Comparison of Priority Queue algorithms for Hierarchical Scheduling Framework

Mikael Åsberg mag04002@student.mdh.se

August 28, 2008

The Time Event Queue (TEQ) is a datastructure that is part of the implementation of a Hierarhcial Scheduling Framework (HSF) [1]. It's main function is to store future task scheduling events (represented as absolute time values) in a sorted manner, thus, it implements a Priority Queue (PQ). A PQ is a queue with elements that are sorted by their priority [3]. The two main operations on a PQ is insert and delete-min. The first mentioned operation will insert an item based on its priority. The second operation will extract an item with highest (or lowest) priority.
As of now, the PQ structure is implemented as a sorted linked list with a median pointer and where binary search is used for insertions.
The efficiency of the current implementation is not up to standard so there is a requirement that the current implementation should be replaced or optimized. An investigation should be made so that other PQ implementations are considered that are well suited for hard real-time systems.
When choosing a suited PQ for the hard real-time scheduler HSF, considerations such as a low number of queue items in combination with good worst case performance is of importance.
This paper will motivate and choose a group of suited PQ algorithms and present empirical tests of each of these algorithms. Based on these results and our requirements, one algorithm will be chosen to be integrated with the HSF.

Related work

In [2] the authors compare suitable PQ algorithms which support both sequential and parallel access. Comparisons are based on the empirical tests of these algorithms. The tests are performed by measuring the amount of time to remove the highest priority item and insert it (with a new priority) into the queue (hold operation). Hold operations are performed with different queue sizes and with different element priority increment distributions (bias). The distribution increment will affect the new location (in the queue) when inserting items. For simplicity, we are only interested in three of these distributions:

- Biased, FIFO-like distribution where most elements are inserted in the end part of the queue.

- Bimodal, LIFO-like distribution where most elements are inserted in the head part of the queue.

- Exponential, this distribution tends to insert most elements in the middle part of the queue.

The following facts can be obtained from this paper [2]:

- Best algorithm with biased distribution and element size 1-1000 is Splay tree, SPEEDESQ and Henriksens (Linked list with median pointer is fastest when element size is around 100 elements and below).

- Best algorithm with bimodal distribution and element size 1-1000 is Splay tree (though in general it is good no matter which distribution).

- Best algorithm with exponential distribution and element size 1-1000 is Splay tree or Calendar queue (quite even but Splay tree is better when element size is around 100 elements and below).

PQ algorithm	*Most effective (bias)*	*Most effective (# items)*	*Comment*
Binary heap	even	-	Even curve (O(log(n)))
Linked list	Biased	1-100	Good biased curve (1-600)
Skew heap	even	-	Better than Binary heap (O(log(n)))
Splay tree	even (Biased)	-	Outperforms Binary and Skew heap
Calendar queue	even (except Camel)	-	Best performance (flat curve)
Lazy queue	even (except Camel)	-	Near Calendar performance
Henriksens	even (Biased)	1-1000	Near Calendar performance (1-1000)
Skip lists	Exponential	-	Uneven bias distributions
SPEEDESQ	even (Biased)	1-100	Even bias, good performance (1-100)

Table 1: Properties of PQ algorithms

• Best algorithm for biased, bimodal and exponential distribution and element size 1-1000 is Splay tree, Calendar queue and Henriksen. Calendar might be slightly worse than the other two.

The paper [2] also show that Skew heap and Splay tree have very good worst case performance with element sizes ranging up to 1000 and where exponential distribution is used. Binary heap has slightly worse performance but it has a worst case performance of $O(log(n))$ for individual operations (queue insert and remove). Authors conclude that Binary heap is the best choice for hard real-time applications.

The authors conclude that SPEEDESQ and Linked list with median pointer are good choices when elements are less than 50.

The strong algorithms that performed best with the bimodal distribution are Calendar queue, Splay tree and Skew heap, where Splay tree is slightly better than the other two.

The conclusion from [2] is that Linked list with median pointer is the best alternative for biased distribution if we assume that task sets range from 1 to about 100 tasks.
Splay tree is the best choice considering bimodal, exponential and all three distributions together.
Finally, because Binary heap show good adaptation to hard real-time applications, this implementation is a good choice.

Table 1 presents properties of the algorithms analyzed in [2] which are interesting considering the TEQ implementation for HSF.

In [3] the author present and compares different sequential algorithms that are suited for discrete event simulations.

Some interesting facts concering the TEQ is that Binary heap, Leftist tree and Binomial queue have worst case $O(log(n))$ for single operations. A notation is that Binary heap are among the worst implementations when number of elements are less than 20.
The following facts can be obtained from this paper [3]:

• Best algorithm with biased distribution and element size 1-1000 is Splay tree, which is marginally better than Henriksens.

• Best algorithm with bimodal distribution and element size 1-1000 is Splay tree, followed by Skew heap.

• Best algorithm with exponential distribution and element size 1-1000 is Splay tree.

PQ algorithm	*Most effective (Bias)*	*Most effective (# items)*	*Comment*
Linked list	Bimodal	1-10	Best alg. in all bias (1-10)
Binary heap	even	-	Even cons. bias and # items
Leftist trees	even (Bimodal)	-	Worse than Binary heap
Two list	Bimodal	1-200	Among the best impl. (1-200)
Henriksens	Biased	1-100	Worst case is near Binary heap
Binomial queue	even	-	Most complex and effec. impl.
Pagodas	even	-	Near Binomial performance
Skew heap	even	-	Near Binomial performance
Splay tree	even (Biased)	-	Better biased than Henriksens
Pairing heaps	even (Biased)	-	Among best biased distr.

Table 2: Properties of PQ algorithms

• Best algorithm for biased, bimodal and exponential distribution and element size 1-1000 is Splay tree followed by Pagodas and Skew heap.

The conclusion from [3] is that Splay tree is the best choice no matter which element distribution and when element size is below 1000. Considering the best worst case implementations, Leftist tree, Binary heap and Binomial queue are the only implementations that have a $O(log(n))$ bound for single operations. The most effective of these are Binary heap and Binomial queue while Binomial is the best. An important notation is that the code complexity is much higher for Binomial queue than Implicit heap.

Finally, a last comment concerning [3] is that the bimodal distribution in [3] is more bimodal (lower bias value) than the corresponding distribution in [2]. The first mentioned paper has a bias value of 0.13, the second has 0.34 for bimodal distribution. This is why the bimodal distribution for Linked list in [3] is very effective compared with the other paper ([2]). The Linked list in [3] does not use a median pointer which makes the implementation less memory consuming but the biased distribution is thereby ineffective.

Table 2 shows an overview of the algorithms presented in [3].

In [4] the author show that Binary heap is the most effective algorithm for bimodal and exponential distributions and second best after Henriksens in biased distribution when element size is 10.

[5] compares four algorithms for discrete event simulation. Their tests show that the Post-order tree is effective when element size rang to up to 200 items.

In [6], Henriksens algorithm has the best result for biased, bimodal and exponential distribution.

Chosen algorithms

Based on the results from the papers [2] and [3], the Splay tree show great efficiency for all three distributions (biased, bimodal and exponential) when element size range up to 500-1000 elements. This is why we have chosen this algorithm for our own testcases.
The most efficient algorithms that have a bounded worst case time for single operations are Binary heap and Binomial queue [3]. Since the HSF is meant the schedule hard tasks, worst case bounded algorithms are of great interest.
Finally, [2], [3] conclude that Linked list with

median pointer has good performance when element size is below 50 and that it actually is the best algorithm when the size of the queue is around 10. It is not rare that tasks-sets might be around 100 tasks, this is why the Linked list is chosen for our tests.
Our chosen algorithms will be tested together with the original TEQ implementation.

The following algorithms will be tested and compared:

- Splay tree
- Binary heap
- Binomial queue
- Linkes list with median pointer
- TEQ

Splay tree [8] is implemented as top-down. The Binomial queue [8] insert function is not optimized for $O(1)$ amortized performance. The Binary heap [7] implementation stores the binary heap in a one dimensional array. The TEQ [1] implementation is implemented as a double linked list and uses binary search for insertions. Linked list with median pointer is implemented with three pointers (front, middle and back) and insertions can be made from the head, end and middle of the queue.

Results

The time measurements were made on a ABB robotics controller equipped with VxWorks 5.2 operating system and a pentium pro 200 Mhz CPU.
During the measurements, the test-code executed in a task with the highest system priority and all system interrupts were disabled.
The code section that was to be measured included a removal of the highest priority node, updating its value and insertion of the node (with new value) in the queue.
Time measurements were made with a high resolution hardware timer with 12 000 000 ticks/sec. The timer was read in the beginning and end of the code section trough a memory mapped address. Each measurement was subtracted with 149,9 hardware ticks (approximately 12,5 microseconds) in order to remove measurement overhead. This value was obtained by measuring 1000000 timer read, and then dividing this value with 1000000.
Updating a node involves calculating a new value (priority) for that node, the newly calculated value is then added with the previous value of the node. The calculation of a new value is dependant on which priority distribution that is used. Biased, bimodal and exponential distribution have different calculations. The calculation algorithms for these three distributions are taken from [3], where each algorithm is shown with its corresponding bias value.
10000 measurements were made for each queue size, involving an amount of two times the queue size 'warm up' hold operations before each main measurement.
All implementations were optimzed so that no memory allocation operation was performed during the hold operations.

Time measurements of the algorithms corresponds quite well to previous studies [2, 3], with the exception that Binomial queue and Splay tree are inefficient (especially Binomial queue). The only explanation for this behaviour is that our implementation might not be as efficient as in previous papers.
Figure 1-3 show that Binary heap has the lowest max values and that the priority distribution does not affect the measurement values. This algorithm has been favoured as one of the best worst case algorithms for individual operations, and well suited for hard-real time systems. The test results do not contradict these facts.
Figure 4-6 display the minimum values. Linked list with median pointer has the best results, especially with biased distribution.
Finally, a comment on the diagrams in *Figure 7-9*. Linked list with median pointer has best result in biased and exponential distribution no matter the

queue size, and with bimodal distribution when queue size is 1-300. The diagram data can also be viewed in table form *Table 3-5*.

Conclusion

Based on these empirical tests, Binary heap and Linked list with median pointer are interesting candidates. The first mentioned algorithm has slightly worse median performance, but it is more reliable considering that it has better worst case performance. In hard real-time systems, best worst case performance is of interest, while soft real-time systems prefer good median performance. The conclusion is that Binary heap is best suited for the HSF, while Linked list with median pointer belongs in a soft real-time system.

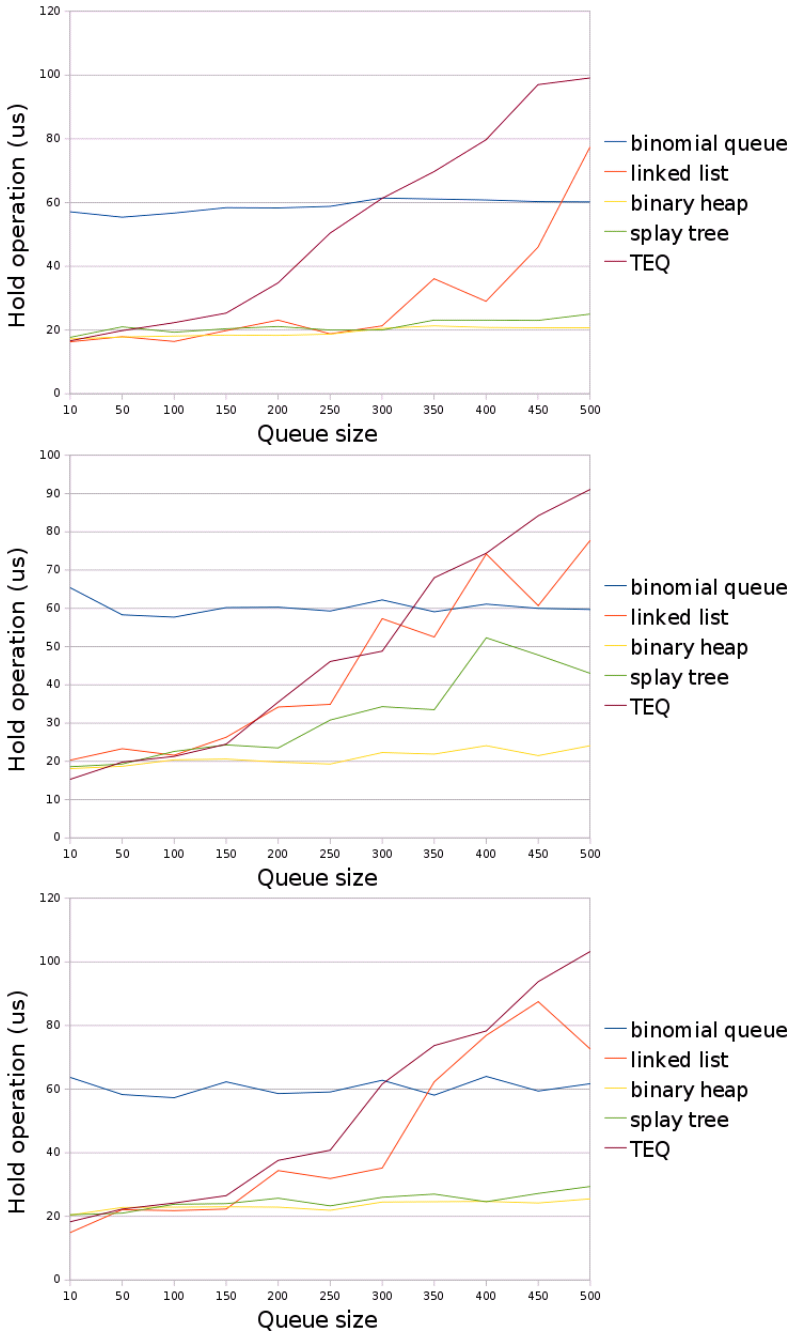

Figure 1-3: Biased, bimodal and exponential distribution with max values

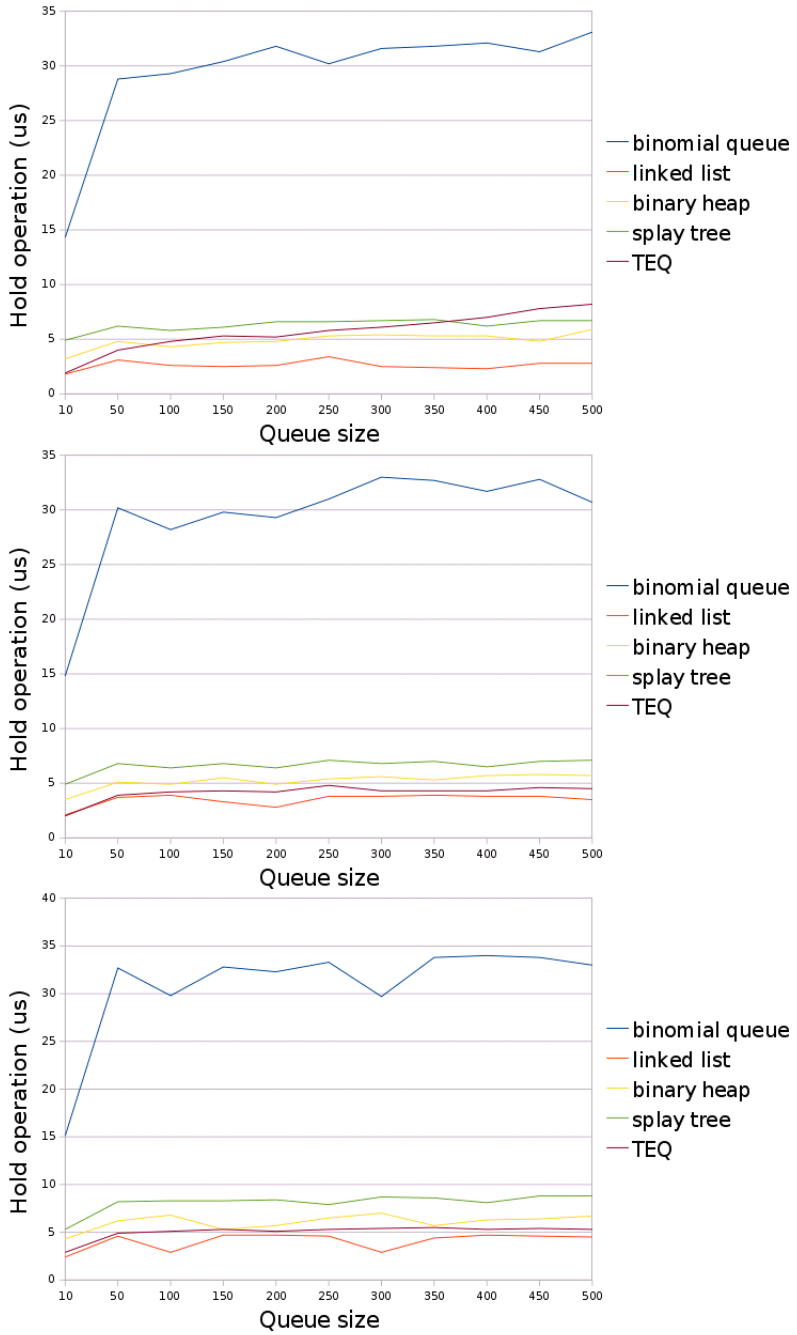

Figure 4-6: Biased, bimodal and exponential distribution with min values

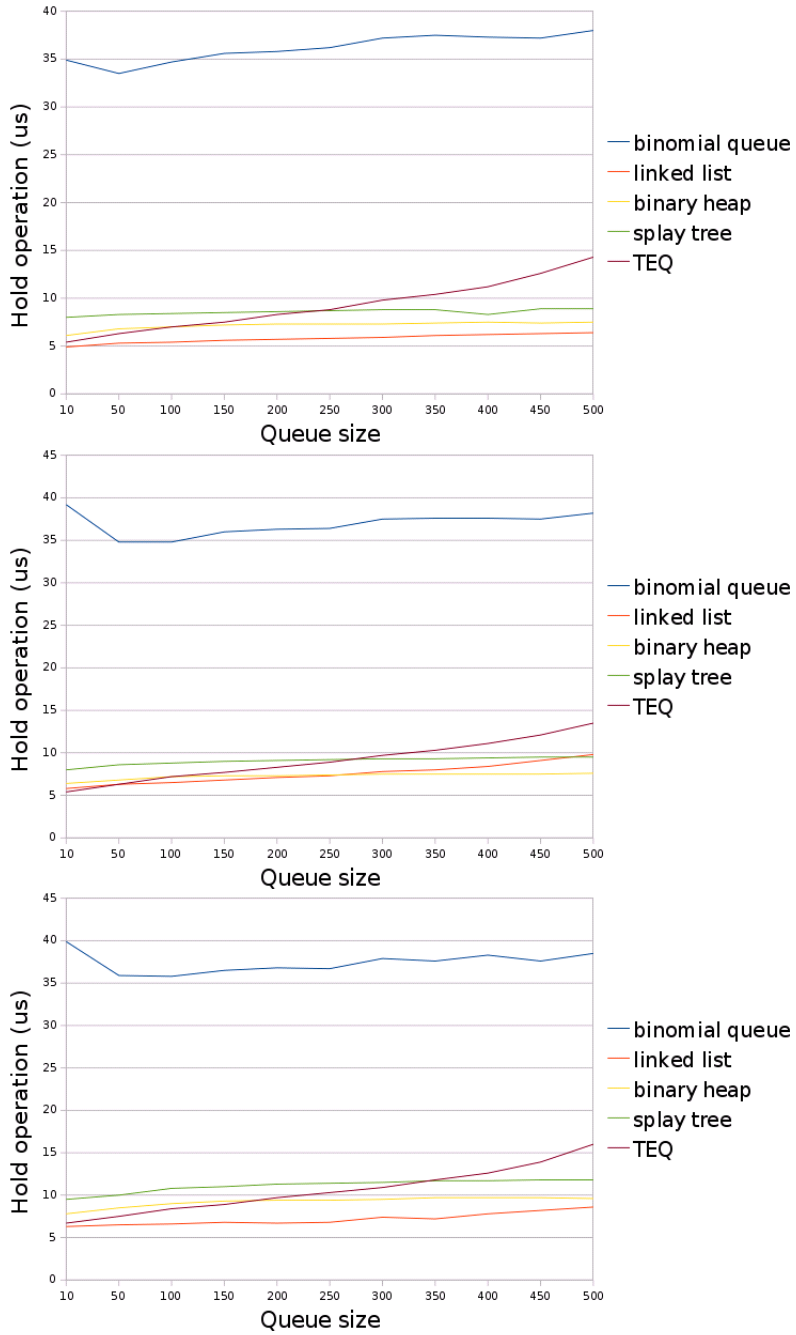

Figure 7-9: Biased, bimodal and exponential distribution with median values

PQ algorithm	$q : 10$	$q : 50$	$q : 100$	$q : 150$	$q : 200$	$q : 250$	$q : 300$	$q : 350$	$q : 400$	$q : 450$	$q : 500$
Linked list	4,4	4,4	4,4	4,7	4,7	4,8	4,8	4,9	5,0	5,2	5,3
Binary heap	6,1	6,8	7,0	7,2	7,3	7,3	7,3	7,4	7,5	7,4	7,5
Splay tree	8,0	8,3	8,4	8,5	8,6	8,7	8,8	8,8	8,8	8,9	8,9
TEQ	5,4	6,3	7,0	7,5	8,3	8,8	9,8	10,4	11,2	12,6	14,3
Binomial queue	34,9	33,5	34,7	35,6	35,8	36,2	37,2	37,5	37,3	37,2	38,0

Table 3: Test results (median) with biased distribution and different queue sizes

PQ algorithm	$q : 10$	$q : 50$	$q : 100$	$q : 150$	$q : 200$	$q : 250$	$q : 300$	$q : 350$	$q : 400$	$q : 450$	$q : 500$
Linked list	4,6	5,0	5,3	5,5	5,9	6,5	6,9	7,5	8,5	9,3	9,8
Binary heap	6,4	6,8	7,2	7,3	7,3	7,4	7,5	7,5	7,5	7,5	7,6
Splay tree	8,0	8,6	8,8	9,0	9,1	9,2	9,3	9,3	9,4	9,5	9,5
TEQ	5,4	6,3	7,2	7,7	8,3	8,9	9,7	10,3	11,1	12,1	13,5
Binomial queue	39,2	34,8	34,8	36,0	36,3	36,4	37,5	37,6	37,6	37,5	38,2

Table 4: Test results (median) with bimodal distribution and different queue sizes

PQ algorithm	$q : 10$	$q : 50$	$q : 100$	$q : 150$	$q : 200$	$q : 250$	$q : 300$	$q : 350$	$q : 400$	$q : 450$	$q : 500$
Linked list	5,8	6,1	6,3	6,4	6,5	6,8	7,4	7,5	8,3	8,9	9,6
Binary heap	7,8	8,5	9,0	9,3	9,4	9,4	9,5	9,7	9,7	9,7	9,6
Splay tree	9,5	10,0	10,8	11,0	11,3	11,4	11,5	11,7	11,7	11,8	11,8
TEQ	6,7	7,5	8,4	8,9	9,7	10,3	10,9	11,8	12,6	13,9	16,0
Binomial queue	39,9	35,9	35,8	36,5	36,8	36,7	37,9	37,6	38,3	37,6	38,5

Table 5: Test results (median) with exponential distribution and different queue sizes

Bibliography

[1] M. Behnam T. Nolte I. Shin M. Åsberg, R. Bril *Towards Hierarchical Scheduling on top of VxWorks* In Proceedings of the 4th International Workshop Operating System Platforms for Embedded Real-Time Applications (OSPERT'08), 2008.

[2] R. Rönngren R. Ayani *A Comparative Study of Parallel and Sequential Priority Queue Algorithms*, ACM Transactions on Modeling and Computer Simulation (Vol. 7, No. 2, p. 157-209): New york, 1997.

[3] D. W. Jones *An empirical comparison of priority-queue and event-set implementations*, Communications of the ACM (Vol. 29, Issue 4, p. 300-311): New york, 1986.

[4] M. Marin *An empirical comparison of priority queue algorithms*, Programming research group, Computing laboratory, University of Oxford.

[5] J. G. Vaucher P. Duval *A comparison of simulation event list algorithms*, Communications of the ACM (Vol. 18, Issue 4, p. 223-230): New york, 1975.

[6] W. M. McCormack R. G. Sargent *Analysis of future event set algorithms for discrete event simulation*, Communications of the ACM (Vol. 24, Issue 12, p. 801-812): New york, 1981.

[7] S. Saunders *Binary Heap Implementation* <http://www.cosc.canterbury.ac.nz/tad.takaoka/alg/heaps/bheap.c>, 15 Aug 2008.

[8] M. A. Weiss *Source Code for Data Structures and Algorithm Analysis in C* <http://www.cs.fiu.edu/ weiss/dsaa_c2e/files.html>, 15 Aug 2008.