PRACTICAL PERFORMANCE ANALYSIS OF REAL-TIME MULTIPROCESSOR SCHEDULING ALGORITHMS

H. Alhussian¹*, S. J. Abdulkadir¹, N. Zakaria¹, A. Patel² and A. Alzahrani³

¹Universiti Technologi Petronas, 32610 Bandar Seri-Iskandar, Perak, Malaysia
²Vieira Computer Networks and Security Laboratory (LARCES), State University of Ceará (UECE), Fortaleza, Ceará, Brazil
³King Saud University, Riyadh, Saudi Arabia

Published online: 01 February 2018

ABSTRACT

This paper presents a practical performance analysing of two real-time multiprocessor scheduling algorithms, namely, Largest Remaining Execution-Time and Local Time Domain (LRE-TL) and Unfair Semi-Greedy (USG). The analysis is intended to reflect the behind-the-scene time overhead incurred by optimal real-time algorithms such as LRE-TL. The overhead is known to be capable of dismissing the actual optimality of such algorithms in practical applications. Here, the time overhead is measured in terms of the number of scheduler invocations and the time required by the scheduling event handlers. In the implementation of the proposed analysis method, the CPU profiler of Oracle Java™ VisualVM was used to monitor the executions of LRE-TL and USG. The profiler measured the number of invocations of the scheduling event handlers for each algorithm and the total time required for all the invocations. The results revealed that USG outperformed LRE-TL on both measures, indicating that optimal algorithms may prove to be non-optimal in practical applications.

Keywords: Real-time; Multiprocessor; Scheduling;

Author Correspondence, e-mail: seddig.alhussian@utp.edu.my
doi: http://dx.doi.org/10.4314/jfas.v10i2s.6
1. INTRODUCTION

The correctness of real-time systems is not only determined by the logical results that it produces, but also the physical time at which the results are produced [1-8]. In order to fulfil the timing constraints of a real-time taskset (i.e. deadlines) in a real-time multiprocessor system, an optimal scheduling algorithm must be used. A scheduling algorithm is said to be optimal if it successfully schedules all the tasks of the system without missing any deadline provided that a feasible schedule exists for the tasks [3, 7, 9-11].

Optimal real-time multiprocessor scheduling algorithms consistently attain the highest processor utilisation, which is equal to the number of processors in the system. However, optimality is always achieved at the expense of scheduling overheads in terms of task pre-emptions and migrations, with significant impact on the practicality of the algorithm. This is because the optimality is mostly achieved by adherence to the fairness rule. Accordingly, tasks are forced to progress through their executions in time quanta or at the end of each time slice in a fluid schedule model of the deadlines of all the tasks in the system. The pre-emptions and migrations engender additional overheads, which must be added to the worst-case execution requirements of the tasks. However, theoretical studies of such algorithms ignore the overheads and the results of practical implementations thus fall short of the predictions.

To support this claim, we developed a method for the practical performance analysis of two real-time multiprocessor scheduling algorithms, namely, Local Remaining Execution-Time and Local Time Domain (LRE-TL) as an example of an optimal scheduling algorithm [12], and Unfair Semi-Greedy (USG), which is a non-optimal algorithm with a performance comparable to that of optimal algorithms [13].

The rest of this paper is organised as follows. Section 2 outlines the challenges that were addressed in the present study. Section 3 briefly reviews related works. Section 4 describes the proposed simulation method for evaluating the practical performance of algorithms. Section 5 presents and discusses the results of the implementation of the proposed method, and the conclusions of the study are finally outlined in Section 6.
2. SIGNIFICANCE OF THE PROPOSED COMPARATIVE PERFORMANCE ANALYSIS METHOD

As mentioned above, the proposed performance analysis method is aimed at extracting critical information regarding the time overhead of frequent scheduler invocations, and the time spent by the schedulers in processing the scheduling events. This is imperative because the overheads of most optimal real-time multiprocessor algorithms are ignored in theoretical studies, resulting in the non-delivery of the claimed optimality of such algorithms in practical applications.

3. LITERATURE REVIEW

Real-time systems are systems that maintain their correctness by outputting results within specific time constraints referred to as deadlines. Meeting the deadlines of a given real-time taskset cannot be achieved without the use of an optimal scheduling algorithm, which is defined in [14] as ‘one which may fail to meet a deadline only if no other one can’. In other words, an optimal scheduling algorithm successfully schedules all the tasks without missing any deadline for a schedulable taskset [9, 12]. Although there have been numerous proposals of optimal real-time multiprocessor scheduling algorithms over the last few years, many of them are impracticable. A recent study [13] showed that optimal scheduling algorithms have significant amounts of scheduling overheads in terms of the numbers of task pre-emptions and migrations, which substantially affect their practicality [9, 13]. An optimal real-time multiprocessor scheduling algorithm should therefore not only meet the deadline constraints of a schedulable taskset, but also have limited scheduling overheads to ensure practicability. The following subsections highlight some features of the most relevant recently reported real-time multiprocessor scheduling algorithms.

P-Fair [15]: This quantum-based algorithm is very strictly and known as Fair Scheduling. It strictly follows the fluid schedule model each time quantum and hence; forces each task to make progress in its execution every time quantum. Therefore, it is known to be computationally expensive and thus; causes a very high overhead in practice due to the frequent task preemptions and migrations.

Boundary Fair (BF) [16, 17]: This was the first algorithm to utilise the Deadline Partitioning (DP) technique. Like P-Fair, it uses quantum-based timing, but enforces fairness at the
deadlines of tasks. It, however, also has the disadvantages of complexity and a high overhead in dealing with round-off issues regarding the amount of allocated work.

Largest Local Remaining Execution First (LLREF) [18]: This was the first quantum based DP-Fair algorithm. It performs unnecessary work and then resorts all the jobs during each scheduling event, completing those with the least laxity. It incorporates ‘T-L Plane’ visualisation.

LRE-TL [19]: Funk and Nadadur [19] proposed the LRE-TL algorithm as an optimal improved version of LLREF. The key feature of the algorithm is the elimination of the need to select tasks for execution based on the largest local remaining execution time within each time slice. In fact, any task with a remaining local execution time will do. This significantly decreases the number of migrations per time slice compared to LLREF. Funk and Nadadur [19] also showed how LRE-TL could be applied to sporadic tasksets and proved its optimality for sporadic tasksets with implicit deadlines. Funk [12] also extended LRE-TL to the support of sporadic tasks with unconstrained deadlines and proved the optimality of the algorithm for the application. A modified version of LRE-TL that significantly decreases the number of task migrations without affecting the optimality of the algorithm has also be developed [20].

USG [13]: USG is a semi-optimal real-time multiprocessor scheduling algorithm that decreases the number of task pre-emptions and migrations and enables the achievement of high levels of schedulability. The main idea behind USG algorithm is relaxing the fairness rule totally, and hence avoiding the enormous number of scheduling overheads in-terms of task preemptions and migrations it generates, though it ensures the optimality of the algorithm. The algorithm (USG) uses a global tasks queue ordered in an increasing laxity. The tasks with zero laxity have higher priority and are always scheduled for immediate execution. If new tasks with less laxity arrive while tasks with more laxity are being executing, the former are instructed to wait until they reach zero laxity, at which time they are considered for execution. However, as a penalty of relaxing the fairness rule totally, USG can sometimes miss a few deadlines, though these are sufficiently few to be tolerated to achieve the benefits of the significantly reduced task pre-emptions and migrations [13].
4. METHODOLOGY

The experimental simulation procedure for both LRE-TL and USG is shown in Figure 1. The process is as follows. Firstly, the required number of tasks, $n$, is set to 4, 8, 16, 32, or 64, and the required number of processors, $m$, is accordingly set to 2, 4, 8, 16, or 32; i.e., $n = 2m$. A while loop then begins to read the tasksets from the specified task file, one-by-one. It should be noted that the task file contains 100000 tasksets, and if $n$ is set to 4, a set of 4 tasks would be read from the file at a time. The scheduler then sets the current time $T_{cur}$ to zero and initialises the scheduling queues [13, 21]. The scheduler of each algorithm is executed for the first 1000 time units for tractability because the task periods are chosen within the range [1, 100]. Hence, the second while loop executes until the current time $T_{cur}$ reaches 1000. As time progresses, the second while loop checks for scheduling events, and if an event is found, the corresponding handler is called upon to handle the fired event, with the statistical matrices updated accordingly. For example, if a Zero-Laxity (Z) event occurs, there would be a pre-emption and migration, and the corresponding pre-emption and migration matrices would be updated. The scheduler also checks whether the pre-empted task will miss its deadline; if so, the deadline matrix is additionally updated. Thereafter, the scheduler instructs each processor to execute its designated task, and the time is advanced. When the second while loop completes its execution, the outer while loop continues with the next tasksets, and this is repeated until all the tasksets in the file are simulated. When the outer while loop finally completes its execution, the collected results in the statistical matrices are printed.
5. RESULTS AND DISCUSSION

The CPU profiler of Oracle JavaTM VisualVM [1] was used to monitor the execution of USG against LRE-TL. Figure 2 shows the interface of Oracle JavaTM VisualVM. The two algorithms were executed on 2, 4, 8, 16, and 32 processors, respectively, using the same above-mentioned tasksets, where were generated with full utilisation. Both algorithms were run for the first 100 time units. For each group of generated tasksets, 1000 samples were
executed. The CPU profiler of JavaTM VisualVM measured the number of invocations for each procedure (method) and algorithm, as well as the total time required for all the invocations.

Figures 3 and 4 respectively show the CPU profiler results for USG and LRE-TL on 2 processors. It can be clearly seen from the figures that the run procedures of both algorithms were executed 1000 times according to the number of tasksets used. USG invoked the initialise procedure 1000 times, implying once per taskset, as noted in Section 4. The total time required for all the invocations of USG was only 82.8 ms. In the case of LRE-TL, in which the initialisation procedure \((TL\_Plane\_Initialize)\) was invoked at the beginning of each TL-plane, there were a total of 6943 invocations for the 1000 tasksets, implying about 7 invocations per taskset. A total time of 2182 ms was required for the invocations.

Further, the procedure for handling the scheduling events in USG \((handleEOrZEvens)\) was invoked 8489 times for the 1000 tasksets, requiring 38.1 ms. The corresponding procedure in LRE-TL \((handleBOrCEvent)\) was invoked 27749 times, requiring 266 ms. The trend was similar for the helper procedures \((removeMin\) and \(removeMaxLaxity\) in USG, and \(removeMin\) in LRE-TL), which are used to insert and remove tasks from the scheduling queues (see Table 1).

Fig.2. Java VisualVM Interface
Table 1 summarises and compares the CPU profiler results for USG and LRE-TL on 2 processors.
The CPU profiler results for both USG and LRE-TL on 4 processors are summarised in Table 2. In this case, the total time required for all the invocations of the initialisation procedure in USG was only 99.1 ms. Conversely, the LRE-TL initialisation procedure (TL_Plane_Initialise) was invoked 12525 times for the 1000 tasksets, implying about 13 invocations per taskset, with a total time of 6993 ms required. Further, the procedure for handling the scheduling events in USG was invoked 15821 times for the 1000 tasksets, requiring a total time of 69.9 ms. The corresponding procedure in LRE-TL was invoked 100024 times, with a total time of 933 ms required.
Table 2. CPU profiler Results for USG and LRE-TL on 4 Processors

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Initialisation</th>
<th>Scheduling events</th>
<th>Removal</th>
<th>Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of invocations</td>
<td>Time (ms)</td>
<td>No. of invocations</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>USG</td>
<td>1000</td>
<td>99.1</td>
<td>15821</td>
<td>69.9</td>
</tr>
<tr>
<td>LRE-TL</td>
<td>12525</td>
<td>6993</td>
<td>100024</td>
<td>933</td>
</tr>
</tbody>
</table>

Table 3 summarises the results of the CPU profiler for both USG and LRE-TL on 8 processors. Here, the time spent on all the invocations of the initialisation procedure in USG was only 143 ms. Conversely, the LRE-TL initialisation procedure (TL_Plane_Initialise) was invoked 22060 times for the 1000 tasksets, implying 23 invocations per taskset, with a total time of 36643 ms required. Further, the procedure for handling the scheduling events in USG was invoked 27292 times for the 1000 tasksets, requiring a total time of 131 ms. The corresponding procedure in LRE-TL was invoked 351941 times, with the total time spent being 1307 ms.
Table 3. CPU Profiler Results for USG and LRE-TL on 8 Processors

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Initialisation</th>
<th>Scheduling events</th>
<th>Removal</th>
<th>Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of invocations</td>
<td>Time (ms)</td>
<td>No. of invocations</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>USG</td>
<td>1000</td>
<td>143</td>
<td>27292</td>
<td>131</td>
</tr>
<tr>
<td>LRE-TL</td>
<td>22060</td>
<td>9772</td>
<td>351941</td>
<td>1175</td>
</tr>
</tbody>
</table>

Table 4 summarises the results of the CPU profiler for both USG and LRE-TL on 16 processors. In this case, all the invocations of the initialisation procedure in USG required only 193 ms. Conversely, the LRE-TL initialisation procedure (TL_Plane_Initialise) was invoked 22060 times for the 1000 tasksets, implying about 23 invocations per taskset, requiring a total time of 9772 ms. The procedure for handling the scheduling events in USG was invoked 44233 times for the 1000 tasksets, requiring a total time of 134 ms. The corresponding procedure in LRE-TL was invoked 351941 times, with a total time of 1175 ms required.

Table 4. CPU Profiler Results for USG and LRE-TL on 16 Processors

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Initialisation</th>
<th>Scheduling events</th>
<th>Removal</th>
<th>Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of invocations</td>
<td>Time (ms)</td>
<td>No. of invocations</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>USG</td>
<td>1000</td>
<td>193</td>
<td>44233</td>
<td>134</td>
</tr>
<tr>
<td>LRE-TL</td>
<td>22060</td>
<td>9772</td>
<td>351941</td>
<td>1175</td>
</tr>
</tbody>
</table>
Table 5 summarises the CPU results for both USG and LRE-TL on 32 processors. Here, the time spent on all the invocations of the initialisation procedure in USG is only 330 ms. Conversely, the LRE-TL initialisation procedure (TL_Plane_Initialise) was invoked 55518 times for the 1000 tasksets, implying about 26 invocations per taskset, with a total time of 48475 ms required. Further, the procedure for handling the scheduling events in USG was invoked 66277 times for the 1000 tasksets, requiring a total time of 340 ms. The corresponding procedure in LRE-TL was invoked 3456556 times, with a total time of 7306 ms required.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Initialisation Invoked</th>
<th>Initialisation Time (ms)</th>
<th>Scheduling events Invoked</th>
<th>Scheduling events Time</th>
<th>Remove Invoked</th>
<th>Remove Time</th>
<th>Insert Invoked</th>
<th>Insert Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>USG</td>
<td>1000</td>
<td>330</td>
<td>66277</td>
<td>340</td>
<td>2875+221903=224778</td>
<td>251</td>
<td>224778</td>
<td>31.9</td>
</tr>
<tr>
<td>LRE-TL</td>
<td>55518</td>
<td>48475</td>
<td>3456556</td>
<td>7306</td>
<td>6480930</td>
<td>2793</td>
<td>4105416</td>
<td>644</td>
</tr>
</tbody>
</table>

6. CONCLUSION

This paper presented a practical method for analysing the performance of optimal real-time and semi-optimal multiprocessor scheduling algorithms. Using LRE-TL and USG as representative examples of the two types of algorithms, respectively, the proposed method was implemented on machines equipped with 2 and 4 processors, respectively. Both algorithms were executed on the same tasksets and monitored by the CPU profiler of Oracle JavaTM VisualVM. The results of the CPU profiler revealed that USG outperformed LRE-TL with respect to the number of invocations of the procedures for handling the scheduling events and the total time spent on the invocations. This indicates the practical applicability of USG, unlike optimal algorithms, which fall short of their theoretical optimality in practical applications.
7. REFERENCES


How to cite this article: