time on this

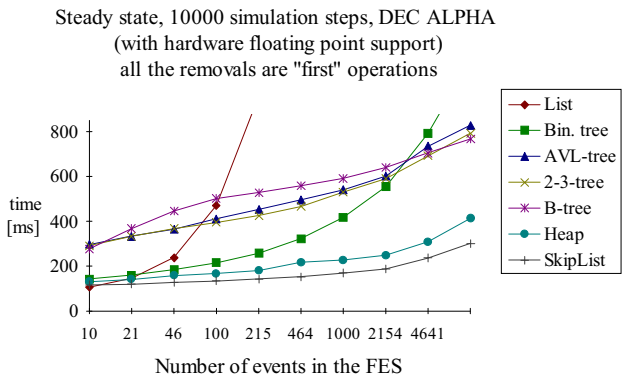
processor. Heap is fairly good here because it requires a relatively small number of key comparisons as we saw earlier.
Binary tree required the least time when it showed its "best case" performance. However, since we cannot guarantee that exponential distribution will be used in the vast majority of the simulations, we must (also) consider the worse performance results of the binary tree.
Skip list was definitely worse than the balanced trees.
Of course, list was inferior to all of them.

Second, we deal with the results produced on the other Intel processor - the one with built-in mathematical co-processor:



As we can see, AVL-tree, 2-3-tree and heap retained their good position, B-tree is significantly worse. We can say the same about the binary tree and the list as we did earlier at the previous processor. Linked list is the worst only above a certain value of *n*, the number of events in the FES. On this platform, skip list has overtaken the well-balanced tree structures and the heap.

Finally, let us have a look at the results on the ALPHA processor. Here we use 0% delete operations and apply uniform distribution to all the data structures:



The graph shows that the heap and especially the skip list are the best choice on this architecture.

CONCLUSION

Seven data structures were examined by simulation and compared to find out which one should be used to implement the Future Event Set of an event driven discrete event simulator.
We conclude that both the processor type and the distribution of the delay of the new events influence which data structure produces the best time characteristics.
By examining the number of key comparisons, we found that the balanced trees and heap produce the lowest values from among all the data structures. For this reason if we use a processor where the CPU time of the event set operations is dominated by the key comparisons (that is, floating point operations are not supported by hardware), balanced trees and heap are the best choice. Binary tree produced good performance characteristics in the case of exponential distribution, though in the case of other distributions binary tree shows worse results.
On the other hand, skip list and also heap have produced much better time characteristics than the balanced trees when advanced hardware floating point processing was used. (ALPHA processor) In this case, it is worth using skip list or heap rather than balanced trees, especially because their algorithms are even simpler than that of the balanced trees.

REFERENCES

Aho, A. V.; J. E. Hopcroft; and J. D. Ullman. 1975. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass.

Gonnet, G. H. 1984. *Handbook of Algorithms and Data Structures.* Addison-Wesley.

Jain, R. 1991. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 441-444, 455.

Knuth, D. E. 1973. *The Art of Computer Programming. Vol. 3. (Sorting and Searching)* Addison-Wesley

Pugh, V. 1990. "Skip Lists: A Probabilistic Alternative to Balanced Trees" *Commun. of the ACM* 33, no. 6 (June) 668-676

Reeves, C. M. 1984. "Complexity Analyses of Event Set Algorithms" *The computer journal* 27. no. 1, 72-79

Wirth, N. 1976. *Algorithms + Data Structures = Programs.* Prentice Hall

BIOGRAPHY

Gábor Lencse was born in Gyôr, Hungary, in 1970. He received his M.S. in electrical engineering and information science from the Technical University of Budapest in 1994. He is currently pursuing his Ph. D. at he same university. The area of his research is computer architectures and parallel processing. He is also interested in PVM.

List, transient state, FES is filled up to 215 events