Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment*

Dong-Ik Oh
Department of Computer Science
Soochunhyang University
Asan, Korea

T.P. Baker
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4019

Internet: doh@cs.fsu.edu Phone: (904)644-6672/5452

September 24, 2001

Abstract

We consider the schedulability of a set of independent periodic tasks under fixed priority preemptive scheduling on homogeneous multiprocessor systems. Assuming there is no task migration between processors and each processor schedules tasks preemptively according to fixed priorities assigned by the Rate Monotonic policy, the scheduling problem reduces to assigning the set of tasks to disjoint processors in such a way that the schedulability of the tasks on each processor can be guaranteed. In this paper we show that the worst case achievable utilization for such systems is between $n(2^{1/2}-1)$ and $(n+1)/(1+2^{1/(n+1)})$, where n stands for the number of processors. The lower bound represents 41 percent of the total system capacity and the upper bound represents 50 to 66 percent depending on n. Practicality of the lower bound is demonstrated by proving it can be achieved using a First Fit scheduling algorithm.

1 INTRODUCTION

This paper addresses the problem of determining whether a given set of hard-deadline periodic tasks can be scheduled on a given homogeneous multiprocessor configuration. Such hardware configurations are becoming increasingly common, as the computational demands of real-time systems go up faster than the increases in the speed of single processors.

The preemptive scheduling of hard-deadline periodic tasks on various models of single and multiprocessor systems has been studied by a number of researchers. This work can be categorized according to whether the assignment of priorities to tasks is fixed or dynamic. The merits of dynamic versus static priorities are difficult to compare; merits of each approach

*This work was partially funded by the Ada Joint Program Office under the Ada Technology Insertion Program, through HQ U.S. Army CECOM, Software Engineering Directorate.

have been mentioned in various papers [6, 7] but it is hard to say that one is better than the other. It is generally known that dynamic priority schemes allow a higher level of processor utilization, but fixed priority scheduling gives more predictable performance under transient overloads.

Schedulability conditions in terms of task utilization have been derived for multiprocessor systems using dynamic priority assignment, such as the earliest deadline first (EDF) policy [8, 9]. In the simplest case, when the deadlines of tasks are the same as their periods, schedulability is guaranteed so long as the sum of the task utilizations does not exceed the full capacity of the total number of processors. However, comparable schedulability in terms of total task utilization have not yet been done for multiprocessor systems with fixed priority assignment. In this paper, we consider the case of fixed priority assignment.

For multiprocessor fixed priority systems, scheduling techniques can be further categorized according to the dynamic or static character of the assignment of tasks to processors. With *static binding*, a task is assigned to a processor permanently through all of its executions. With *dynamic binding*, the task-to-processor assignments may be changed during execution of the tasks. Rajkumar, Sha and Lehoczky [4] and Dhall and Liu [2] argue that dynamic binding is not appropriate for multiprocessor systems under priority scheduling. Therefore, in this paper, we concentrate on systems using static binding.

We assume the tasks assigned to each processor are assigned priorities according to the Rate Monotonic (RM) policy. Liu and Layland have shown in [1] that RM scheduling is optimal among fixed priority assignments for a single processor. Precise tests are given in [5, 6] for determining whether a system of tasks is schedulable. However, the simplest and probably the most widely used schedulability test is the Liu and Layland result that the processor utilization for a system of m tasks must exceed $m(2^{1/m} - 1)$ before any task can miss its deadline. We call this the *minimax utilization*, since it is the minimum utilization over all maximal task sets, where a maximal task set is one that "fully utilizes" the processor in the sense that it can be scheduled but the execution time of any one of the tasks cannot be increased without causing some task to miss its deadline. The minimax utilization in the environment of a single processor system is also called the *worst case achievable utilization* since it is the minimum utilization bound for guaranteed schedulability. The limit of the worst case utilization, $m(2^{1/m} - 1)$, as m approaches infinity is ln2. Thus, for utilizations between 1 and ln2 schedulability is guaranteed for all tasks sets; for utilizations between ln2 and 1 schedulability can be guaranteed only by more precise analysis, and not for all task sets.

If RM scheduling is used for the set of tasks on each processor, the multiprocessor scheduling problem reduces to assigning a set of m tasks to n processors. An optimal algorithm should be able to schedule a set of tasks using the least number of processors possible.

The optimal task-to-processor assignment problem is at least as hard as the bin packing problem, which is known to be NP-hard [3]. Therefore, unless $\mathcal{P} = \mathcal{NP}$, there is no polynomial time bounded algorithm to get an optimal task-to-processor assignment.

Several heuristic algorithms are developed and analyzed in [2, 3]. The effectiveness of the algorithms is analyzed in terms of the number of bins/processors required to schedule a given set of tasks.

We are interested in extending this work to obtain a multiprocessor schedulability condition in terms of total task utilization, similar to the Liu and Layland minimax formula, for situations where the number of processors n is fixed. That is, we want to find the multiprocessor worst case achievable utilization bound $U_{min}(n)$, which is the greatest lower bound on the total utilizations of task systems that are not schedulable on n processors. It will follow that we can safely schedule a set of tasks as long as their total utilization does not exceed $U_{min}(n)$. At first glance, it might seem that an optimal task-to-processor assignment algorithm should be able to achieve a utilization between ln2 and 1 on each processor, for a total utilization of at least $n \cdot \ln 2$. This is not true. We will show that the worst case achievable utilization $U_{min}(n)$ lies between $n(2^{1/2} - 1)$ and $(n+1)/(1+2^{1/(n+1)})$, for $n \geq 2$.

The remainder of the paper is organized as follows. In Section 2, we define our system model and the scheduling algorithm, review some of the related work and discuss why the previous work is not directly applicable to our problem. Section 3 proves the worst case achievable utilization under our model. Section 0.4 discusses the quality of the bounds, and utility of the lower bound as a practical schedulability test. Section 0.5 concludes.

2 Background

2.1 System Model

Our system model is based on that of [1]. In short, we assume we have a set of independent periodic tasks whose periods are equal to their deadlines. A few more assumptions are imposed in order to apply the model to multiprocessor environments. The additional assumptions are:

1. Each processor has the same computing power and executes independently of the others.

2. Utilization for each task is less than or equal to 1.

3. Once a task is assigned to a processor it executes only on that processor.

We define some of the symbols used in this paper as follows:

1. τ_i : ith task in the system.
2. C_i : computation time required for τ_i.
3. T_i : period of τ_i.
4. C_i/T_i : utilization of τ_i.
5. P_i : ith processor in the system.
6. m_i : number of tasks assigned to P_i.
7. u_i : current utilization of P_i. i.e. $u_i =$ Sum of utilizations of all tasks assigned to P_i.
8. U : total utilization of the tasks in the system. i.e. $U = \sum_{i=1}^{m} C_i/T_i$. where m is the total number of tasks in the system.

2.2 First Fit Scheduling (FFS) Algorithm and Related Work

The First Fit (FF) algorithm is a heuristic polynomial time bin packing algorithm. The simplicity and effectiveness of this algorithm make it a good alternative to an optimal bin packing algorithm which requires exponential execution time. FFS is the application of FF to the problem of task-to-processor assignments. Here we describe the assignment procedure of FFS:

Step 1: set $i = 1$

Step 2: set $j = 1$

Step 3: if τ_i is schedulable on P_j according to the sufficient schedulability condition given in [1] (*i.e.*, $u_i \leq m_i(2^{1/m_i} - 1)$),
then assign τ_i to P_j and go to step 5

Step 4: set $j = j + 1$;
if $j \leq$ total number of processors then go to Step 3;
else Declare Failure and Stop

Step 5: set $i = i + 1$;
if $i \leq$ total number of tasks in the system then go to Step 2;
else Declare Success and Stop

The performance of this and other similar algorithms have been analyzed in several studies [2, 10]. However, in those studies, the effectiveness of the algorithms is measures by the number of bins/processors required for a set of tasks, as compared to the optimum. Such a bound does not translate into a solution to our problem, which is to find a lower bound on how high the total utilization needs to go before we have to worry about missing deadlines, for a given hardware configuration with a fixed number of processors.

In order to see the inadequacy of such results to our problem, we consider (below) one of the theorems presented in [2] for FFS. (Similar but worse bounds are known for the number of processors for the next-fit algorithm. However, it turns out that whether we use the next-fit algorithm or FFS makes no difference for the utilization bounds derived in the present study.)

> Let N be the number of processors required to feasibly schedule a set of tasks by the Rate Monotonic First Fit Scheduling (RMFFS) algorithm and N_0 be the minimum number of processors required to feasibly schedule the same set of tasks. Then, as N_0 approaches infinity, $2 \leq \lim_{N_0 \to \infty} N/N_0 \leq 4 * 2^{1/3}/(1 + 2^{1/3})$.

First, we can't use this as a schedulability test for RMFFS, since it requires us to know N_0. Second, even if we did know N_0, the result above would not give us a very tight bound. Consider the set of n tasks whose utilizations C_i/T_i are $0.5 + \epsilon$ for all τ_i. The minimum number of processors required to feasibly schedule this set of tasks is obviously $N_0 = n$. The theorem does not guarantee schedulability using RMFFS unless the number of processors is at least $2n$. However, the total utilization U of this set of tasks is $0.5n$ as $\epsilon \to 0$. This is less than the worst case achievable utilization bound we are going to present for $2n$ processors, which is $2n(2^{1/2} - 1) \simeq 0.828n$.

Various formulas given in other studies [2, 3] also are stated in terms of the number of bins as in above example, and therefore cannot be used to derive a bound in terms of the total task utilization. This motivated our study.

3 Worst Case Achievable Utilization Bound in Multiprocessor Systems

In this section we give a condition for schedulability with fixed priority scheduling on homogeneous multiprocessor systems in terms of the total utilization U of tasks, by proving the following theorem.

Theorem 1 *The worst case achievable utilization $U_{min}(n)$ with fixed priority preemptive scheduling on a homogeneous multiprocessor system with $n \geq 2$ processors is bounded as follows:*

$$n(2^{1/2} - 1) < U_{min}(n) \leq (n + 1)/(1 + 2^{1/(n+1)})$$

Proof of the upper bound:

In order to prove the upper bound, we show that for any $n \geq 2$ there is a set of tasks with total utilization $U = (n+1)/(1+2^{1/(n+1)}) + \epsilon$ which cannot be scheduled (by any algorithm) if $\epsilon > 0$.

We use a set of tasks similar to the set used in [2]. Consider the set of tasks $\{\tau_i = (C_i, T_i) | i = 0, \ldots, n\}$, where $C_i = \alpha^i$, $T_i = \beta C_i$, $\alpha = 2^{1/(n+1)}$ and $\beta = \alpha + 1$.

When any of two tasks in the set are scheduled on a single processor they will either fully utilize [1] the processor or one of them will miss the deadline. In order to verify this, we apply the critical time zone argument, described in [1], to analyze the schedulability of task sets on a single processor with Rate Monotonic scheduling. The critical time zone for a task τ_j is a time interval between a request for τ_j and the corresponding deadline, such that the available time to execute τ_j (after the time used by higher priority tasks is taken away) is minimized. It is shown in [1] that a critical time zone occurs between time 0 and time T_j, when τ_j is requested simultaneously with all higher priority tasks at time 0.

Consider any pair of tasks τ_i and τ_j where $i < j$. Since τ_i has a shorter period, it gets higher priority according to Rate Monotonic scheduling. For the schedulability of τ_j, there are two mutually exclusive cases to consider:

1. $T_i + C_i \geq T_j$ (see Figure 0.1)

   In this case, τ_i and τ_j are exactly schedulable together on a single processor if $T_i - C_i = C_j$, and τ_j overflows if $T_i - C_i < C_j$.

   $$T_i - C_i = \beta\alpha^i - \alpha^i = \alpha^j(\frac{\beta - 1}{\alpha^{j-i}}) = C_j(\frac{\alpha}{\alpha^{j-i}})$$

   For $j = i+1$, we have $j - i = 1$. Since $T_i - C_i = C_j(\frac{\alpha}{\alpha^{j-i}})$, $T_i - C_i = C_j(\frac{\alpha}{\alpha}) = C_j$. This means τ_i and τ_j are exactly schedulable together on a single processor.

   Otherwise, $j > i + 1$, so $j - i > 1$ and $T_i - C_i < C_j$. This means τ_i and τ_j are not schedulable together on a single processor.

2. $T_i + C_i < T_j$ (see Figure 0.2)

   From our choice of values for the task periods, it follows that $T_j < 2T_i$, so in this case τ_i and τ_j are exactly schedulable together on a single processor if $T_j - 2C_i = C_j$, and τ_j overflows if $T_j - 2C_i < C_j$.

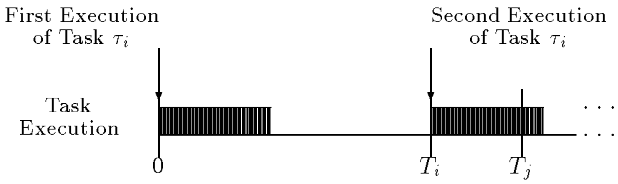

Figure 1: Critical zone for τ_j when $T_i + C_i \geq T_j$

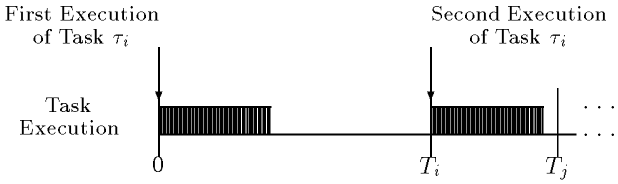

Figure 2: Critical zone for τ_j when $T_i + C_i < T_j$

$$\begin{aligned} T_j - 2C_i &= \beta\alpha^j - 2\alpha^i = \alpha^j(\beta - \frac{2}{\alpha^{j-i}}) \\ &= C_j(1 + \alpha - \frac{2}{\alpha^{j-i}}) \\ &= C_j(1 + \frac{\alpha^{j-i+1} - 2}{\alpha^{j-i}}) \end{aligned}$$

For $i = 0$ and $j = n$, we have $j - i = n$ and so $\frac{\alpha^{j-i+1}-2}{\alpha^{j-i}} = \frac{\alpha^{n+1}-2}{\alpha^n} = \frac{2-2}{\alpha^n} = 0$. It follows that $T_j - 2C_i = C_j$, and so τ_i and τ_j are exactly schedulable together on a single processor.

Otherwise, $j - i < n$, so $\frac{\alpha^{j-i+1}-2}{\alpha^{j-i}} < 0$, and $T_i - 2C_i < C_j$. This means τ_i and τ_j are not schedulable together on a single processor.

Adding up the utilizations of all of these tasks, we get $U = (n+1)/(1 + 2^{1/(n+1)})$. Now, consider increasing U by $\epsilon > 0$ and distributing a non-zero amount of ϵ to each task in the system. That is, consider increasing C_i by $\epsilon_i > 0$ where $\sum_{i=0}^{n} \epsilon_i = \epsilon$. If any pair of tasks are scheduled on a processor, one of them will miss the deadline. However, ϵ can be arbitrarily small, since each of the ϵ_i's can be arbitrarily small. We have shown that this set of tasks might be scheduled using n processors, but the task set becomes unschedulable with an arbitrarily small increase in total utilization. This shows that we have an upper bound on the worst case achievable utilization, $U_{min}(n)$.

Proof of the lower bound:

IN order to prove the lower bound, we show that for any set of tasks and for any $n \geq 2$ the FFS algorithm can produce a schedulable task-to-processor assignments using n processors as long as $U \leq n(2^{1/2} - 1)$.

The proof is by induction on the number of tasks. Let m denote the total number of tasks in the system. Note that by an assumption made in Section 2.1, $C_i/T_i \leq 1$ for all tasks τ_i.

(Basis case):

When $m = 1$, we have only one task and we can schedule it on one processor, since $C_i/T_i \leq 1$. It follows trivially that for every $n \geq 2$, if $U \leq n(2^{1/2} - 1)$, FFS produces a schedulable task-to-processor assignment using n processors.

(Induction case):

Assume the lower bound holds when $m \leq k$. Consider the case when $m = k + 1$.

We want to show that, for every $n \geq 2$, if $U = \sum_{i=1}^{k+1} C_i/T_i \leq n(2^{1/2} - 1)$ then FFS produces a schedulable assignment for all $k + 1$ tasks using n processors.

In this case we can reason from the given condition on U as follows:

$$
\begin{aligned}
& \sum_{i=1}^{k+1} C_i/T_i \leq n(2^{1/2} - 1) \\
\Rightarrow\ & \sum_{i=1}^{k} C_i/T_i + (C_{k+1}/T_{k+1}) \leq n(2^{1/2} - 1) \\
\Rightarrow\ & \sum_{i=1}^{k} C_i/T_i \leq n(2^{1/2} - 1) - C_{k+1}/T_{k+1} \\
\Rightarrow\ & \sum_{i=1}^{k} C_i/T_i \leq (n-1)(2^{1/2} - 1) + (2^{1/2} - 1) - C_{k+1}/T_{k+1}
\end{aligned}
$$

There are two mutually exclusive cases:

1. $C_{k+1}/T_{k+1} \geq 2^{1/2} - 1$.

   In this case we can reason from the given condition on U as follows:

$$
\begin{aligned}
& \sum_{i=1}^{k} C_i/T_i \leq (n-1)(2^{1/2} - 1) + (2^{1/2} - 1) - C_{k+1}/T_{k+1} \\
\Rightarrow\ & \sum_{i=1}^{k} C_i/T_i \leq (n-1)(2^{1/2} - 1)
\end{aligned}
$$

   Given the above condition, we want to show that k tasks are schedulable using $n - 1$ processors.

There is a special case to consider when $n = 2$. If $n = 2$, $\sum_{i=1}^{k} C_i/T_i \leq 2^{1/2} - 1$ so tasks $\tau_1, \ldots, \tau_k$ are schedulable using $n - 1$ processors, which in this case is one processor.

For $n > 2$, by the induction hypothesis, k tasks are schedulable using $n - 1$ processors. Consider scheduling the task τ_{k+1}. Since $\tau_1, \ldots, \tau_k$ are schedulable using $n - 1$ processors, even if the FFS algorithm is not able to schedule τ_{k+1} on any of the processors $P_1, \ldots, P_{n-1}$, it still should be able to schedule τ_{k+1} using P_n.

2. $C_{k+1}/T_{k+1} < 2^{1/2} - 1$.

   By the induction hypothesis, for any $n \geq 2$ FFS is able to produce a schedulable partition for the set of tasks $\tau_1, \ldots, \tau_k$ using n processors, because the given condition $U = \sum_{i=1}^{k+1} C_i/T_i \leq n(2^{1/2} - 1)$ implies $\sum_{i=1}^{k} C_i/T_i \leq n(2^{1/2} - 1)$. When k tasks are scheduled on n processors there should be at least one processor P_i with $u_i \leq 2^{1/2} - 1$, because otherwise $U > n(2^{1/2} - 1)$. Among those processors whose utilization is less than or equal to $2^{1/2} - 1$, let P_i be the processor whose index i is the largest. That is to say pick P_i such that $\forall_{t=1 \cdots n}(u_t \leq 2^{1/2} - 1 \Rightarrow t \leq i)$.

   Now, assume that FFS cannot schedule τ_{k+1} on any of the processors $P_1, \ldots, P_n$. In particular, consider the case of P_i. If $m_i = 1$, we have utilization with τ_{k+1} of $u_i + (C_{k+1}/T_{k+1}) < 2(2^{1/2} - 1)$, so τ_{k+1} would be schedulable. Thus, the only condition under which FFS is not able to schedule τ_{k+1} on P_i is if:

$$m_i \geq 2 \text{ and } u_i + C_{k+1}/T_{k+1} > m_{i+1}(2^{\frac{1}{m_{i+1}}} - 1)$$

   However, if this condition occurs, there should be at least one other processor P_j so that $u_j \leq 2^{1/2} - 1$ and $i \neq j$, because otherwise $U > n(2^{1/2} - 1)$. Furthermore, $j < i$ because we picked i to be the largest index among all processors whose utilization is less than or equal to $2^{1/2} - 1$. Then, since $m_i \geq 2$, we should have a task τ_s on the processor P_i such that $C_s/T_s \leq (2^{1/2} - 1)/2$. However, this means we could schedule τ_s on P_j instead P_i because $u_j + C_s/T_s < m_{i+1}(2^{1/m_{i+1}} - 1)$. This contradicts the assignment policy of the FFS algorithm, so it must be that FFS is able to schedule τ_{k+1} on P_i.

□

4 Utility of the Lower Bound

The lower bound given in Theorem 1 is a sufficient condition for guaranteed schedulability in multiprocessor fixed priority preemptive scheduling systems. One can easily compute the total utilization of a set of tasks, and if it is below this bound one can rely that it is schedulable by the FFS algorithm.

An important feature of this schedulability test is that it can be applied in systems where the task-to-processor assignments are not entirely static. That is, if a new task arrives one can assign it to a processor without fear of causing any other tasks to miss their deadlines, as long as the sufficient condition is met. This feature is especially important in online scheduling where one cannot tolerate the large overhead of finding a new optimal task-to-processor assignment (and the overhead of reassigning the existing tasks) each time a new task enters the system. This extra feature comes from the use of the FFS algorithm in the proof of the lower bound. With other kinds of bin-packing algorithms such as First Fit Decreasing and First Fit Increasing [3] this kind of online scheduling is not possible because all the existing task-to-processor assignments have to be reexamined.

Our upper and lower bounds on the achievable utilization, $n(2^{1/2}-1)$ and $(n+1)/(1+2^{1/(n+1)})$, are plotted in Figures 0.3 and 0.4. From these figures, it may seem that in the case of a small number of processors the sufficient condition ($U \leq n(2^{1/2}-1)$) for guaranteed schedulability is weak. However, comparing this to the corresponding schedulability condition for single processor systems ($U \leq \ln 2$) we see that for $n = 2$ we already have better total utilization than with a single processor — i.e. the second processor is not wasted.

5 Conclusion

In this study we have shown that the worst case achievable utilization for n processor systems with static task-to-processor assignment and fixed priority preemptive scheduling lies between $n(2^{1/2}-1)$ and $(n+1)/(1+2^{1/(n+1)})$. The lower bound is a practical schedulability test, as it is a constructive result based on analysis of the First Fit scheduling algorithm. An extra feature of this schedulability test is that it can be applied in systems where new tasks may arrive dynamically.

So far as we know, this is the first result relating the schedulability of multiprocessor fixed priority preemptive scheduling systems to total task utilization. For further studies, it would be interesting to try to narrow the interval between the lower and upper bounds, with or without imposing some other restrictions.

References

[1] Liu, C.L. and Layland J. W. "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", JACM 20.1, 1973, pp. 46–61.

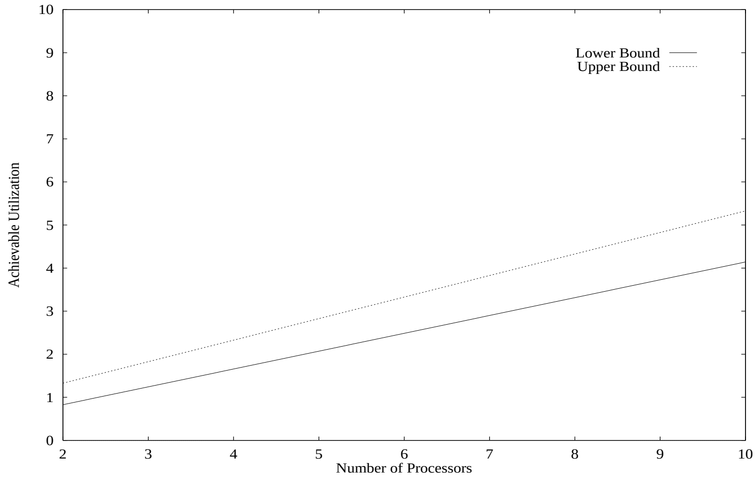

Figure 3: Lower and upper bounds on worst case achievable utilization

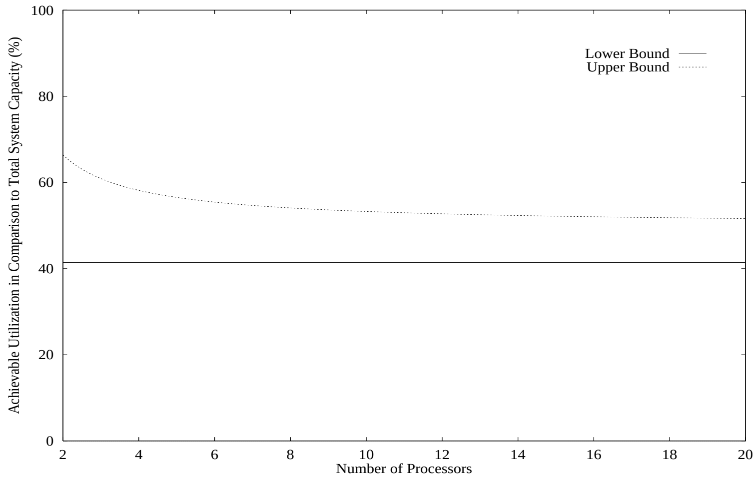

Figure 4: Lower and upper bounds as percent of total system capacity

[2] Dhall, S.K. and Liu, C.L. "On a Real-Time Scheduling Problem", Operations Research 26.1, February 1979, pp. 127–140.

[3] Garey, M. R. and Johnson, D. S. Computers and Intractability. New York: W.H. Freeman, 1979, pp. 121–137.

[4] Rajkumar, R., Sha, L. and Lehoczky, J.P. "Real-Time Synchronization Protocols for Multiprocessors", Proceedings of the Real-Time System Symposium, IEEE, 1988, pp. 259–272.

[5] Joseph, M. and Pandya, P., "Finding Response times in a Real-Time System", BCS Computer Journal, 29 (5), 1986, pp. 390–395.

[6] Lehoczky, J.P., Sha, L., Ding, Y., "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", Proceedings of the Real-Time System Symposium, IEEE, 1989, pp. 166–171.

[7] Chen, M.I. and Lin, K. J. "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems", Journal of Real-Time Systems, 2, 1990, pp. 325–346.

[8] Leung, J.Y. and Merrill, M.L. "A Note on Preemptive Scheduling of Periodic, Real-Time Tasks", Information Processing Letters, 11 (3), 1980, pp. 115–118.

[9] Coffman, E.G. Introduction to Deterministic Scheduling Theory, in: E.G. Coffman, Jr., Ed., Computer and Job-Shop Scheduling Theory, New York: Wiley, 1976, pp. 1–50.

[10] Dhall, S.K. "Scheduling Periodic-Time-Critical Jobs on Single Processor and Multiprocessor Computing Systems", Ph.D. dissertation, University of Illinois, Urbana, 1977.