Hard Real-Time Constraints in Implementing the Myopic Scheduling Algorithm

M S Hasan, K Muheimin-Us-Sakib[1], and M A Hossain[2]

Faculty of Computing, Engineering and Technology
Staffordshire University, Stafford, UK
m.s.hasan@staffs.ac.uk,

[1]Institute of Information Technology,
University of Dhaka, Dhaka 1000
muheymin@yahoo.com,

[2]School of informatics, Department of Computing
University of Bradford, UK
m.a.hossain1@bradford.ac.uk

Abstract-**This paper presents an investigation into the constraints in implementing the Myopic scheduling algorithm. The Myopic algorithm is a hard real-time (RT) task scheduling algorithm for multiprocessor systems. It selects a suitable task based on a heuristic function, H from a subset (feasibility check window, K) of all ready tasks. On the other hand, the original heuristic scheduling algorithm chooses the task with the least H value from all N ready tasks. Performance of the Myopic algorithm significantly depends on the chosen heuristic function and the size of the feasibility check window since it considers only K tasks from N tasks (where $K \leq N$). This research investigates the impact of scheduling non-periodic hard RT tasks using the Myopic algorithm and evaluates the performance for different parameters to demonstrate the merits and constraints of the algorithm. The effects of the feasibility check window size, K , choice of heuristic function, H , the worst case processing time of tasks, T_p on the performance of the Myopic algorithm under various loads are investigated. Finally, the performance of the algorithm is evaluated as task completion ratio, presented and discussed through a set of experiments.**

Keywords-**Original heuristic scheduling algorithm, Myopic algorithm, feasibility check window size K , Earliest Starting Time (EST), processing time T_p , task deadline (T_D), laxity time (T_L).**

I. INTRODUCTION

Hard real-time systems are used in time critical applications like avionics, nuclear weapon control, robotics etc [1]. These systems must guarantee that all/most of the tasks are completed within their explicit deadlines and resource utilisation is maximised [2]. It requires an efficient task scheduling algorithm. Scheduling can be performed in two ways – statically and dynamically. In static algorithms, the order of tasks and the times they start execution can be determined in advance and they are suitable for RT periodic tasks [2]. On the other hand, dynamic algorithms deal with non-periodic tasks whose characteristics are not known a priori. When new tasks arrive, the scheduler selects the most suitable task [2]. In practice, RT task starting time, deadline etc. are not known in advance. Therefore, it creates the challenge of efficient scheduling non-periodic RT tasks dynamically. This paper investigates the application of the *Myopic* scheduling algorithm [3] for dynamically generated non-periodic RT tasks on a uni-processor system.

Periodic RT tasks and static scheduling strategies are considered in many literatures for instance, [4], [5] etc. Upper bound of processor utilisation for hard RT preemptive tasks and the rule for optimum *Fixed Priority* scheduling have been derived in [4]. The task model of [4] also assumes that they are independent, periodic with fixed priorities and fixed execution time. In [6], the authors showed that for uni-processor systems, scheduling with *simple heuristic* considering resource requirements is more effective than the scheduling that ignores resource requirements e.g., earliest deadline fast (EDF). Run time scheduling problems and sufficient conditions optimal scheduling for multi-processor systems have been investigated in [7]. Both the *Myopic* and the *original heuristic scheduling* algorithms use heuristic function and are proposed in [3] for multiprocessor systems. It has been shown that *integrated heuristic function* (formed by deadline and earliest starting time constrains) performs better than *simple heuristics* such as *EDF*, minimum processing time first etc. The Myopic algorithm has less computational overhead and therefore, is more effective than the original heuristic scheduling algorithm [3]. In [8], the author proposed a new heuristic function $H(T) = T_D + W_1 \times T_{EST} + W_2 \times T_L$ for the Myopic algorithm to consider deadline, resource requirement and processing (Laxity) time of tasks in RT multiprocessor systems.

The proposed multiprocessor RT scheduling algorithm in [2] is a variant of the Myopic algorithm that exploits the parallelism in the tasks. It has investigated the effect of feasibility check window size and relative weight of deadline and resource requirement for multi-processor systems. It follows *centralised scheduling scheme*: a central processor called *scheduler* distributes the arriving tasks to the other processors. Fault-tolerant extensions of the Myopic algorithm are presented in [9] and [10] for multiprocessor systems. The algorithm in [9] is capable of handling both processor and task failures. The algorithm proposed in [10] maintains two copies of each task and can handle more than one processor faults. An extension of the Myopic algorithm with resource reclaiming capability has been proposed in [11] that can execute non-periodic RT tasks concurrently on multiple processors. In [12], the author presented a variant of the Myopic algorithm for RT task scheduling in multiprocessor systems that can execute self-diagnosis under normal load situation.

The Myopic algorithm was basically proposed for dynamic task scheduling in multiprocessor systems and most of the previous works did not consider the relationship between worst case task processing time, T_P and feasibility check window size, K . The Myopic algorithm for uni-processor system with non-periodic RT tasks and the effects of window size *K* has been reported earlier [13]. This research is the extended investigation to explore the impact in implementing the hard real-time scheduling. The impacts of feasibility check window size *K*, values of *W*, worst case processing time T_P on the performance under various load conditions are investigated using simulations.

This investigation explores the impact of scheduling non-periodic hard RT tasks using the Myopic algorithm and evaluates the performance to demonstrate the merits and constraints of the algorithm. The impact of the feasibility check window size, K , choice of heuristic function, H, the worst case processing time of tasks, T_p on the performance of the Myopic algorithm under various loads are investigated. It is worth noting that the performance of the algorithm is evaluated as task completion ratio, presented and discussed through a set of experiments. Section II of the paper describes the real-time scheduling algorithms and task model. Simulation details and results are discussed in section III. Finally, section IV draws conclusion of the paper.

II. REAL TIME SCHEDULING ALGORITHMS

A scheduling algorithm is said to be *preemptive* if the currently executing task leaves the CPU when a higher priority task arrives. Otherwise, it is *non-preemptive*. Some of the frequently used RT scheduling algorithms are discussed below.

- Fixed priority (FP) scheduling: A fixed priority is assigned to each task before execution and it is preemptive. It is the most common scheduling strategy and widely used in commercial RT operating systems [14]. However, it generates irregular delay patterns and the CPU is not utilised properly [2], [14].

- Rate Monotonic (RM) scheduling: Shorter period tasks have higher priorities and it is pre-emptive [15]. It is optimal in the sense that if there exists any static priority assignment algorithm that satisfies the deadlines of a task set, then RM also satisfies the deadlines of that task set [14], [15].

- Earliest Deadline First (EDF) scheduling: Tasks are prioritised based on their deadline and task with the earliest deadline is assigned the highest priority. The priorities are dynamic and task period can vary [14]. It is optimal in the following sense. If it is possible to schedule a task set using preemption then EDF generated schedule will also meet the deadlines of the task set.

A. The Task Model

This section explains the definitions/terms used in the paper followed by the original heuristic and the Myopic algorithms.

- A task is considered to be *feasible* if the scheduling satisfies its timing and resource constraints [2].
- The schedule in which every task is feasible is called *feasible schedule* [2], [9].
- A feasible schedule for a subset of tasks is defined as a *partial schedule* [2], [3].
- A partial schedule is said to be *strongly feasible* if all the schedules obtained by extending it by any one of the remaining tasks are also feasible [2], [3], [9].

Tasks are considered to have the following properties [2], [3] to evaluate the constraints and impacts:

- T_G (Generation time): An absolute time when the task is generated or submitted.
- T_P (Processing time): The worst case processing time of a task.
- T_D (Deadline): An absolute time by which it must complete its execution.
- $\{R_{REQ}\}$ (Resource requirement vector): Resources can be requested exclusively or in shared mode by tasks.
- T_{EST} (Earliest start time): An absolute time when a task can begin execution. In other words, the time when all the required resources of a task are available. It must meet the condition $(T_D - T_P) \geq T_{EST} \geq T_G \geq 0$.
- T_L (Laxity time): It defines the urgency of the task and computed as $T_L = T_D - T_{EST} - T_P$. A task with zero laxity must be executed immediately [7].
- Tasks are aperiodic and non-preemptive.

B. The Original Heuristic Scheduling

The original heuristic scheduling algorithm starts with an empty schedule and adds tasks one by one [3]. At each level, it selects a task from the set of all available tasks $\{Task\}$ based on a heuristic function, H. Anyone of the following functions can be used as H in this scheduling [3].

- $H(T) = Min(T_D)$: The earliest deadline of all tasks or *EDF*.
- $H(T) = Min(T_P)$: The task with the shortest processing time.
- $H(T) = Min(T_{EST})$: The earliest T_{EST} carrying task.
- $H(T) = Min(T_D - T_{EST} - T_P)$: The task with the shortest laxity time.
- $H(T) = T_D + W \times T_P$
- $H(T) = T_D + W \times T_{EST}$

The first four functions are called *simple heuristics* and last two choices are *integrated heuristics* that combine two simple heuristics using weight parameter W [3]. Here, W controls the relative importance of T_D with T_P or T_{EST}. The algorithm picks the task with the smallest H (heuristic) value to form the partial schedule. After choosing the first task, the schedule becomes a *partial schedule* [2], [3]. Then the algorithm checks for the strong feasibility. If it is not met then the algorithm can take any one of the following steps:

- The algorithm may abort
- It can backtrack and change the latest chosen task etc.

For N tasks set there will be N steps and in each step the algorithm will compute H for at best N tasks. So, the complexity of the algorithm is $O(N^2)$.

C. Myopic Scheduling Algorithm

The Myopic algorithm is explained with the following additional terms [3]:

- $\{Task\_remaining\}$: the tasks that have not been scheduled.
- N_R: the number of tasks in the set $\{Task\_remaining\}$.
- K: Feasibility check window, the maximum number of tasks in $\{Task\_remaining\}$ that will be considered.
- N_K: Actual number of tasks that are considered, $N_K = Min(K, N_R)$.
- $\{Task\_considered\}$: the first K tasks in the $\{Task\_remaining\}$ that are considered.

The tasks in $\{Task\_remaining\}$ are always kept sorted by increasing order of deadlines, T_D [6]. The Myopic algorithm works like the original Heuristic Algorithm with the exception that it applies the heuristic and strong feasibility to only K tasks, ($K \le N$) that is defined as *feasibility check window* instead of N tasks [3], [9]. This algorithm is called the Myopic because it is a short sighted approach for decision making.

There are N steps for including N tasks and in each step H is applied to only K tasks. So, the complexity becomes $O(KN)$, ($K \le N$) [9], [11]. For small value of K (window size) the scheduling executes faster. But since the algorithm considers only few tasks to choose the best one, it exhibits worse performance. On the other hand, if it considers all the tasks $K = N$ then the scheduling operation executes slower and becomes the original scheduling algorithm [3].

III. EXPERIMENTS AND RESULTS

The Myopic algorithm is implemented on a high performance Pentium PC using C++ programming language, assuming the model as discrete time model. The performance is measured by task completion ratio since it is the most important metric for RT scheduling algorithms [8]. For a specific set of conditions or value of parameters, five observations were taken and presented. The conditions and parameters are described below:

A. Choice of Heuristic Function

Performance of the Myopic algorithm significantly depends on the chosen heuristic function [8]. Among all the heuristic functions listed in [3], $H(T) = T_D + W \times T_{EST}$ considers both the deadline and resource requirements [2], [11] and exhibits better performance than the other choices [3], [12]. Therefore, it has been chosen as *H* in this investigation.

B. Range of Weight parameter, W

This parameter controls the relative weight of T_D and T_{EST}. If $W = 0$ then the heuristic becomes completely $H(T) = Min(T_D)$ or in other words, *EDF* algorithm. If, W is set to 1, then T_D and T_{EST} both have the same importance. For the purpose of simplicity, this investigation considered the cases $W = 0.5$ and $W = 1.0$.

C. Task Properties

Tasks are considered as non-preemptive and non-periodic. Tasks are scheduled when the current task is completed. All the tasks have random T_D, T_P, T_G and T_{REQ} values. T_P is varied from 5 to 21 time units.

D. Range of feasibility window size, K

Window sizes implementing the algorithm are considered for $2, 4, 6, 8, 10$. window size 1 is not considered, because the Heuristic $H(T) = T_D + W \times T_{EST}$ with window size 1 becomes *EDF* algorithm.

E. Effect of Load

This section presents the effect of feasibility check window size under different load of tasks. To demonstrate the impact, loads of 200, 500 and 1000 tasks are used for the parameter $W = 0.5$ and $W = 1.0$. The simulation results are described below.

Case 1: 200 tasks with T_P = 10 to 11 time unit and T_L = 100 time unit

Figure 1(a) and 1(b) show the performance of the Myopic algorithm for 200 tasks with processing time 10 to 11 time unit and the value for W is 0.5 and 1.0 respectively. It is noted from the figure 1(a) and 1(b) that the task completion ratio increases slightly for larger window size. It may be due to the fact that for larger window the algorithm gets wider choice to select a suitable task. For window size 2, the choice for the task is very limited and most probably, for this reason, readings show lower task completion ratio. A significant level of oscillation is also noted for window size 6 in figure 1(a) which could be due to the random nature of the tasks. From figure 1(b), it is perceived that the task completion ratio is lower than the performance shown in figure 1(a). In general, it is reflected from figure 1(b) that the task completion ratio drops due to higher relative weight of T_{EST}. It is also noted from figure 1(b) that instead of oscillation the performance becomes constant at window size 6 and higher.

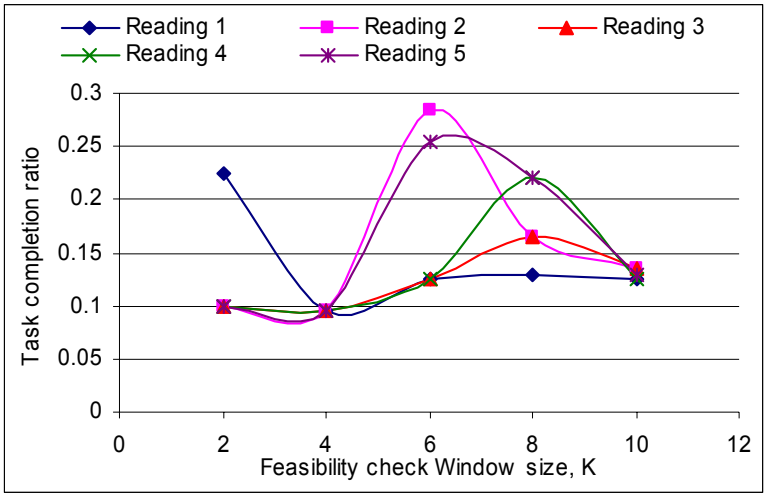

Figure 1(a). Performance of the Myopic algorithm for 200 tasks with T_P = 10 to 11 time unit and W=0.5.

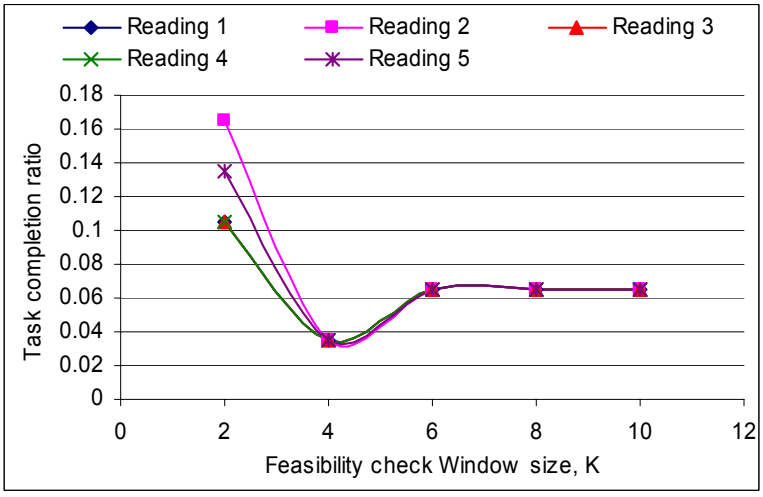

Figure 1(b). Performance of the Myopic algorithm for 200 tasks with T_P = 10 to 11 time unit and W=1.0.

Case 2: 500 tasks with T_P = 10 to 11 time unit and T_L = 100 time unit

Figure 2(a) and 2(b) depict the performance of the Myopic algorithm for 500 tasks with processing time 10 to 11 time unit and the value for W is 0.5 and 1.0 respectively. It is observed from figure 2(a) and 2(b) that they show the similar trend: the task completion ratio

gets higher with the larger window size. These figures also depict significant level of oscillations due to random nature of the tasks.

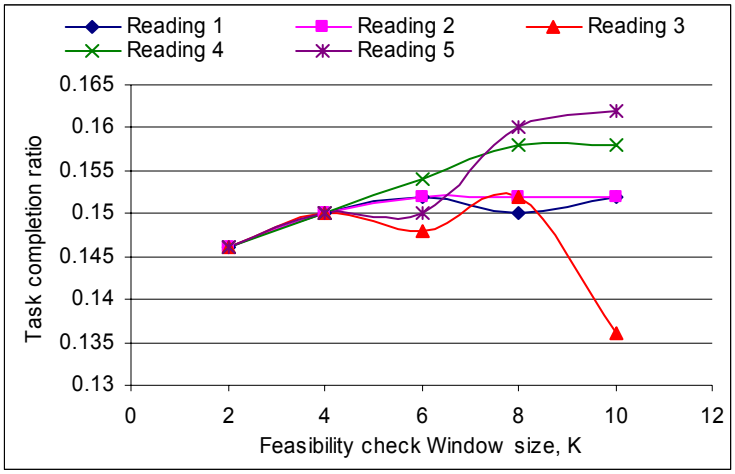

Figure 2(a). Performance of the Myopic algorithm for 500 tasks with T_P = 10 to 11 time unit and W=0.5.

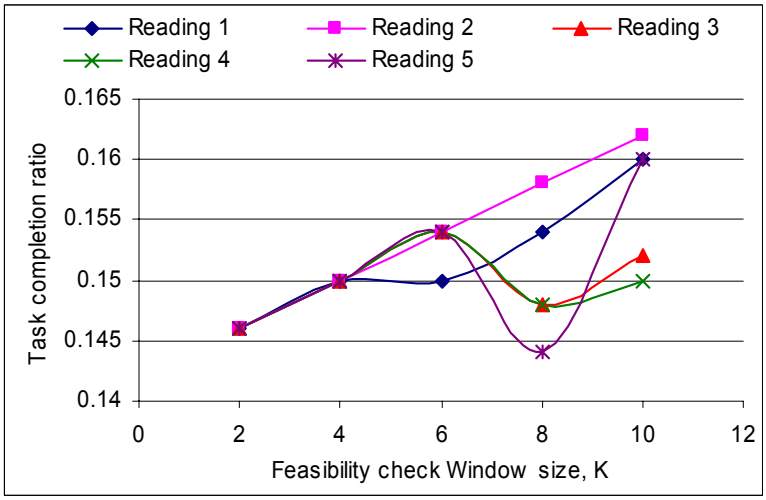

Figure 2(b). Performance of the Myopic algorithm for 500 tasks with T_P = 10 to 11 time unit and W=1.0.

Case 3: 1000 tasks with T_P = 10 to 11 time unit and T_L = 100 time units

Figure 3(a) and 3(b) show the performance of the Myopic algorithm for 1000 tasks with processing time 10 to 11 time unit and the value for W is 0.5 and 1.0 respectively. It can be observed from figure

3(a) and 3(b) that they all show the same tendency: increasing task completion ratio with larger window size. However, some observations show a significant level of oscillations for window size over 4 in figure 3(a) and 3(b). It is also noted that the degree of oscillation is significant for higher window size. However, average task completion ratio falls a little with increasing value of *W*.

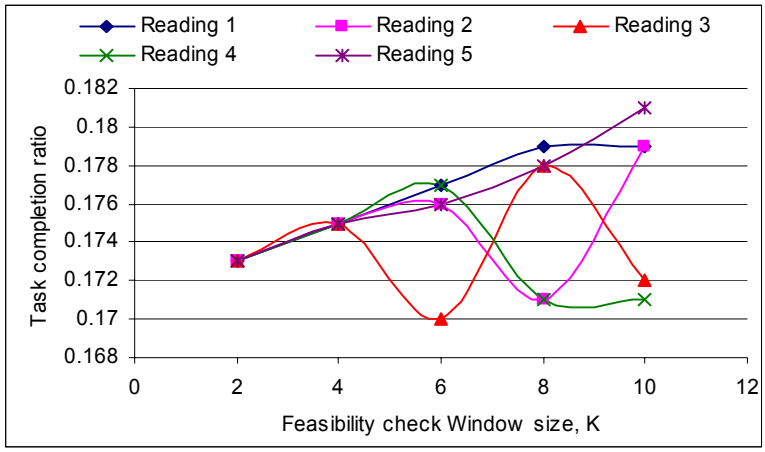

Figure 3(a). Performance of the Myopic algorithm for 1000 tasks with T_P = 10 to 11 time unit and W=0.5.

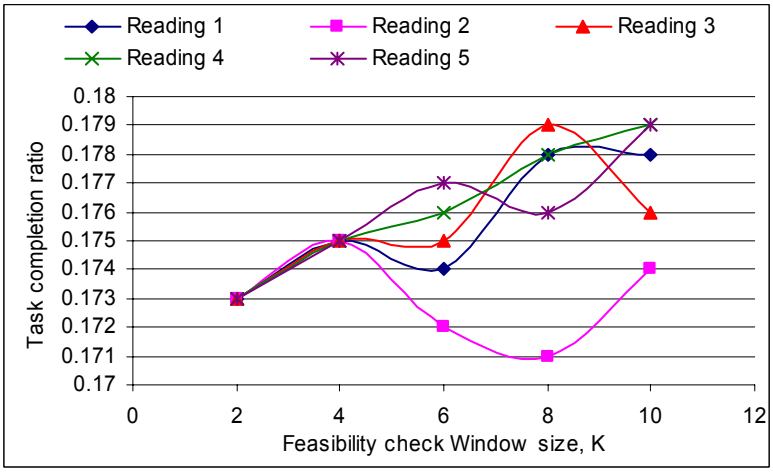

Figure 3(b). Performance of the Myopic algorithm for 1000 tasks with T_P = 10 to 11 time unit and W=1.0.

It can be perceived from the above three cases that the algorithm achieved the best performance for higher number of tasks and the performance increase with the window size. It indicates that the original algorithm should show the best performance. But since the original heuristic algorithm has higher

complexity $O(N^2)$ than that of Myopic $O(KN)$, it spends more time in selecting an appropriate task that can have a bad impact on scheduling tasks of short T_P. This issue has been explored in the following section.

F. Effect of Processing Time, T_P

This section presents the impact of window size due to variation of processing time. To demonstrate the effect, load of 500 tasks with T_P = 5 to 6 and 20 to 21 time unit are used. The load of 500 tasks with T_P = 10 to 11 time unit has been already discussed in the previous section.

Case 1: 500 tasks with T_P = 5 to 6 time unit and T_L = 100 time units

Figure 4(a) and 4(b) show the performance of the Myopic algorithm for 500 tasks with processing time 5 to 6 time unit and the values for W are 0.5 and 1.0 respectively. In this case, the scheduling time dominates T_P and more time is spent for scheduling rather than executing the tasks. So, the task completion ratio drops in both cases: W=0.5 and W=1.0. Since T_P is small compared to scheduling time, the effect of T_{EST} is insignificant in choosing a task. However, figure 4(a) and 4(b) depict the same performance in terms of task completion ratio.

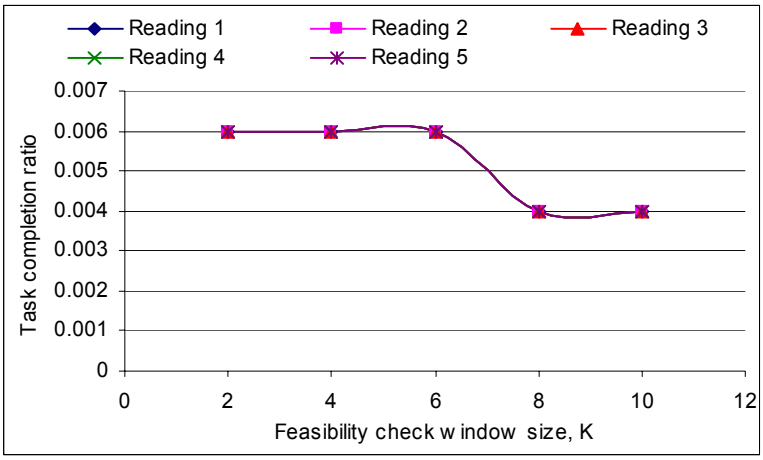

Figure 4(a). Performance of the Myopic algorithm for 500 tasks with T_P = 5 to 6 time unit and W=0.5.

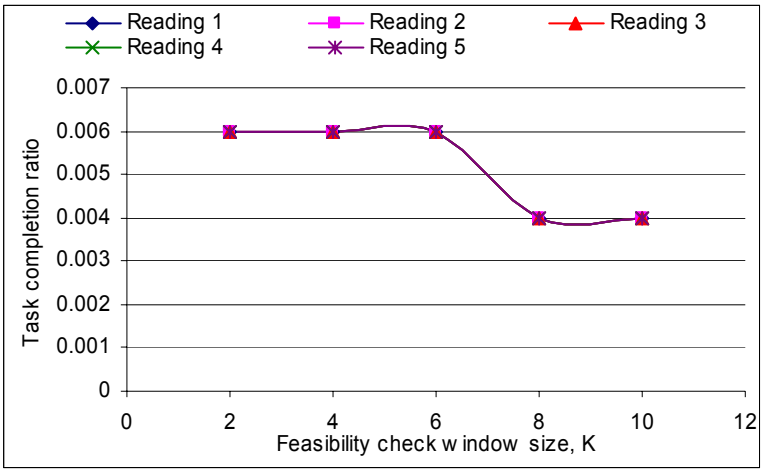

Figure 4(b). Performance of the Myopic algorithm for 500 tasks with T_P = 5 to 6 time unit and W=1.0.

Case 2: 500 tasks with T_P = 20 to 21 time unit and T_L = 100 125 time units

Figure 5(a) and 5(b) depict the performance of the Myopic algorithm for 500 tasks with processing time 20 to 21 time unit and the values for W are 0.5 and 1.0 respectively. Figure 5(a) and 5(b) show that for longer tasks duration, the performance (task completion ratio) is independent of window size. Moreover, it is clearly demonstrated that the value of W does not have any effect on the performance for tasks with higher T_P.

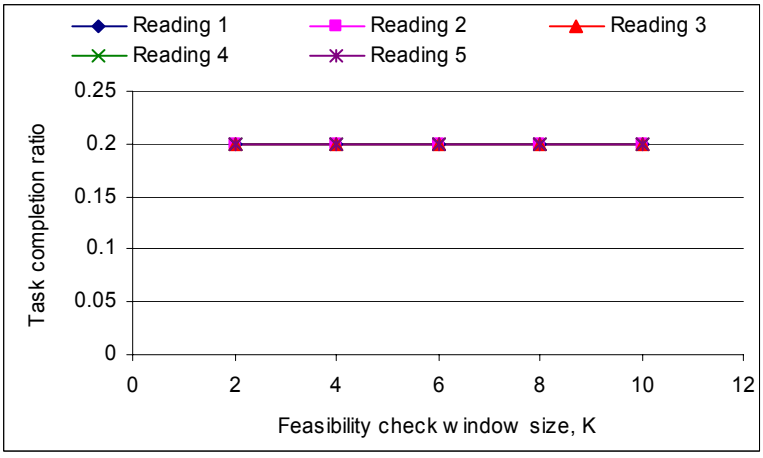

Figure 5(a). Performance of the Myopic algorithm for 500 tasks with T_P = 20 to 21 time unit and W=0.5.

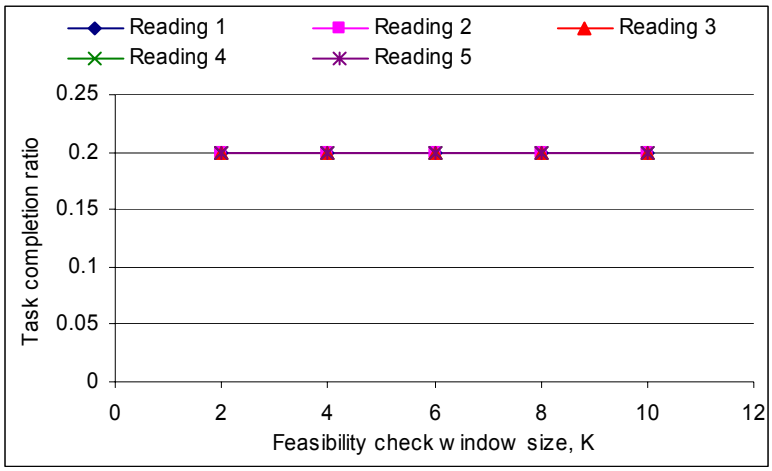

Figure 5(b). Performance of the Myopic algorithm for 500 tasks with T_P = 20 to 21 time unit and W=1.0.

IV. CONCLUSION

This paper has presented the impact of the performance in implementing the Myopic algorithm for different feasibility check window sizes. A set of experiments have been performed to demonstrate the performance issues of the algorithm. It is noted that the window size plays a vital role on the performance of the Myopic algorithm, in particular, for tasks of lower processing time. For relatively large number of tasks and lower processing time, the algorithm achieved better performance and this increases further for larger feasibility check window size. It is also noted that the algorithm achieved better performance for the higher processing time, without any impact of the window size.

Finally, it is worth noting that the processing time and window size has a significant impact on the performance in implementing the Myopic algorithm even in a uniprocessor computing domain.

REFERENCE

[1] M L Dertouzos, and A K Mok, "Multiprocessor On-Line Scheduling of Hard Real-Time Tasks," IEEE Transactions: Software Engineering, vol. 15, No. 12, 1989, pp. 1497-1506

[2] G Manimaran and C S R Murthy, "An efficient dynamic scheduling algorithm for multiprocessor real-time systems", IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 3, Mar 1998, pp. 312 – 319.

[3] K Ramamritham, J A Stankovic and P F Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 2, Apr 1990, pp. 184-194.

[4] C L Liu and J W Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment", Journal of ACM, vol. 20, no. 1, 1973, pp. 46-61.

[5] K Ramamritham, “Allocation and Scheduling of Precedence-Related Periodic Tasks”, IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 4, Apr 1995, pp. 412-420.
[6] W Zhao, K Ramamritham and J A Stankovic, “Scheduling Tasks with Resource Requirements in Hard Real-Time Systems”, IEEE Transaction on Software Engineering, vol. 12, no. 5, May 1987, pp. 567-577.
[7] M L Dertouzos and A K Mok, “Multiprocessor online scheduling of hard-real-time tasks”, IEEE Transactions on Software Engineering, vol. 15, no. 12, Dec 1989, pp. 1497-1506.
[8] Z Xiangbin and T Shiliang, “An improved dynamic scheduling algorithm for multiprocessor real-time systems”, Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies, Aug 2003, pp. 710 - 714
[9] G Manimaran and C S R Murthy, “A new study for fault-tolerant real-time dynamic scheduling algorithms”, Journal of Systems Architecture, vol. 45, no.1, Sep 1998, pp.1-13.
[10] G Manimaran and C S R Murthy, “A fault-tolerant dynamic scheduling algorithm for real-time multiprocessor systems and its analysis”, IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.11, Nov 1998, pp. 1137-1152.
[11] G Manimaran and C S R Murthy, “Dynamic Scheduling of Parallelizable Tasks and Resource Reclaiming in Real-time Multiprocessor Systems”, Proceedings of the 4th International Conference on High Performance Computing, Dec 1997, pp. 206 – 211.
[12] K Mahesh, G Manimaran, C S R Murthy and A K Somani, “Scheduling algorithm exploiting spare capacity and task laxities for fault detection and location in real-time multiprocessor systems”, Journal of Parallel and Distributed Computing, vol.51, no.2, Jun 1998, pp.136-150.
[13] M S Hasan, K M Sakib and M A Hossain, “Hard Real-Time Constraints in Implementing the Myopic Scheduling Algorithm”, Proceedings of the 12th International Conference on Advanced Computing and Communications, Dec 2004, pp. 545-549.
[14] Anton Cervin, “Integrated Control and Real time Scheduling”, PhD Thesis, Lund Institute of Technology, Lund, Sweden, Apr 2003.
[15] M S Branicky, S M Phillips and W Zhang, “Scheduling and Feedback Co-Design for Networked Control Systems”, Proc of the IEEE Conference on Decision and Control, Las Vegas, Dec 2002.