A New Task Model and Utilization Bound for Uniform Multiprocessors

Shelby Funk
Department of Computer Science, The University of Georgia
Email: shelby@cs.uga.edu

***Abstract*— This paper introduces a new model for describing jobs and tasks for real-time systems. Using this model can improve utilization bounds on uniform multiprocessors, in which each processor has an associated speed. Traditionally, a job executing on a processor of speed s for t units of time will perform $s \times t$ units of work. However, this uniform scaling only occurs if tasks are completely CPU bound. In practice, tasks will have some portion of execution that does not scale with the increased CPU speed. Dividing the execution into CPU execution and fixed execution allows the scheduler to place CPU bound tasks on faster processors, as they can take full advantage of the extra speed. This model is used in a new utilization test for EDF scheduling with restricted migration, r-EDF, in which tasks are allowed to migrate, but only at job boundaries. The test is proven to be better than the existing r-EDF test for uniform multiprocessors.**

Keywords: hard real-time systems, periodic tasks, earliest deadline first, uniform heterogeneous multiprocessors, migration,memory bound tasks

I. Introduction

In hard real-time systems, all jobs have deadlines and missing a deadline is considered a system failure. Before these systems can be run, tests must be performed that ensure that no jobs will miss their deadlines. These tests must account for the worst-case behavior of the system, which is a function of both the worst-case arrival configuration of jobs, and their worst-case execution times (WCET). In general, this WCET is estimated and any estimate must provide an *upper bound* of the actual execution time. In order to allow systems to utilize as much of the processor as possible, the estimate of the WCET should be as accurate as possible.

This paper introduces a new task model for scheduling analysis on uniform multiprocessors and uses this model to improve a known schedulability test. This model applies specifically to systems in which processor speeds may vary. It has been explored in the context of dynamic voltage scaling (DVS) processors [9], [2], [7]. To our knowledge, this model has not been used in the context of uniform multiprocessors.

In uniform multiprocessors, each processor has an associated speed s. In traditional analysis, we assume that if a job executes on a processor of speed s for t time units, then $s \times t$ units of work are performed. This assumption is conditional on all jobs scaling perfectly to CPU speed. In reality, jobs have portions of time that are CPU bound and other portions, such as memory accesses, that do not scale with the CPU speed. This paper explores the power of dividing the WCET into two portions and performing schedulability analysis assuming the CPU portion executes more quickly on faster processors, but the "fixed" portion does not. Because we must account for worst case behavior, if we do not distinguish between the CPU portion and fixed portion but assume perfect scaling with the CPU speed, the given WCET must be increased to account for the fact that only part of the execution actually scales with the CPU speed.

Example 1: Assume J's CPU execution time is 4 and fixed execution time is 2 and J can execute either on a processor of speed 1 or on a processor of speed 2. On the speed-1 processor, J takes 6 time units to complete.

On the speed-2 processor, J takes 4 time units to complete because the CPU execution can be completed in half the time. If we were assign J a single execution requirement, we would have to say J's WCET is 8. If we say J's WCET is 6, then we would calculate that J requires only 3 time units when executing on a speed-2 processor, which could result in a deadline miss.

This paper provides an r-EDF-schedulability test using CPU and fixed execution times. r-EDF uses a uniprocessor EDF scheduling algorithm on each processor. This algorithm allows tasks to migrate, but only at job boundaries, when the overhead due to migration is smallest. This model allows for better load balancing among the processors [1], while still putting a bound on the level of migration. The results presented in this paper apply to any type of uniform multiprocessor system, including heterogeneous multicore systems, in which different cores have different clock speeds. The results will have more impact in systems that have larger costs associated with memory accesses, cache misses, etc. Slower memory accesses cause a job's fixed costs to be a larger proportion of the overall execution and it becomes more important to make a distinction between the scalable and non-scalable parts of the job.

This paper is organized as follows. Section II introduces our model and definitions. Section III presents a new test for r-EDF scheduling on uniform multiprocessors. Section IV proves that finding the exact bound presented in Section III is NP-complete and provides an ILP formulation to calculate the value as well as providing approximation methods that efficiently find an upper bound by relaxing the problem assumption. Finally, Section V concludes and discusses future work.

II. Model and Definition

The analysis that follows assumes we are executing a set of n periodic [6] or sporadic [3], [8] tasks $\tau = \{T_1, T_2, \ldots, T_n\}$ on an m-processor uniform multiprocessor $\pi = [s_1, s_2, \ldots, s_m]$, where $s_1 \geq s_2 \geq \ldots \geq s_m$. In a mild abuse of notation, we allow s_i to denote both the i^{th} processor and the speed of the i^{th} processor. $S(\pi)$ denotes the total speed of π – i.e., $\sum_{i=1}^{m} s_i$.

Tasks are described by the 3-tuple $T_i = (p_i, e_{cpu,i}, e_{fixed_i})$, where $p_i, e_{cpu,i}$ and e_{fixed_i} are T_1's period, CPU execution time, and fixed execution time, respectively. We assume $e_{cpu,i}$ scales with processor speed, but e_{fixed_i} does not – if T_i executes on processor s_j, it will require $e_{cpu,i}/s_j + e_{fixed,i}$ time units to complete. Periodic and sporadic tasks generate jobs, and we let $T_{i,j}$ denote the j^{th} job of task T_i. Each job will have CPU and fixed execution requirements $e_{cpu,i}$ and $e_{fixed,i}$, respectively. If T_i is a periodic task, then it will release a new job at times $r_{i,j} = j \cdot p_i$, where $j = 0, 1, 2, \ldots$. If T_i is a sporadic task, then consecutive jobs will be released at least p_i time units apart. For both types of tasks, $T_{i,j}$'s deadline is $r_{i,j} + p_i$.

The utilization of T_i is also split into a CPU portion, $u_{C,i} = e_{cpu,i}/p_i$, and a fixed portion, $u_{F,i} = e_{fixed,i}/p_i$. These values measure the proportion of processor time the CPU execution and fixed executed will require, respectively, when executing on a speed-1 processor. Because e_{cpu} scales and e_{fixed} does not, the required proportion of CPU execution decreases as speed increases, but the proportion of fixed execution does not. Thus, by the optimality of EDF on uniprocessors [6], a task set $\tau = \{T_1, T_2, ..., T_n\}$ is EDF-schedulable on a speed-s uniprocessor if $\sum_{i=1}^{n}(u_{C,i}/s + u_{F,i}) \leq 1$ – or, equivalently, if $\sum_{i=1}^{n}(u_{C,i} + s \cdot u_{F,i}) \leq s$. Hence, we define T_i's total utilization on s_j as follows $u_{i,j} = u_{C,i} + s_j \cdot u_{F,i}$. The total CPU utilization of τ, denoted $U_{cpu}(\tau)$, is $U_{cpu}(\tau) = \sum_{i=1}^{n} u_{C,i}$.

If s_k is executing tasks $T_{i_1}, T_{i_2}, \ldots T_{i_r}$, then the s_k's slack, denoted $slack_k$, is equal to $s_k - \sum_{j=1}^{q} u_{i_j,k}$. This is the maximum total utilization that can be added to s_k without incurring a deadline miss.

A task T_i is said to be *present* on processor s_k at time t if that task's utilization contributes to the calculation of $slack_k$ — i.e., if some job

$T_{i,j}$ was assigned to processor s_k and $r_{i,j} \leq t < r_{i,j} + p_i$. An assignment, $a(.)$ is a mapping of tasks to processors – if $a(i) = k$ at time t then T_i is active on processor s_k at time t. At times, we need to consider the set $\tau \setminus \{T_i\}$ – i.e., τ without some task T_i. We denote this set τ_{-i}.

We let $Pack_{\pi,\tau}$ denote the set of all possible ways of assigning the tasks of τ to the processors of π without allowing slack to be negative on any processor. Finally, we let $MP_{\pi,\tau}$ denote the maximum sum of $s_{a(j)} \cdot u_{F,j}$ that can be achieved for any assignment $a(.)$.

$$MP_{\pi,\tau} = \max_{a \in Pack_{\pi,\tau}} \left\{ \sum_{j \in \tau} s_{a(j)} \cdot u_{F,j} \right\}$$

Intuitively, $MP_{\pi,\tau}$ is the maximum processing capacity that might be wasted due to letting tasks with a large proportion of fixed execution execute on the fastest processors.

III. A New Test for r-EDF on Uniform Multiprocessors

Using the model described above, we can improve the schedulability test for r-EDF on uniform multiprocessors. Theorem 1 below gives the new schedulability test. Corollary 1 proves that theorem provides a tighter bound than the previous schedulability test.

Theorem 1: Let π be any m-processor uniform multiprocessor and let $\tau = \{T_1, T_2, \ldots, T_n\}$ be any periodic or sporadic task set. Define $M_{\pi,\tau}$ as follows.

$$M_{\pi,\tau} = \max_{1 \leq i \leq n} \left\{ (m-1)u_{C,i} + S(\pi) \cdot u_{F,i} + MP_{\pi,\tau_{-i}} \right\} \quad (1)$$

If the following inequality is satisfied

$$U_{cpu}(\tau) \leq S(\pi) - M_{\pi,\tau} \ . \quad (2)$$

then τ is r-EDF-schedulable on π.

Proof: (By contradiction.) Assume a job T_i generates a job at a time t when all of the processors of π do not have enough slack to admit T_i safely. Let t_o be the earliest time at which this occurs. By the optimality of EDF on uniprocessors all jobs with deadlines at or before t_o met their deadlines.

Let $P(t_o)$ be the set of tasks that are present on some processor at time t_o and let $a_o(.)$ denote the mapping of tasks to processors at time t_o — if $T_j \in P(t_o)$ then T_j is executing on processor $s_{a_o(j)}$. Because T_i has a job that has just arrived and all earlier jobs of T_i met their deadlines, T_i can not be in $P(t_o)$. On the other hand, every task of τ other than T_i may be in $P(t_o)$ and these tasks may be assigned to processors in a way that causes the maximum total utilization on π. Therefore, the following is an upper bound for $\sum_{T_j \in P(t_o)} u_{j,a_o(j)}$.

$$\begin{aligned} & \max_{a \in Pack_{\pi,\tau_{-i}}} \left\{ \sum_{T_j \in \tau_{-i}} (u_{C,j} + s_{a(j)} \cdot u_{F,j}) \right\} \\ = & \sum_{T_j \in \tau_{-i}} u_{C,j} + MP_{\pi,\tau_{-i}} \\ \leq & \sum_{T_j \in \tau_{-i}} u_{C,j} + M_{\pi,\tau} - (m-1)u_{C,i} - S(\pi)u_{F,i}. \end{aligned}$$

The last step follows from the definition for $M_{\pi,\tau}$.

Because $slack_k < u_{i,k}$ for all $k, 1 \leq k \leq m$,

$$\begin{aligned} \sum_{k=1}^{m} slack_k &< \sum_{k=1}^{m} \cdot u_{i,k} \\ &= m \cdot u_{C,i} + S(\pi) \cdot u_{F,i} \end{aligned}$$

Note that the total slack is the difference between $S(\pi)$ and $\sum_{T_j \in P(t_o)} u_{j,a_o(j)}$. Hence, the following is a lower bound for $\sum_{T_j \in P(t_o)} u_{j,a_o(j)}$.

$$S(\pi) - m \cdot u_{C,i} - S(\pi) \cdot u_{F,i}$$

Combining these two bounds gives

$$\begin{aligned} \sum_{T_j \in \tau_{-i}} u_{C,j} &> S(\pi) - u_{C,i} - M_{\pi,\tau} \\ \Rightarrow U_{cpu}(\tau) &> S(\pi) - M_{\pi,\tau} \ , \end{aligned}$$

which contradicts the condition of the theorem. ■

The previous utilization bound for r-EDF-scheduling on uniform multiprocessors [4], [5] is an immediate corollary to Theorem 1 above.

Corollary 1: Any periodic task set τ satisfying

$$U_{sum}(\tau) \leq S(\pi) - (m-1)u_{max}(\tau) \qquad (3)$$

will meet all its deadlines when scheduled on uniform multiprocessor π using r-EDF.

Proof: Assume the above condition holds. Because execution is not divided into e_{cpu} and e_{fixed}, we have to set $u_i = e_i/p_i$ to its maximum value in order to ensure all tasks will meet their deadlines. For all tasks T_i, the maximum value of the total utilization on any processor is $u_{i,1} = u_{C,i} + s_1 u_{F,i}$. Thus, $U_{sum}(\tau) = U_{cpu}(\tau) + s_1 \sum_{j=1}^{n} u_{F,j}$ and $u_{max}(\tau) = \max_{1 \leq j \leq n}\{u_{C,j} + s_1 \cdot u_{F,j}\}$. Substituting these identities into Condition 3 gives the following upper bound for $U_{cpu}(\tau)$.

$$S(\pi) - \max_{1 \leq i \leq n} \{(m-1)u_{C,i} + (m-1)s_1 u_{F,i} + s_1 \sum_{j=1}^{n} u_{F,j}\}$$

Combining the two s_1 terms gives the following bound.

$$S(\pi) - \max_{1 \leq i \leq n} \{(m-1)u_{C,i} + ms_1 u_{F,i} + s_1 \sum_{T_j \in \tau_{-i}} u_{F,j}\}$$

$$\leq S(\pi) - \max_{1 \leq i \leq n} \{(m-1)u_{C,i} + S(\pi)u_{F,i} + MP_{\pi,\tau_{-i}}\}.$$

The final inequality is the condition of Theorem 1. ■

In the last step, the bound increases by at least

$$(m \cdot s_1 - S(\pi))u_{F,j} + s_1 \sum_{T_j \in \tau_{-i}} u_{F,j} - MP_{\pi,\tau_{-i}} \ ,$$

and perhaps more if the index i associated with the maximum values changes. If s_1 is significantly faster than the other processors of π, this could be a significant savings. For example, if $\pi = [2, 1, 1, 1]$, then the coefficient of $u_{F,i}$ is reduced from 8 to 5. It is hard to determine how much the entire bound changes because $MP_{\pi,\tau_{-i}}$ is difficult to compute. The next section explores methods of calculating this value.

IV. Calculating the Utilization Bound

Unfortunately, in Theorem 2 below, we prove that finding the exact bound presented in Theorem 1 is an NP-hard problem. The NP-hardness stems from finding the packing that results in the largest value for $\sum_{T_j \in \tau_{-i}} s_{a(j)} \cdot u_{F,j}$ among all possible packings. Therefore, we need to employ approximation strategies to find an upper bound for this expression.

First, we prove that finding $MP_{\pi,\tau_{-i}}$ is NP-hard by proving the following decision problem is NP-complete.

The mixed task penalty problem: Given a uniform multiprocessor $\pi = [s_1, s_2, \ldots, s_m]$, a periodic or sporadic task set $\tau = \{T_1, T_2, \ldots, T_n\}$, and a number K, does there exist an assignment $a : \tau \rightarrow \pi$ of tasks to processors such that $\sum_{i=1}^{n} s_{a(i)} \cdot u_{F,i} \geq K$?

Theorem 2: The mixed task penalty problem is NP-complete.

Proof: This problem is clearly in NP. We will show the bin packing problem is reducible to the mixed task penalty problem. The bin packing problem is stated as follows.

Given a positive integer M, a finite set I of ℓ items, where each $x_i \in I$ has a size $w(x_i)$ such that $0 < w(x_i) \leq 1$, can all the items in I be placed into M unit-sized bins so that to total sized of the items in each bin is at most 1?

Given any bin packing instance we let π be comprised of M speed-1 processors and let $|\tau| = \ell$ and for each $x_i \in I$, the associated task $T_i \in \tau$ has fixed utilization $u_{F,i} = w(x_i)$ and CPU utilization $u_{C,i} = 0$. Finally, we let $K = \sum_{i=1}^{\ell} w(x_i)$. Then the maximum sum in the mixed task penalty problem is K if and only if all the items in I can fit into M bins. ■

Even though the mixed task penalty problem is NP-complete, we can find $M_{\pi,\tau}$ (which requires finding $MP_{\pi,\tau}$) using an integer linear program (ILP).

A. Finding $M_{\pi,\tau}$ Using an ILP

The ILP has $2 \times m \times n$ integer variables, $x_{i,j}$ and $y_{i,j}$, where $1 \leq i \leq n$ and $1 \leq j \leq m$. The variable $x_{i,j}$ indicates T_i is assigned to processor s_j. The one exception is if $y_{i,j}$ also equals 1. In this case, T_i is the task that maximizes $M_{\pi,\tau}$. Below, we present the constraints for the ILP.

The first constraint ensures none of the processors are over filled. For each processor s_j, we have the following constraint.

$$\sum_{i=1}^{n}(x_{i,j} - y_{i,j})u_{i,j} \leq s_j$$

The remaining constraints ensure the values of $x_{i,j}$ and $y_{i,j}$ are valid. The first constraint below ensures each task is assigned to only one processor. The second constraint ensures only one task is *not* assigned to any processor. The third constraint ensures $y_{i,j}$ removes a task from the processor it was assigned to. Note that even though the first two constraints are not set equal to 1, their values will both be 1, as making these values as large as possible increases the objective function.

$$\sum_{j=1}^{m} x_{i,j} \leq 1$$

$$\sum_{i=1}^{i}\sum_{j=1}^{m} y_{i,j} \leq 1$$

$$\sum_{j=1}^{m}(x_{i,j} - y_{i,j}) \geq 0$$

Finally, we maximize the objective function.

$$\sum_{i=1}^{n}\sum_{j=1}^{m}[s_j u_{F,i}(x_{i,j} - y_{i,j}) + ((m-1)u_{C,i} + S(\pi)u_{F,i})y_{i,j}]$$

If the ILP is able to find a solution, it will be the maximum possible value of $(m-1)u_{C,i} + S(\pi)u_{F,i} + MP_{\pi,\tau_{-i}}$ – i.e., the ILP objective function will equal $M_{\pi,\tau}$.

Unfortunately, ILPs can be unstable and a solution might not be found even if one exists. If this occurs, the ILP can be relaxed into an LP. The LP constraints and objective function would be the same. However, the variables x and y might take on fractional values. If this happens, it can be viewed as allowing jobs generated by the given task to migrate among processors. Below, we discuss another method for bounding $M_{\pi,\tau}$ by allowing jobs to migrate.

B. Bounding $M_{\pi,\tau}$ in Polynomial Time

The mixed task penalty problem is similar to the 0-1 knapsack problem, in which items are given values and weights and we want to select items to place into a knapsack that so that a weight limit is not exceeded and the value is maximized. The difference is that we have multiple knapsacks and the values and weights of the items vary depending on which knapsack we use – if task T_i is assigned to processor s_j, we give T_i a value of $s_j \cdot u_{F,i}$ and a weight of $u_{i,j}$.

Even so, we can use knowledge about the knapsack problem to find an upper bound for $MP_{\pi,\tau}$. Specifically, we know that the fractional knapsack problem, which allows fractions of items to be selected, can be solved in polynomial time using a greedy algorithm. The greedy algorithm sorts the items by the ratio of value to weight and selects the item with the largest ratio to put in the knapsack repeatedly until that item won't fit, at which point the fraction of the item that makes the total weight equal to the limit is selected.

The greedy solution to the fractional knapsack problem cannot be directly applied to the mixed task penalty problem because, once again, the value and the weight both vary depending on which processor the job is assigned to. However, we can use this idea to create the algorithm $fracM_{\pi,\tau}$, which is shown in Figure 1. Given an assignment of tasks to processors, $a(.)$, let $value(a)$ be the sum of $s_{a(i)}u_{F,j}$ over all tasks. Let $MaxSum_{\pi,\tau}$ be the maximum value of any assignment when

tasks can be assigned to multiple processors. The algorithm $fracM_{\pi,\tau}$ finds $MaxSum_{\pi,\tau}$ by using the fractional knapsack technique on one processor at a time, always working with the fastest processor that has remaining slack.

We first note that, even though the value and weight of tasks varies between processors, we can sort tasks once by $u_{F,i}/u_{C,i}$ to get the proper ratio of value to weight.

Lemma 1: Let T_i and T_j be any two tasks. Then, for any processor s_k, we have $\frac{s_k u_{F,i}}{u_{i,k}} \leq \frac{s_k u_{F,j}}{u_{j,k}}$ if and only if $\frac{u_{F,i}}{u_{C,i}} \leq \frac{u_{F,j}}{u_{C,j}}$.

Proof:

$$\begin{aligned} & \frac{u_{F,i}}{u_{C,i}} \leq \frac{u_{F,j}}{u_{C,j}} \\ \Leftrightarrow & \frac{u_{C,i}}{u_{F,i}} \geq \frac{u_{C,j}}{u_{F,j}} \\ \Leftrightarrow & \frac{u_{C,i} + s_k u_{F,i}}{u_{F,i}} \geq \frac{u_{C,j} + s_k u_{F,j}}{u_{F,j}} \\ \Leftrightarrow & \frac{u_{F,i}}{u_{i,k}} \leq \frac{u_{F,j}}{u_{j,k}} \end{aligned}$$

■

Algorithm $fracM_{\pi,\tau}$ sorts tasks by the ratio of fixed utilization to CPU utilization (lines 2 through 4). It then uses the fractional knapsack solution to assign tasks to processors in decreasing order of processor speed (lines 5 through 17). If a task is selected that does not fit on the current processor (i.e., $u_{i,j} > slack_j$), it divides the task into two pieces and assigns one piece to the current processor and the remaining piece is added to τ (lines 10 through 14). Let αT_i denote a task with CPU and fixed utilization equal to $\alpha u_{C,i}$ and $\alpha u_{F,i}$, respectively. Specifically, if $u_{i,j} > slack_j$ we let $\alpha = slack_j/u_{i,j}$ and assign αT_i to s_j and add $(1-\alpha)T_i$ to τ to be assigned to the next processor. This reduces $slack_j$ to 0 so $fracM_{\pi,\tau}$ begins assigning tasks to s_{j+1}. Note that splitting tasks this way will have no effect on the value of $r(i)$ used to determine the order in which tasks are added to processors. Finally, the algorithm $fracM_{\pi,\tau}$ returns the sum of $s_{a(i)}u_{F,i}$ that results from this assignment. Below, we prove that $fracM_{\pi,\tau}$ returns $MaxSum_{\pi,\tau}$.

$fracM_{\pi,\tau}(\pi,\tau)$

1. $MaxSum \leftarrow 0$
2. for $i \leftarrow 1$ to n do
3. $\quad r(i) \leftarrow u_{F,i}/u_{C,i}$
4. sort τ in decreasing order by $r(i)$
5. for $j \leftarrow 1$ to m
6. $\quad slack \leftarrow s_j$
7. $\quad$ while $slack > 0$ and $\tau \neq \emptyset$
8. $\quad\quad T_i \leftarrow$ task with maximum value of $r(i)$
9. $\quad\quad$ Remove T_i from τ
10. $\quad\quad$ If $u_{i,j} > slack$
11. $\quad\quad\quad \alpha \leftarrow slack/u_{i,j}$
12. $\quad\quad\quad T'_i \leftarrow (u'_{C,i}, u'_{F,i}) \leftarrow (1-\alpha)(u_{C,i}, u_{F,i})$
13. $\quad\quad\quad$ Add T'_i to τ
14. $\quad\quad$ else
15. $\quad\quad\quad alpha \leftarrow 1$
16. $\quad\quad slack \leftarrow slack - \alpha u_{i,j}$
17. $\quad\quad MaxSum \leftarrow MaxSum + \alpha s_j u_{F,i}$
18. return $MaxSum$

Fig. 1. Algorithm $fracM_{\pi,\tau}$

Theorem 3: Let $\tau = \{T_1, T_2, \ldots, T_n\}$ be any periodic task set and $\pi = [s_1, s_2, \ldots, s_m]$ be a uniform multiprocessor. Assume $a(.)$ is an assignment of tasks τ to processors of π in which migration of tasks is permitted. If $s_{a(i)} \geq s_{a(j)}$ whenever $u_{F,i}/u_{C,i} > u_{F,j}/u_{C,j}$ then $value(a) = MaxSum_{\pi,\tau}$.

Proof: (By contradiction.) Assume $u_{F,i}/u_{C,i} > u_{F,j}/u_{C,j}$ and $s_{a(i)} = \ell$ and $s_{a(j)} = k$, where $s_k > s_\ell$.

Define α_i and α_j as follows.

$$\alpha_i = \min\left\{1, \frac{u_{i,k}}{u_{j,k}}\right\}$$
$$\alpha_j = \frac{\alpha_i u_{i,k}}{u_{j,k}}$$

By the above equation, we know that

$$\alpha_i u_{i,k} = \alpha_j u_{j,k}. \quad (4)$$

Hence, $\alpha_i T_i$ will take the same amount of slack from s_k as $\alpha_j T_j$. Let $\hat{a}(.)$ be the assignment that results from moving $\alpha_i T_i$ to s_k and $\alpha_j T_j$ to s_ℓ. We wish to prove $value(a) < value(\hat{a})$. We

begin by proving the following properties.

Claim 1: $\alpha_i u_{F,i} > \alpha_j u_{F,j}$

This follows directly from Lemma 1 and Equation 4. Multiplying these together proves the claim.

Claim 2: $\alpha_i u_{i,\ell} < \alpha_j u_{j,\ell}$. In particular,

$$\alpha_j u_{j,\ell} - \alpha_i u_{i,\ell} = (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell). \quad (5)$$

Recall $u_{i,k} = u_{C,i} + s_k u_{F,i}$. Therefore,

$$\begin{aligned}
&\alpha_j u_{j,\ell} - \alpha_i u_{i,\ell} = \\
&\alpha_j(u_{C,j} + s_\ell u_{F,j}) - \alpha_i(u_{C,i} + s_\ell u_{F,i}) \\
&\quad +(\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) \\
&\quad -(\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) = \\
&\alpha_i(u_{C,i} + s_k u_{F,i}) - \alpha_j(u_{C,j} + s_k u_{F,j}) \\
&\quad +(\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) = \\
&(\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell).
\end{aligned}$$

The last step follows from Equation 4. By assumption $s_k > s_\ell$, and, by Claim 1, $\alpha_i u_{F,i} > \alpha_k u_{F,k}$. Therefore, the final result is positive. This proves the claim.

Claim 2 tells us that moving T_j to s_ℓ will increase the total utilization of tasks assigned to s_ℓ. In the worst case $slack_\ell = 0$, and swapping T_i and T_j forces some task(s) to be removed from s_ℓ. Assume some portion, α_q of T_q is forced off of s_ℓ, where $u_{F,q}/u_{C,q} \leq u_{F,j}/u_{C,j}$. Specifically,

$$\alpha_q = \frac{\alpha_k u_{k,\ell} - \alpha_i u_{i,\ell}}{u_{q,\ell}}. \quad (6)$$

Then moving $\alpha_q T_q$ to a slower processor ensures s_ℓ will not be overfilled as a result of swapping T_i and T_j. (Note that T_q may be T_k. Also, if T_k forces multiple tasks to be removed from s_ℓ, the proof below will have to be modified slightly. We assume a single task is removed from s_ℓ for readability.) Hence,

$$\begin{aligned}
&\sum_{h=1}^{n} s_{\hat{a}(h)} u_{F,h} - \sum_{h=1}^{n} s_{a(h)} u_{F,h} \\
&\geq \alpha_i u_{F,i}(s_k - s_\ell) + \alpha_j u_{F,j}(s_\ell - s_k) \\
&\quad -\alpha_q u_{F,q} s_\ell \\
&= (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) \\
&\quad -\frac{\alpha_j u_{j,\ell} - \alpha_i u_{i,\ell}}{u_{q,\ell}} u_{F,q} s_\ell.
\end{aligned}$$

By Claim 2, this is equal to

$$\begin{aligned}
&(\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell) \\
&\quad -\frac{(\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell)}{u_{q,\ell}} u_{F,q} s_\ell \\
&= (\alpha_i u_{F,i} - \alpha_j u_{F,j})(s_k - s_\ell)(1 - \frac{s_\ell u_{F,q}}{u_{q,\ell}}).
\end{aligned}$$

Each of these terms is positive. Therefore, $value(\hat{a}) > value(a)$. Thus, if any other order is used, $value(a) \neq MaxSum_{\pi,\tau}$. ■

Because packing without migration is a special case of packing with migration, algorithm $fracM_{\pi,\tau}$ finds an upper bound for $MP_{\pi,\tau}$. In order to find $M_{\pi,\tau}$, we loop through all tasks T_i in τ – each time calling $fracM_{\pi,\tau}(\pi, tau_{-i})$and calculating $(m-1)u_{C,i} + S(\pi) \cdot u_{F,i} + MaxSum_{\pi,\tau}$. After looping through all tasks, we return the maximum value.

Even replacing this upper bound for $M_{\pi,\tau}$ into Theorem 1 will improve the analysis compared to the previous test, stated in Corollary 1. In the proof of the corollary, we see that $u_{F,i}$ must be multiplied by s_1 for all $i = 1, 2, \ldots, n$ if the previous test is used. By contrast, algorithm $fracM_{\pi,\tau}$ is able to multiply some of the fixed priorities by smaller values.

1) Running Time: Lines 2 through 3 of algorithm $fracM_{\pi,\tau}$ take $\Theta(n)$ time. Line 4 takes $O(nlgn)$ time, assuming a reasonable sorting algorithm is used. Lines 5 and 6 are executed m times. Lines 7 through 17 are executed once for each task plus once for each fraction of a task that is added to τ. At most $(m-1)$ tasks are added back to τ. Therefore lines 7 through 17 are executed $\Theta(m+n)$ times,

giving a total running time for $fracM_{\pi,\tau}$ of $O(n \log n + m)$. In general, we assume $n > m$, which gives the running time of $O(n \log n)$. Algorithm $fracM_{\pi,\tau}$ is executed n times, giving a total running time of $O(n^2 \log n)$.

V. Conclusion and Future Work

This paper has presented a variation of the task model that differentiates between execution that scales with CPU speed and execution that does not. We have shown that using this model can improve a known utilization bound for scheduling on uniform multiprocessors — namely the bound for EDF scheduling with restricted migration. Unfortunately, the bound involves using an NP-complete problem. Hence, we have shown methods of finding this bound using integer linear programming. We have also shown a method to approximate this bound in $O(n^2 \log n)$ time by calculating the expression when jobs are allowed to migrate.

In the future, we hope to apply this task model to EDF using other migration strategies and also to RM scheduling. We also plan to determine the competitive ratio of $MaxSum_{\pi,\tau}$, the output of algorithm $fracM_{\pi,\tau}$, and improve the estimate if the ratio is too large.

References

[1] Sanjoy K. Baruah and John Carpenter. Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. *Journal of Embedded Computing*, 1(2), 2004.

[2] Enrico Bini, Giorgio Buttazzo, and Giuseppe Lipari. Speed Modulation in Energy-Aware Real-Time Systems. In *ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems*, pages 3–10, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Michael Dertouzos and Aloysius K. Mok. Multiprocessor scheduling in a hard real-time environment. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, 1989.

[4] Shelby Funk. *EDF Scheduling on Heterogeneous Multiprocessors*. PhD thesis, CS Department, UNC Chapel Hill, 2004.

[5] Shelby Funk and Sanjoy K. Baruah. Restricting EDF migration on uniform multiprocessors. Technical Report TR03-035, CS Department, UNC Chapel Hill, 2003.

[6] Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[7] Mauro Marinoni and Giorgio Buttazzo. Elastic DVS Management in Processors With Discrete Voltage/Frequency Modes. *IEEE Transactions on Industrial Informatics*, 3:51–62, 2007.

[8] Aloysius K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[9] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, pages 40–51. IEEE Computer Society, 2003.