Optimizing Preemption-Overhead Accounting in Multiprocessor Real-Time Systems [*]

Bryan C. Ward
Dept. of Computer Science
University of North Carolina at Chapel Hill

Abhilash Thekkilakattil
Div. of Software Engineering
Mälardalen University,
Västerås, Sweden

James H. Anderson
Dept. of Computer Science
University of North Carolina at Chapel Hill

ABSTRACT

There exist two general techniques to account for preemption-related overheads on multiprocessors. This paper presents a new preemption-related overhead-accounting technique, called analytical redistribution of preemption overheads (ARPO)*, which integrates the two previous techniques to minimize preemption-overhead-related utilization loss. ARPO is applicable under any* job-level fixed priority (JLFP) *preemptive scheduler, as well as some limited-preemption schedulers. ARPO is evaluated in a new experimental-design framework for overhead-aware schedulability studies that addresses unrealistic simplifying assumptions made in previous studies, and is shown to improve real-time schedulability.*

1. INTRODUCTION

The widespread availability of multicore processors has motivated their adoption in real-time systems. As a result, numerous multiprocessor scheduling algorithms have been proposed. Some of these algorithms, such as PFair [4], and RUN [26], have been proven optimal[1] under common analysis assumptions. However, empirical results have demonstrated that many of these algorithms are often impractical due to high *runtime overheads*, despite their optimality (under overhead-oblivious analysis assumptions) [9]. Overheads are therefore an important consideration in the design, development, and analysis of multiprocessor scheduling algorithms, as they are a significant source of pessimism in the associated schedulability analyses [6].

[*]Work supported by NSF grants CNS 1016954, CNS 1115284, CNS 1218693, and CNS 1239135, AFOSR grant FA9550-14-1-0161, and a grant from General Motors Corp. The first author was also supported by an NSF graduate research fellowship. The second author was supported by the Swedish Research Council project CONTESSE (2010-4276).

[1]An optimal scheduler guarantees all deadlines are satisfied for any feasible task system, *i.e.*, one for which a correct schedule exists.

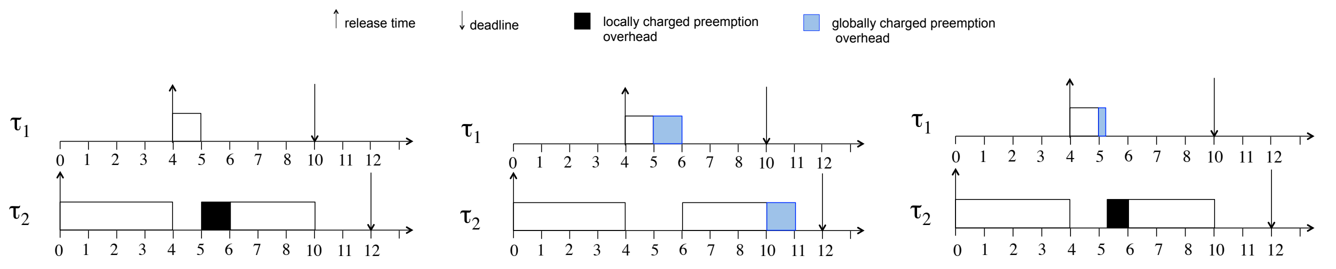
RTNS 2014 , October 8 - 10 2014, Versailles, France
Copyright is held by the owner/authors(s). Publication rights licensed to ACM.
ACM CM 978-1-4503-2727-5/14/10 ...$15.00.
http://dx.doi.org/10.1145/2659787.2659811.

Arguably one of the most significant overhead sources in multiprocessor real-time systems, and indeed the overhead source that most affects the aforementioned optimal schedulers, are *preemption-related overheads*. There are several preemption-related overheads such as scheduling, context switching, pipeline delays, and most significantly *cache-related preemption and migration delays* (*CPMDs*). CPMDs model the temporal overhead associated with the loss of cache affinity as a result of a preemption or migration to another processor, and are often the largest and most analytically complex source of pessimism owing to multiple cache levels on multicore chips. In the remainder of this paper, all of these overhead sources are accounted for generally as preemption-related overheads.

Consider the example depicted in Fig. 1 (a) in which task τ_2 is preempted by τ_1 at time $t = 4$. When τ_2 resumes at $t = 5$, it may have lost cache affinity and therefore may execute for longer than it would have if it was not preempted. Because τ_2 may have evicted all of τ_1's *working set*, or the set of cache lines with which τ_1 had affinity, the CPMD cost is the *working-set size* (*WSS*) multiplied by the cache-miss cost. Note that in Fig. 1 (a), this overhead is modeled as occurring between time $t = 5$ and $t = 6$, though in practice, τ_2 executes during this time—the overhead simply models the increase in execution time of τ_2 on account of the preemption overhead.

To ensure that deadlines are not missed on account of preemption-related overheads, such overheads must be incorporated into schedulability analysis. To do so, intuitively, execution times must be *inflated* to account for the effects of overheads. The inflated task system may then be analyzed using overhead-oblivious schedulability tests, consequently guaranteeing the absence of deadline misses even in the presence of preemption-related overheads. There exist two general techniques for inflating task execution times to account for such overheads, which herein we call *task-* and *preemption-centric accounting*, respectively [13, 21].

Under task-centric accounting [13, 21], each task's execution cost is inflated by its worst-case preemption cost multiplied by an upper bound on the number of times it may be preempted. For example, in Fig. 1 (a), the preemption cost of one time unit would be included in the execution time of τ_2. However, when more than two tasks are considered, a bound on the number of times each task can be preempted, and thus how many times the preemption cost is charged, is usually a gross upper bound on the number of preemptions observed in practice. The magnitude of such pessimism is further amplified in globally scheduled multiprocessor systems (*i.e.*, systems in which tasks are scheduled on multiple CPUs


release time
deadline
locally charged preemption overhead
globally charged preemption overhead
τ1
τ2
0 1 2 3 4 5 6 7 8 10 11 12

(a) Task-centric overhead accounting. (b) preemption-centric overhead accounting. (c) Proposed ARPO overhead accounting.

Figure 1: Example task system demonstrating different preemption-overhead accounting techniques.

from a single ready queue), which can support higher task counts (and hence generally have higher preemption-frequency estimates).

Perhaps motivated by this deficiency of task-centric accounting, preemption-centric overhead accounting [13, 21] uses an alternative charging scheme that is not based on bounds on the number of preemptions. Fundamentally, preemption-centric accounting is based on the idea of charging the overhead of preemptions to the *relinquishing task*,[2] *i.e.*, task that completes and allows a previously preempted task to resume execution, instead of the preempted task, as in task-centric accounting. In the example in Fig. 1 (b), under preemption-centric accounting, the preemption cost of one time unit would instead be included in the execution cost of task τ_1 instead of τ_2.

There are also trade-offs associated with preemption-centric overhead accounting. For example, consider a two-task system with one task having a very small WSS, and the other with a very large WSS. Under preemption-centric accounting, the small-WSS task would be charged for the large-WSS task's preemption-overhead cost, which may significantly increase the effective utilization of the small-WSS task, and therefore cause the system to be unschedulable. Importantly, based on these observations, neither task- nor preemption-centric overhead accounting strictly dominates the other.

Contributions. In this paper, we present a hybrid preemption-overhead accounting scheme that we call *analytical redistribution of preemption overhead* (*ARPO*). ARPO incorporates ideas from both task- and preemption-centric accounting through a *linear-program* optimization. Intuitively, ARPO allows for the preemption overhead to be accounted for in part by the preempted task and in part by the relinquishing task, as depicted in Fig. 1 (c). Linear programming can be used to determine what fraction of each preemption overhead should be charged to the relinquishing task instead of the preempted task. This allows the linear-program solver to analytically redistribute preemption overheads among the tasks to balance different sources of analysis pessimism and thereby minimize utilization inflation due to preemption-related overheads.

We also extend the experimental design of previous overhead-aware schedulability studies (*e.g.* [6, 9]) to consider more realistic task systems. To our knowledge, no previous overhead-aware multiprocessor schedulability study has considered task systems with varying WSSs. We address such issues in a schedulability study of ARPO and have devised an improved experimental-design framework, which we claim more accurately reflects characteristics of real systems.

Organization. In Sec. 2, we describe related work and formalize our task model. In Secs. 3 and 4, we describe ARPO in the context of fully preemptive and limited-preemption schedulers, respectively. In Sec. 5, we discuss potential extensions and applications of our work. In Sec. 6, we present our new experimental-design framework for overhead-aware schedulability studies, and evaluate ARPO using it. Finally, we conclude in Sec. 7.

2. BACKGROUND

In this section, we describe our system model and assumptions, describe relevant background material, and place our work in the context of previous research.

2.1 Related Work

Previous work has shown that, depending on the platform, CPMDs can be over 1 ms, and can increase execution times by up to 33% [6, 10]. Furthermore, preemptions also increase bus contention [7]. Consequently, there has been a large body of work to reduce preemption-overhead-related utilization loss. Specifically, two principal approaches have been taken: (i) improved accounting techniques of preemption-related overheads, and (ii) alternative scheduling policies that give rise to fewer preemptions and thereby lesser preemption-related overheads using existing accounting techniques.

To apply classic overhead-oblivious schedulability analysis (*e.g.* [1, 3, 5, 18, 25]), tasks' worst-case execution times (WCETs) are inflated to account for overheads. For example, Busquets-Mataix *et al.* [12] presented a preemption-centric accounting technique that incorporates the effects of instruction caches. Many others (*e.g.*, [27, 28]) have applied similar preemption-centric methodology to preemption-related overheads. Lee *et al.* [19] proposed a task-centric preemption-overhead accounting technique that calculates the worst-case preemption overhead at each point in the task code, and uses a mixed integer linear program to bound each tasks' worst-case response time. While all of the previously mentioned papers focus on uniprocessors, recent work (*e.g.*, [31]) has focused on multi-core schedulability analysis following either task- or preemption- centric strategies.

[2]This overhead is sometimes described as being charged to the preempting task in uniprocessor systems; however, in such systems, the preempting task is also the relinquishing task. In multiprocessor systems, the preempting and relinquishing tasks may be different.

Others have addressed preemption-related utilization loss by developing limited-preemption schedulers, which reduce the number and cost of preemptions. Burns *et al.* [11] introduced co-operative scheduling, in which tasks cooperate to reduce preemptions. Baruah [2] introduced a limited-preemption scheduling technique called *floating non-preemptive-region scheduling*, which delays preemptions for a bounded duration. There exist many other techniques to reduce preemptions such as assigning preemption thresholds—a detailed survey of limited-preemption scheduling on uniprocessors is available in [14]. The advantage of limiting preemptions is that, since the point of preemption is known in advance, preemption overheads can be bounded less pessimistically. Bertogna *et al.* [7] presented a method to optimally place preemption points in the task code for uniprocessors, and added the cost of preemptions to the preempted tasks' WCETs. Marinho *et al.* [24] presented schedulability analysis for limited-preemption multiprocessor global fixed-priority scheduling that extended their previous work [15]. Thekkilakattil *et al.* [29] presented schedulability analysis for global limited-preemption *earliest-deadline-first* (EDF) scheduling, and quantified the *cost* of limiting preemptions using resource augmentation.

We build upon both preemption-overhead accounting techniques and propose a hybrid accounting technique that is applicable under both preemptive and limited-preemption scheduling.

2.2 System Model

We consider a system of n sporadic real-time *tasks* $\Gamma = \{\tau_1, \tau_2, \ldots \tau_n\}$ that execute on m globally scheduled processors. Each τ_i is a potentially infinite sequence of *jobs*, and is characterized by a *minimum job inter-arrival time* T_i, a *relative deadline* D_i (for simplicity of presentation, we assume implicit deadlines, *i.e.*, $D_i = T_i$), and an *execution time* (without overheads), C_i. The utilization of τ_i is $u_i = C_i/T_i$ and the total task-system utilization is $U = \sum_{i=1}^{n} u_i$.

We make the general assumption of a *job-level fixed-priority* (*JLFP*) *scheduler*, *i.e.*, each job's priority is constant, but jobs of the same task may have different priorities. For example, both EDF and *fixed-priority* (FP) scheduling are JLFP. Since job priorities affect preemption relations, we introduce the following notation. We let $X_i(\tau_j)$ upper bound the number of times a job of τ_i can be preempted by τ_j. For example, under FP scheduling, if τ_j is of higher priority than τ_i, then $X_i(\tau_j) = \lceil \frac{T_i}{T_j} \rceil$, and $X_j(\tau_i) = 0$. The maximum overhead incurred as a result of the preemption of a task τ_i is denoted by $\Delta_i^{\max}$ and the maximum preemption overhead incurred in the task set is given by $\Delta^{\max} = \max_{\tau_i \in \Gamma}\{\Delta_i^{\max}\}$.

To account for preemption-related overheads, we derive an inflated task set Γ', which is assumed to be overhead-free, but is at least as hard to schedule as Γ with overheads. Formally, Γ' is a safe hard-real-time approximation, which is defined as follows.

DEF. 1. ***(Def. 3.1 from [9])*** *A task set* Γ' *is a* safe hard real-time approximation *of* Γ *if and only if the following assertion holds: If there exists a legal collection of jobs such that a task in* Γ *misses a deadline in the presence of overheads, then there exists a legal collection of jobs such that a task in* Γ' *misses a deadline in the absence of overheads.*

For notational clarity, we use the prime symbol (as in Γ') to denote taskset parameters in the inflated taskset Γ', *i.e.*, C_i' is the inflated execution cost of τ_i. We assume that for each $\tau_i' \in \Gamma'$, $D_i' = D_i$, and $T_i' = T_i$. Determining smaller but safe values of C_i' is the subject of this work.

2.3 Preemption-Overhead Accounting Techniques

Using these definitions, we formally review task- and preemption-centric preemption-overhead accounting, before building upon them in Secs. 3 and 4. We will also illustrate each of these approaches on the example task system described in Tbl. 1. In these examples, for simplicity we consider rate-monotonic fixed-priority scheduling, *i.e.*, the shorter-period tasks have higher priority. Consequently, $X_i(\tau_j)$ is computed as described previously.

Example 1. Consider the example task set in Tbl. 1. Under task-centric preemption-overhead accounting Γ' is computed as follows. τ_1 is the highest priority and is therefore never preempted. Thus $C_1' = C_1 = 1$. Both τ_2 and τ_3 however can be preempted, and thus their execution costs are inflated as follows: $C_2' = C_2 + \left(\left\lceil \frac{T_2}{T_1} \right\rceil \times \Delta_2^{\max}\right) = 4$, and $C_3' = C_3 + \left(\left\lceil \frac{T_3}{T_1} \right\rceil \times \Delta_3^{\max} + \left\lceil \frac{T_3}{T_2} \right\rceil \times \Delta_3^{\max}\right) = 12$. Therefore, the total utilization of the resulting task set Γ' is $U' = 1/6 + 4/8 + 12/12 \approx 1.66$.

Task-centric preemption-overhead accounting, while a safe approximation (recall Def. 1), is subject to two major sources of analysis pessimism. First, $X_i(\tau_j)$ is an *upper bound* on the number of preemptions that may occur, and depending upon the scheduler or the actual task set, may not be very tight. This is particularly true on globally scheduled multiprocessor systems where the number of tasks can be large and the release of a high-priority job only preempts one of up to m running tasks. The second source of analysis pessimism stems from how every task is assumed to be preempted by every possible higher-priority job. When a job is released, it may cause at most one preemption, but that preemption is being accounted for by every task, instead of only one task. This second source of pessimism motivates the design of preemption-centric accounting.

Preemption-centric accounting. As shown in Fig. 1 (b), under preemption-centric overhead accounting, the relinquishing task effectively "pays for" for the preemption overhead of the task that resumes upon its completion. To do so, the execution time of each task is inflated to account for the worst-case preemption overhead it may induce. This can be modeled as follows

$$C_i' = C_i + \Delta^{\max}. \tag{1}$$

($\Delta^{\max}$ is used instead of $\Delta_i^{\max}$, as τ_i is "paying for" the overhead of another task.) Intuitively, this is a safe approximation because it is accounting for the overhead at a higher priority.[3]

Task-centric accounting. As depicted in Fig. 1 (a), under task-centric preemption-overhead accounting, the execution time of every task is inflated to account for every possible preemption that may occur in a valid schedule of

[3]An additional optimization is possible based on the observation that under preemption-centric accounting, a task need not pay for its own preemption. Thus, $\max_{\tau_j \in \Gamma \setminus \{\tau_i\}} \Delta_j^{\max}$ can be used instead of $\Delta^{\max}$.

τ_i	C_i	T_i	$\Delta_i^{\max}$
τ_1	1	6	0
τ_2	2	8	1
τ_3	4	12	2

Table 1: Example task set.

Γ [13, 21]. Therefore, since each task τ_j may preempt τ_i $X_i(\tau_j)$ times and each preemption has an overhead of at most $\Delta_i^{\max}$ time, C_i' can be computed as

$$C_i' = C_i + \sum_{\tau_j \in \Gamma} X_i(\tau_j)\Delta_i^{\max}. \quad (2)$$

Example 2. Consider again the example task system in Tbl. 1. The worst-case preemption overhead $\Delta^{\max} = 2$, is charged to each task. Therefore, $C_1' = C_1 + \Delta^{\max} = 3$, $C_2' = C_2 + \Delta^{\max} = 4$, $C_3' = C_3 + \Delta^{\max} = 6$, and the utilization of the resulting task set Γ' is $U' = 3/6 + 4/8 + 6/12 = 1.5$.

Note that preemption-centric accounting had a total utilization of 1.5, while task-centric accounting had a total utilization of 1.66. In this example (and indeed many systems, as will be seen in Sec. 6), preemption-centric accounting less pessimistically accounts for preemption overheads. However, it is still subject to analysis pessimism, most significantly due to the fact that every task has to "pay for" the largest preemption overhead in the system. When the WSSs and therefore CPMD overheads are highly variant from task to task, preemption-centric accounting can be very pessimistic.

Comparison. These two examples demonstrate that the overhead-accounting technique influences the task-set utilization and therefore schedulability since each technique is subject to different sources of analysis pessimism. Task-centric accounting is more pessimistic when the number of tasks is high, while preemption-centric accounting is more pessimistic when the WSSs are highly variant. This motivates our work, which is a hybrid of these two techniques that balances these pessimism sources.

3. ARPO—FULLY PREEMPTIVE MODEL

In this section, we present ARPO, a hybrid preemption-overhead accounting technique based on both task- and preemption-centric accounting. Under ARPO, Γ' is derived from Γ by charging a part of the preemption overhead to the relinquishing task and the rest to the preempted task's WCET. Such a charging scheme, combined with an optimization framework, allows for different sources of analysis pessimism to be balanced, which can improve task-set schedulability. A simple example that illustrates the intuition of ARPO is presented in Fig. 1 (c).

Under ARPO, the cost of preemption overheads is redistributed or balanced between local per-task (*i.e.*, task-centric) charges, and global (*i.e.*, preemption-centric) charges that all tasks must "pay." This balance is defined by the global charge G that is added to all tasks in the system. To ensure that every preemption overhead is fully accounted for, any remaining overhead not accounted for by the global charge G, must be accounted for locally. This is modeled by the following execution-cost inflation equation

$$C_i' = C_i + \sum_{\tau_j \in \Gamma} X_i(\tau_j)\max(0, \Delta_i^{max} - G) + G. \quad (3)$$

To better illustrate this idea, we will consider the two extremes. First, consider the case that $G = 0$. In this case, it can be trivially shown that (3) reduces to (2), *i.e.*, task-centric accounting. Similarly, if $G = \Delta^{\max}$, (3) reduces to (1), *i.e.*, preemption-centric accounting. ARPO also allows G to be set between these two extremes to produce a hybrid of task- and preemption-centric accounting that redistributes how overheads are accounted to reduce task-set utilization.

Example 3. Recall the example from Tbl. 1, and let $G = 1$. By (3), $C_1' = C_1 + G = 2$, $C_2' = C_2 + \left(\left\lceil \frac{T_2}{T_1} \right\rceil \max(0, \Delta_2^{\max} - G)\right) + G = 3$, $C_3' = C_3 + \left(\left\lceil \frac{T_3}{T_1} \right\rceil \max(0, \Delta_3^{\max} - G) + \left\lceil \frac{T_3}{T_2} \right\rceil \max(0, \Delta_3^{\max} - G)\right) + G = 9$. Consequently, the total utilization of the task set Γ' is $U' = 2/6 + 3/8 + 9/12 \approx 1.46$.

Note that under task- and preemption-centric overhead accounting, the utilization of the task set is 1.66 and 1.5, respectively. By using ARPO we further reduce the total task-set utilization to 1.46. This demonstrates how ARPO can balance different sources of pessimism to improve preemption-overhead accounting.

Before showing how to choose the global overhead charge G, we first fulfill our proof obligation to show that Γ' produced using (3) is a safe approximation of Γ (recall Def. 1).

THEOREM 1. *Γ' computed by (3) is a safe hard real-time approximation of Γ if $G \geq 0$.*

PROOF. For contradiction, assume that Γ' is not a safe approximation. Then in the presence of preemption overheads, a task $\tau_i \in \Gamma$ may miss a deadline but Γ' is schedulable, *i.e.*, no task in Γ' will ever miss a deadline assuming no overheads. This implies that in the schedule of Γ, which includes preemption overheads, there exists a preemption whose overhead is not entirely accounted for by Γ'. Without loss of generality, assume that the unaccounted for preemption is of a job of task τ_i, and this preemption has an overhead of $\Delta_i \leq \Delta_i^{\max}$.

Observe that $\sum_{\tau_j \in \Gamma} X_i(\tau_j)\max(0, \Delta_i^{\max}) \geq 0$. Thus, by (3), $\forall \tau_j' \in \Gamma'$, $C_j' \geq C_j + G$. Therefore, by (1), and the assumption that preemption-centric accounting results in a safe hard real-time approximation, G time is accounted towards every preemption. Thus, by the assumption that τ_i's preemption, Δ_i, is not entirely accounted for, $G < \Delta_i$.

For each preemption, G has been accounted for by the task that relinquished the processor to allow τ_i to execute. Therefore, $\max(0, \Delta_i - G) \leq \max(0, \Delta_i^{\max} - G)$ remains unaccounted for. By definition, τ_i can be preempted at most $\sum_{\tau_j \in \Gamma} X_i(\tau_j)$ times. By (3), C_i' includes $\max(0, \Delta_i^{\max} - G)$ overhead for every possible preemption. Therefore, since $G + \max(0, \Delta_i^{\max} - G) \geq \Delta_i$, Δ_i is totally accounted for. Contradiction. $\square$

After proving that ARPO is a safe approximation for any value of $G \geq 0$, and observing that ARPO can balance different sources of analysis pessimism to reduce utilization, the natural question is: what value of G is best? While there may be several possible optimization criteria, which are further discussed in Sec. 5, here we show how linear programming can be used to minimize the utilization of Γ'.

Optimization formulation. In the construction of our linear program, we identify a number of constants with respect to the linear program. Specifically, all original (non-inflated) task parameters, *e.g.*, T_i, D_i, C_i, u_i, $\Delta_i^{\max}$, and $X_i(\tau_j)$ are constants. We assume G, and $\forall \tau_i \in \Gamma$, C_i' are variables in the linear program. Additionally, we add another variable L_i for each task $\tau_i \in \Gamma$, which corresponds to the max term in the summation in (3) and models how much of the preemption overhead is charged locally by the preempted task. Using these constants and variables, we can reformulate (3) as a set of linear constraints with the objective to minimize U'.

First, we observe the $\max(0, \Delta_i^{\max} - G)$ is not a linear expression. However, using well-known techniques, we can model this term as follows.

Constraint Set 1. *The linear constraints corresponding to* $\max(0, \Delta_i^{\max} - G)$ *are given by*

$$\forall \tau_i \in \Gamma : L_i \geq \Delta_i^{max} - G,$$
$$\forall \tau_i \in \Gamma : L_i \geq 0.$$

Intuitively, if $\max(0, \Delta_i^{\max} - G) > 0$, then the first constraint will limit how L_i is set, otherwise the second constraint will ensure $L_i \geq 0$.

After developing Constraint Set 1, we can model the rest of (3) as follows.

Constraint Set 2. *The linear constraints corresponding to (3) are given by*

$$\forall \tau_i \in \Gamma : C_i' \geq C_i + \sum_{\tau_j \in \Gamma} X_i(\tau_j) L_i + G.$$

We note that for Constraint Set 2 to be linear, $X_i(\tau_j)$ must be a constant. However, in most common JLFP schedulers, such as EDF or FP scheduling, $X_i(\tau_j)$ is the ceiling of the two tasks' periods, $\lceil \frac{T_j}{T_i} \rceil$, if under FP τ_j has a higher priority than τ_i, or under EDF if τ_j has a shorter period than τ_i. Because task periods are assumed to be constants, $X_i(\tau_j)$ is also a constant.

Constraint Set 3. *The linear constraint corresponding to the assumption that G must be positive (as required by Thm. 1), is given by*

$$G \geq 0.$$

While Constraint Sets 1-3 are sufficient to correctly model (3), we can also add the following constraint as an optimization.

Constraint Set 4. *The following linear constraints ensure that (if possible) after inflation no per-task utilization overutilizes a single processor.*

$$\forall \tau_i \in \Gamma : u_i' = \frac{C_i'}{T_i'} \leq 1. \quad (4)$$

Constraint Set 4 demonstrates the flexibility and power of using linear programming to choose a global charge, G. The LP solver can minimize the total utilization of the inflated task system, subject to the constraint that each tasks' utilization is at most one (if any per-task utilization exceeds one, all of its deadlines may be missed, and thus the system is unschedulable).

We note that minimizing system utilization, subject to Constraint Set 4 does *not* necessarily imply an optimal setting of G for hard real-time multiprocessor systems, for which most schedulability tests are not utilization-based.[4] In other words, an alternative setting of G, which may increase system utilization, may result in a schedulable task system. We discuss this further in Sec. 5.

τ_i	C_i	T_i	b_i	$B_{i,k}$	$\Delta_{i,k}$
τ_1	1	5	1	1	0.00
τ_2	10	15	7	3.00, 0.75, 2.25, 0.75, 1.50, 0.75, 1.00	0.25, 1.00, 0.00, 0.50, 0.25, 0.25, 0.00

Table 2: Example limited-preemption task set.

4. ARPO—LIMITED-PREEMPTION MODEL

In this section, we describe the application of ARPO to limited-preemption schedulers. While there exist a number of different models for limited-preemption scheduling (described in Sec. 2), they are all motivated by the desire to reduce preemption-related overheads, and improve WCET analysis [2, 7]. By limiting the number of preemptions, for example, by enforcing that preemptions occur at a limited number of *preemption points* in the code, bounds on the number of preemptions can be improved. Furthermore, tighter bounds on the preemption costs can be calculated for each preemption point,[5] and can be incorporated more accurately into the analysis. For these reasons, it is interesting and important to study ARPO in the context of limited-preemption schedulers. Specifically, ARPO can redistribute preemption overheads among the preempted and relinquishing task differently than in the preemptive case, because the bounds on the number of times a task may be preempted are in many cases much tighter.

Limited-preemption task model. Before we present ARPO in the context of limited-preemption schedulers, we must first formalize our extended limited-preemption task model. Unless stated otherwise, our limited-preemption task model uses the same notation as the preemptive model described in Sec. 2. Each tasks' execution time is composed of a set of b_i non-preemptable blocks. The maximum execution time of the kth block is given by $B_{i,k}$. It follows that the overhead-oblivious WCET of a task τ_i is the sum of all of its

[4]In fact, we tried adding a constraint corresponding to the utilization-based schedulability condition in [17], and found the schedulability to be inferior to optimizing for utilization and applying more expensive non-utilization-based schedulability tests.

[5]Such bounds are derived through timing-analysis tools. Tighter bounds are made possible by exploiting knowledge about the state of the task at each preemption point, for example, that certain cache lines may not be accessed again before the completion of the job.

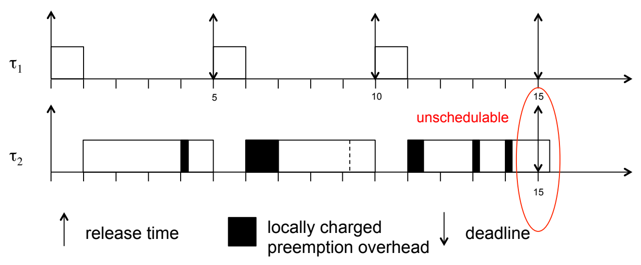


Figure 2: Illustration of task-centric preemption-overhead accounting applied to the limited-preemption task system in Tbl. 2.

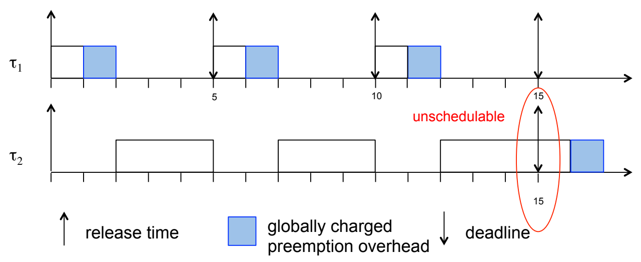


Figure 3: Illustration of the preemption-centric accounting applied to the limited-preemption task system in Tbl. 2.

non-preemptable blocks, $C_i = \sum_{k=1}^{b_i} B_{i,k}$. Under this model, preemptions can only occur at block boundaries. The worst-case overhead of a preemption after block $B_{i,k}$ is denoted by $\Delta_{i,k}$ (Δ_{i,b_i} is always 0).

Example 4. We demonstrate the limited-preemption scheduling model through the simple example task system shown in Tbl. 2, assumed to be scheduled by EDF on $m = 1$ processor. First, we apply task-centric preemption accounting to Γ, which is depicted in Fig. 2. Under limited-preemption scheduling, task-centric accounting is intuitive—the execution time of each task is inflated to account for the preemption overhead after each block. Thus, in the example task system C_2' is equal to $C_2 + \sum_{k=1}^{b_2} \Delta_{2,k} = 12.25$. The WCET of task τ_1' is equal to 1 since there is only a single block, which can never be preempted. Consequently, the total utilization of the task set Γ' is $U' = 1/5 + 12.25/15 \approx 1.02$. Note that this task system is not schedulable on a uniprocessor.

For the purpose of comparison, we briefly review preemption-centric accounting in the context of limited-preemptive scheduling.

Example 5. Under preemption-centric accounting, which is illustrated in Fig. 3, each task's execution cost is inflated by $\Delta^{\max}$, just as under a preemptive scheduler. In the example task set in Tbl. 2, the largest preemption cost is one time unit. Thus, $C_1' = C_1 + 1 = 2$ and $C_2' = C_2 + 1 = 11$, and the resulting utilization of Γ' is $U' = 2/5 + 11/15 = 1.133$. Note that Γ is also not schedulable in this case, and in fact, the utilization under preemption-centric accounting is greater than under task-centric accounting in contrast to the example

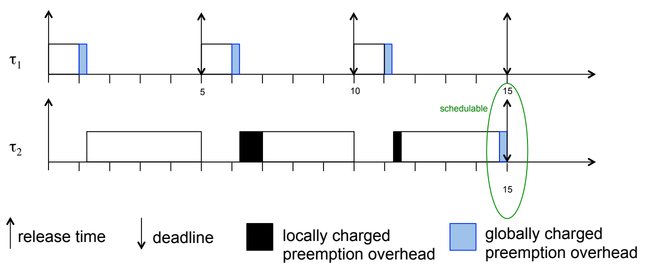

τ1
τ2
5
10
15
15
schedulable
release time
deadline
locally charged preemption overhead
globally charged preemption overhead

Figure 4: Illustrate of ARPO applied to the limited-preemption task system in Tbl. 2.

given in the previous section.

Observe that in this example, preemption-centric accounting does not use any of the limited-preemption-specific task-system information, such as the number of preemptions b_i or the duration of specific preemptions $\Delta_{i,k}$. Under ARPO, this information is incorporated into the overhead accounting, while leveraging the benefits of preemption-centric accounting. Specifically, equation (3), can be extended as follows.

$$C_i' = C_i + \sum_{k=1}^{b_i} \max(0, \Delta_{i,k} - G) + G \qquad (5)$$

Example 6. Consider again the example task system in Tbl. 2, which is depicted using ARPO in Fig. 4, and assume that $G = 0.25$. In the resulting task set Γ', the $C_1' = C_1 + G = 1.25$ and the WCET of task τ_2' is given by $C_1' = C_1 + \sum_{k=1}^{b_i} \max(0, \Delta_{i,k} - G) + G = 10 + (1 - 0.25) + (0.5 - 0.25) + 0.25 = 11.25$. Therefore, the total task-set utilization is $U' = 1.25/5 + 11.25/15 = 1$, which is schedulable under EDF on a uniprocessor.

This example demonstrates how redistributing 0.25 time units of overhead from τ_2 to τ_1 is safe and reduces overhead-related analysis pessimism, while still producing a safe approximation. Next, we show how the linear program from Sec. 3 can be altered for limited-preemption schedulers.

Optimization formulation. To formulate ARPO as applied to limited-preemption schedulers, the first two constraints sets described previously must be replaced to reflect (5). Constraint Sets 3 and 4 can be used without modification. The constants and variables are largely the same as in the previous formulation with the following exception. Instead of introducing a single variable L_i for each task $\tau_i \in \Gamma$, there is a variable to model the local charge for each preemption point, $L_{i,k}$ for each $\tau_i \in \Gamma$ and $k \in \{1, \ldots, b_i\}$.

Constraint Set 1 is replaced by the following.

Constraint Set 5. *The linear constraints corresponding to* $\max(0, \Delta_{i,k} - G)$ *are given by*

$$\forall \tau_i \in \Gamma, k \in \{1, \ldots, b\} : L_{i,k} \geq \Delta_{i,k} - G,$$
$$\forall \tau_i \in \Gamma, k \in \{1, \ldots, b\} : L_{i,k} \geq 0.$$

Similarly, Constraint Set 2 is replaced by the following.

Constraint Set 6. *The linear constraints corresponding to (5) are given by*

$$\forall \tau_i \in \Gamma : C'_i \geq C_i + \sum_{k=0}^{b_i} L_{i,k} + G.$$

These two constraints, in conjunction with Constraint Sets 3 and 4 model ARPO under limited-preemptive schedulers. The minimization objective of this linear program is the same as before: minimize total task-set utilization. We note that ARPO does *not* account for any blocking caused by the effects of non-preemptive execution. Such blocking must be incorporated separately in schedulability analysis.

5. DISCUSSION

Thus far, we have presented ARPO, a general preemption-overhead accounting technique, and shown how linear programming can be used to minimize the contribution of preemption-related overhead to system utilization. While minimizing utilization in most cases improves schedulability, there are some task systems for which this is not the case. Most often this is due to the presence of one or more tasks with very large utilizations, which are more difficult to schedule. In the remainder of this section, we discuss how G can be optimally set, sometimes at the expense of increased runtime complexity.

Soft real-time schedulability. It has been shown that global EDF is optimal with respect to soft real-time schedulability under the definition of soft real-time correctness that requires the extent of any deadline tardiness to be analytically bounded (*i.e.*, finite). There exists a large body of work on soft real-time scheduling (see [16] for recent work and relevant citations). An important and relevant result in soft real-time scheduling is that $U \leq m$ implies soft real-time schedulability. So while deadline misses are possible, the maximum tardiness of each job is bounded. Thus, when ARPO is applied to soft real-time systems scheduled under global EDF, the linear program described in Sec. 3 produces an optimal setting of G. Specifically, if there exists a setting of G that results in a system with bounded tardiness, then the task system produced by the ARPO linear program will also have bounded tardiness. This follows from the fact that the linear program minimizes system utilization, and the utilization-based soft real-time schedulability test is $U \leq m$.

Minimizing response times. Ideally for hard real-time system, we would like to include constraint(s) in our ARPO linear program to ensure that the response time of each task is at most its deadline. The result of such a formulation would be that if the solver returned a feasible solution, then the system would be schedulable. However, most algorithms for computing response times are not of polynomial time complexity (*e.g.*, [8, 22]), and therefore cannot be incorporated in a linear program, which can be solved in polynomial time. For this reason, in this work we have optimized for total task-set utilization, which we believe is a close proxy for schedulability.

An alternative approach, which is more computationally expensive, is to instead consider an *integer linear program (ILP)*. Lisper and Mellgren [20] demonstrated how classic response-time analysis equations can be formulated as an ILP. It may be possible to integrate ideas from ARPO into such an ILP so that overhead-aware response times can be minimized, thereby ensuring schedulability if it is feasible for any setting of G. However, it has been previously noted [20] that such an ILP can be very expensive to solve.

6. EVALUATION

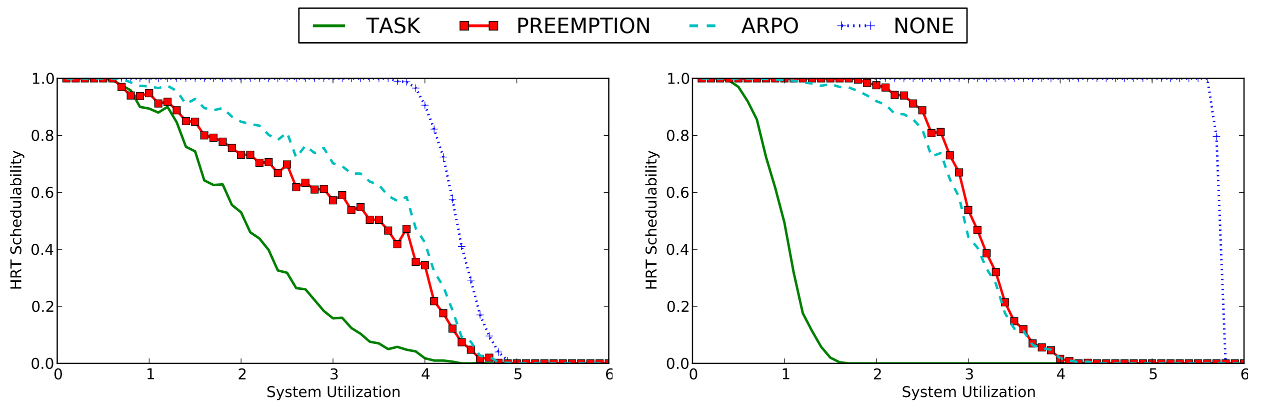
In this section, we present a new experimental design for overhead-aware schedulability experiments, and use it to evaluate ARPO. To our knowledge, this is the first multiprocessor overhead-aware schedulability study to consider task systems with different per-task WSSs.

Experimental design. In this section, we evaluate ARPO on the basis of hard real-time schedulability, which we assess by determining the fraction of randomly generated task systems that are schedulable. We randomly generated task systems using a largely similar methodology to previous studies [9]. We considered *implicit-deadline* task systems (*i.e.*, $D_i = T_i$) scheduled by global EDF on a 6-core processor.[6] We considered *short*, *moderate*, and *long* periods, selected uniformly over $[3, 33)ms$, $[10, 100)ms$, and $[50, 250)ms$, respectively. Per-task utilizations were chosen either uniformly over $[0.001, 0.1]$ (*light*), $[0.1, 0.4]$ (*medium*), $[0.5, 0.9]$ (*heavy*); exponentially with a mean of 0.1 (*light*), 0.25 (*medium*) and 0.5 (*heavy*); or bi-modally over either $[0.001, 0.5)$ or $[0.5, 0.9]$ with respective probabilities of 8/9 and 1/9 (*bimo-light*), 6/9 and 3/9 (*bimo-medium*), and 4/9 and 5/9 (*bimo-heavy*). For each design point, between 500 and 5,000 task systems were generated, or until the mean was estimated to within a confidence interval of 0.05.

To the best of our knowledge, all previous overhead-aware multiprocessor schedulability studies have assumed that all tasks have the same WSS (non-constant WSSs have been considered in uniprocessor schedulability studies, *e.g.*, [23]). While this assumption has proved useful in studying multiprocessor overhead accounting techniques, it is subject to two significant weaknesses. First, such an assumption is rather unrealistic, as in practice different tasks' cache usage may clearly differ within the same system, and also pessimistic, as assuming the largest task's WSS for all tasks, though valid, is an over-approximation. Second, by setting a task's WSS without considering its execution requirement (or vice versa), it is possible to create tasks with unrealistic WSSs. Tasks with small execution times cannot access as much memory during their execution, and thus must have smaller WSSs in practice. This observation motivates the WSS selection in our experimental design.

In our experiments, we randomly choose WSSs corresponding to the maximum amount of data a task may access within some fraction of its execution time. This process is based on the assumption that longer-running tasks likely access more memory than shorter-running tasks. Such fractions were chosen as either a constant of 0.1 (*light*), 0.25 (*medium*), or 0.5 (*heavy*); uniformly over $[0.01, 0.1]$ (*light*), $[0.1, 0.25]$ (*medium*), or $[0.25, 0.5]$ (*heavy*); or bi-modally over either $[0.01, 0.1]$ or $[0.25, 0.5]$ with respective probabilities of 8/9 and 1/9 (*bimo-light*), 6/9 and 3/9 (*bimo-medium*), and 4/9 and 5/9 (*bimo-heavy*). Importantly, under all of these different WSS distributions, the worst-case preemption overhead is no

[6]As suggested by [9], clustered scheduling improves schedulability for higher core counts. We therefore focus on lower core counts, or the schedulability within a cluster.



(a) Bimo-light memory, long periods, heavy task utilizations.

(b) Bimo-light memory, long periods, light task utilizations.

Figure 5: Sample schedulability graphs.

greater than the task's execution time.

Overhead measurements used in our schedulability study were taken from previous experiments [9] on a 24-core 64-bit Intel Xeon L7455 system with four physical sockets and uniform memory access (UMA). We evaluated schedulability assuming tasks were globally scheduled within each six-core socket. Each socket has a shared 12 MB L3 cache.

Using this experimental design, we generated over 200 schedulability graphs, which can be found online [30]. Herein, we present a representative sample of a few graphs in Fig. 5, which depicts relevant trends. Each graph plots hard real-time (HRT) schedulability against system utilization under a specific choice of per-task period, utilization, and WSS random distributions. The curves denoted `TASK` and `PREEMPTION` represent task- and preemption-centric overhead accounting, respectively. The curve denoted `ARPO`, represents ARPO overhead accounting based on the linear program described in Sec. 3. Finally, the curve denoted `NONE`, depicts is a reference curve that depicts schedulability assuming no overheads are accounted for, and is therefore an upper bound on overhead-aware schedulability.

Obs. 1. *ARPO can improve real-time schedulability, in some cases providing more than half a processor's worth of additional processing capacity.*

This observation is supported by Fig. 5 (a). This demonstrates how redistributing preemption overheads can reduce task-set utilization, and thereby have a significant positive impact on schedulability.

Obs. 2. *Choosing G in ARPO via the utilization-based LP optimization can sometimes slightly decrease HRT schedulability.*

This observation is supported by Fig. 5 (b). This result may seem surprising at first, given that ARPO generalizes both preemption-centric accounting ($G = \Delta^{\max}$) and task-centric accounting ($G = 0$). However, because the LP solver in ARPO minimizes total utilization, it is possible that by redistributing the overheads to minimize utilization, some tasks' worst-case response time may be larger than using task- or preemption-centric accounting, which may yield a larger effective utilization. As discussed in Sec. 5, there may be ways of setting G to improve schedulability, however, they likely are more computationally expensive.

Obs. 3. *ARPO provides a greater schedulability benefit for task systems with fewer tasks (i.e., heavier-utilization tasks).*

This observation is supported by comparing insets (a) and (b) of Fig. 5 in which per-task utilizations are heavy and light, respectively. Note that in Fig. 5 (a), ARPO provides a significant schedulability benefit. This is to be expected, as task-centric accounting is most beneficial when the number of tasks is small, because bounds on the number of times each task can be preempted are less pessimistic. For task systems in which the number of tasks is large, G is chosen by the LP solver to charge the preemption overheads predominantly to the relinquishing task, similarly to preemption-centric accounting. Thus, in such cases, performance is very similar to preemption-centric overhead accounting.

7. CONCLUSION

In this paper, we have presented a new preemption-overhead accounting technique, ARPO, which is a hybrid of classical task- and preemption-centric overhead accounting techniques. In particular, preemption overheads can be charged in part to the preempted task, and in part to the task that relinquished the processor allowing the preempted task to resume. We have demonstrated how to redistribute the cost of such preemptions among all tasks in the system through the use of a linear program, which minimizes the total task-set utilization including overheads. ARPO is applicable to any JLFP scheduler, and also some limited-preemption schedulers. We also presented the first experimental-design framework for multiprocessor schedulability experiments that considers task systems wherein the WSS varies from task to task. We evaluated ARPO using this experimental-design framework, and demonstrated that ARPO can significantly benefit real-time schedulability.

References

[1] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *Real-Time Systems Symposium (RTSS)*, December 2003.

[2] S. Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2005.

[3] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. Improved multiprocessor global schedulability analysis. *Real-Time Systems*, 2010.

[4] S. Baruah, J. Gehrke, and G. Plaxton. Fast scheduling of periodic tasks on multiple resources. In *Parallel Processing Symposium*, 1995.

[5] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium (RTSS)*, 1990.

[6] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *The Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT)*, 2010.

[7] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo. Preemption points placement for sporadic task sets. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2010.

[8] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Real-Time Systems Symposium (RTSS)*, 2007.

[9] B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2011.

[10] B. Bui, M. Caccamo, L. Sha, and J. Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2008.

[11] A. Burns, K. Tindell, and A. Wellings. Effective analysis for engineering real-time fixed priority schedulers. *Transactions on Software Engineering*, 1995.

[12] J. V. Busquets-Mataix., J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In *Real-Time Technology and Applications Symposium (RTAS)*, 1996.

[13] G. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004.

[14] G. Buttazzo, M. Bertogna, and G. Yao. Limited preemptive scheduling for real-time systems: A survey. *Transactions on Industrial Informatics*, 2012.

[15] R. Davis, A. Burns, J. Marinho, V. Nelis, S. Petters, and M. Bertogna. Global fixed priority scheduling with deferred pre-emption. In *Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2013.

[16] J. Erickson, J. Anderson, and B. Ward. Fair lateness scheduling: Reducing maximum lateness in G-EDF-like scheduling. *Real-Time Systems*, 2014.

[17] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. 2003.

[18] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 1986.

[19] C. Lee, J. Hahn, Y. Seo, S. Min, R. Ha, S. Hong, C. Park, M. Lee, and C. Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *Transactions on Computers*, 1998.

[20] B. Lisper and P. Mellgren. Response-time calculation and priority assignment with integer programming methods. In *Work-in-progress and Industrial Sessions, Euromicro Conference on Real-Time Systems*, June 2001.

[21] J. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[22] L. Lundberg. Multiprocessor scheduling of age constraint processes. In *Real-Time Computing Systems and Applications (RTCSA)*, 1998.

[23] W. Lunniss, S. Altmeyer, and R. Davis. A comparison between fixed priority and EDF scheduling accounting for cache related pre-emption delays. *Leibniz Transactions on Embedded Systems*, 2014.

[24] J. Marinho, V. Nelis, S. Petters, M. Bertogna, and R. Davis. Limited pre-emptive global fixed task priority. In *Real-Time Systems Symposium (RTSS)*, 2013.

[25] C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *ACM Symposium on Theory of Computing*, 1997.

[26] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. RUN: Optimal multiprocessor real-time schedluing via reduction to uniprocessor. In *Real-Time Systems Symposium (RTSS)*, 2011.

[27] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2005.

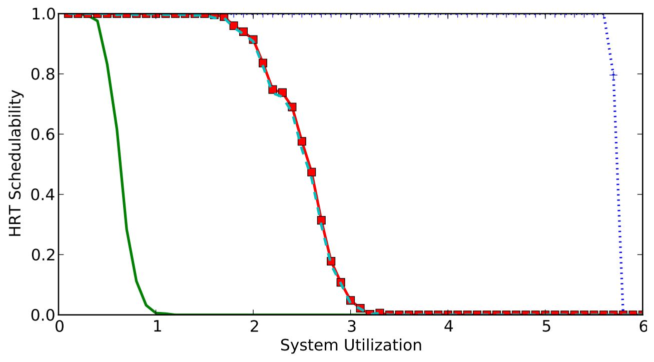
[28] Y. Tan and V. Mooney. Timing analysis for preemptive multitasking real-time systems with caches. In *Transactions on Embedded Computing Systems*, 2007.

[29] A. Thekkilakattil, S. Baruah, R. Dobrin, and S. Punnekkat. The global limited preemptive earliest deadline first feasibility of sporadic real-time tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2014.

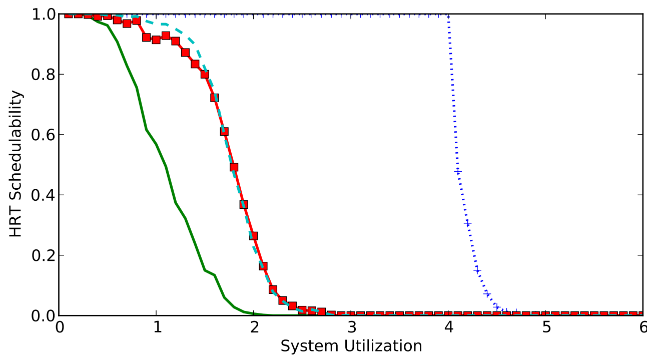
[30] B. Ward, A. Thekkilakattil, and J. Anderson. Optimizing preemption-overhead accounting in multiprocessor real-time systems (online appendix). 2014. `http://www.cs.unc.edu/~anderson/papers.html`.

[31] M. Xu, L. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. Gill. Cache-aware compositional analysis of real-time multicore virtualization platforms. In *Real-Time Systems Symposium (RTSS)*, 2013.

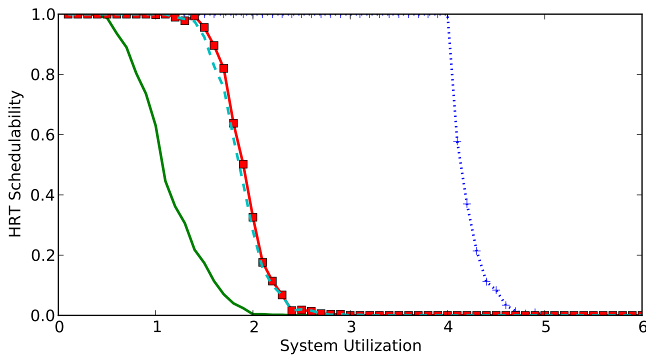
APPENDIX



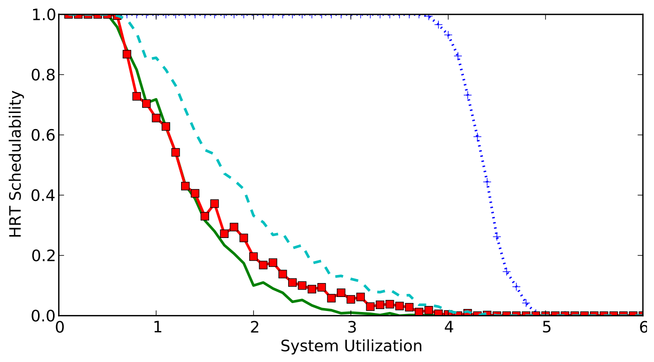
Period: uni-long, Task utilization: uni-light, Memory utilization: bimo-heavy



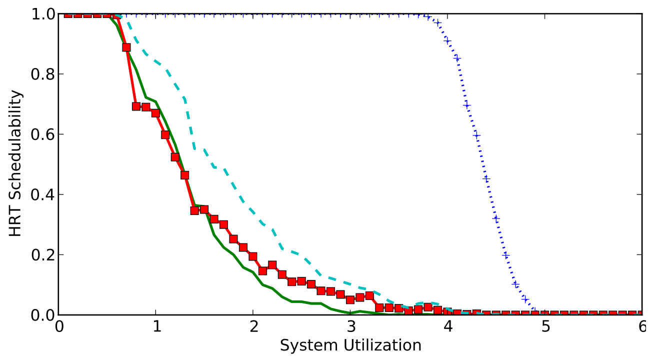
Period: uni-moderate, Task utilization: uni-medium, Memory utilization: bimo-heavy



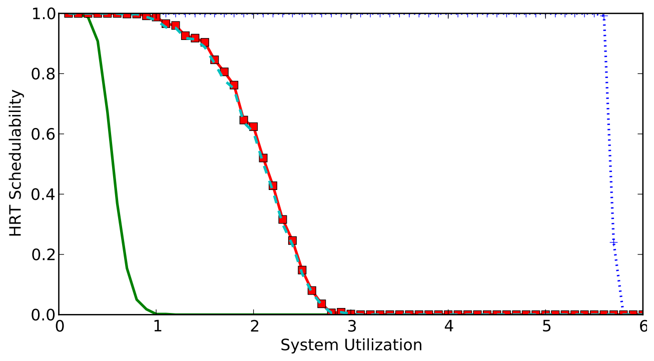
Period: uni-long, Task utilization: uni-medium, Memory utilization: bimo-heavy



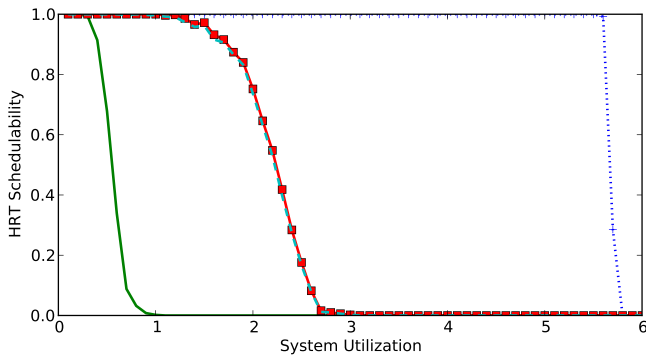
Period: uni-short, Task utilization: uni-heavy, Memory utilization: bimo-heavy



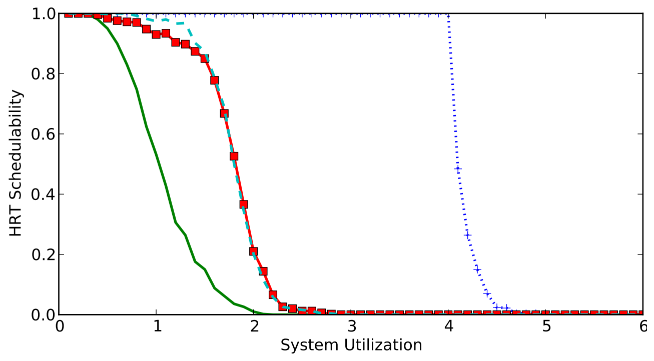
Period: uni-moderate, Task utilization: uni-heavy, Memory utilization: bimo-heavy



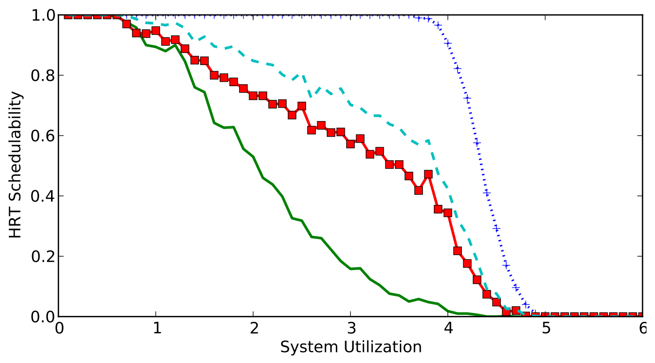
Period: uni-short, Task utilization: uni-light, Memory utilization: bimo-heavy



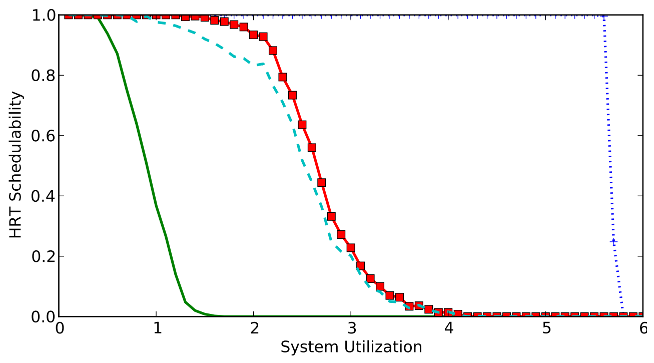
Period: uni-moderate, Task utilization: uni-light, Memory utilization: bimo-heavy



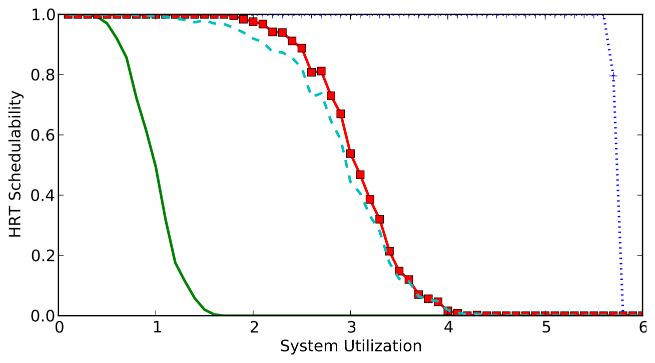
Period: uni-short, Task utilization: uni-medium, Memory utilization: bimo-heavy



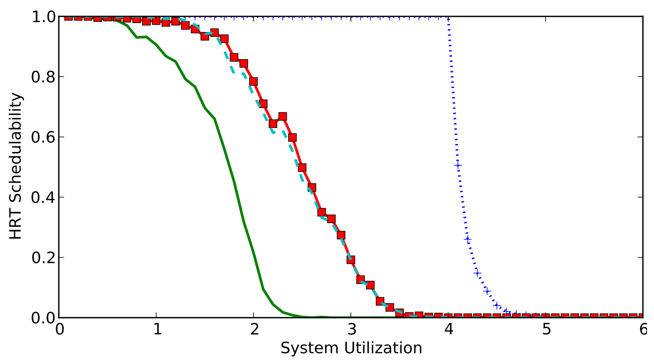
Period: uni-long, Task utilization: uni-heavy,
Memory utilization: bimo-light



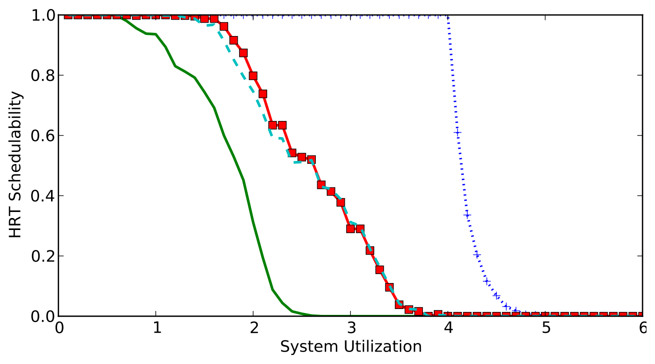
Period: uni-moderate, Task utilization: uni-light,
Memory utilization: bimo-light



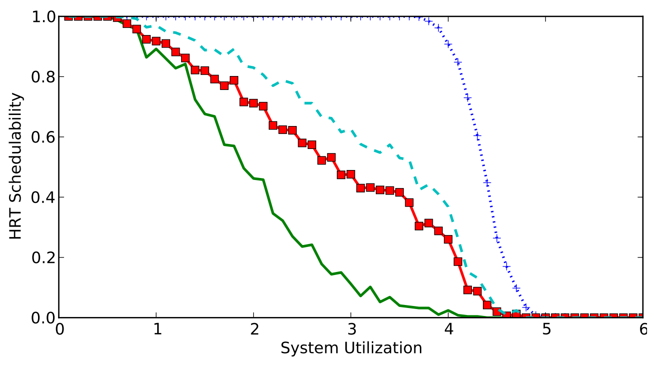
Period: uni-long, Task utilization: uni-light,
Memory utilization: bimo-light



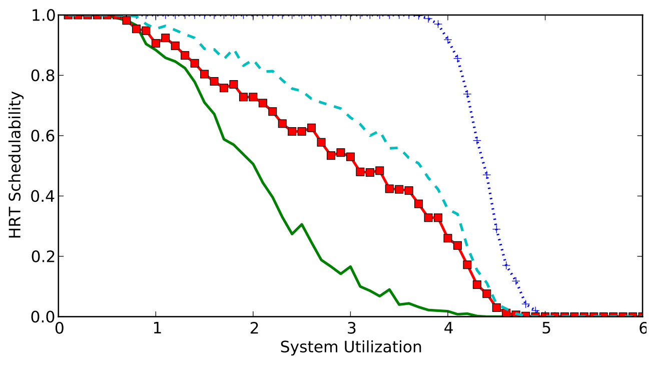
Period: uni-moderate, Task utilization: uni-medium,
Memory utilization: bimo-light



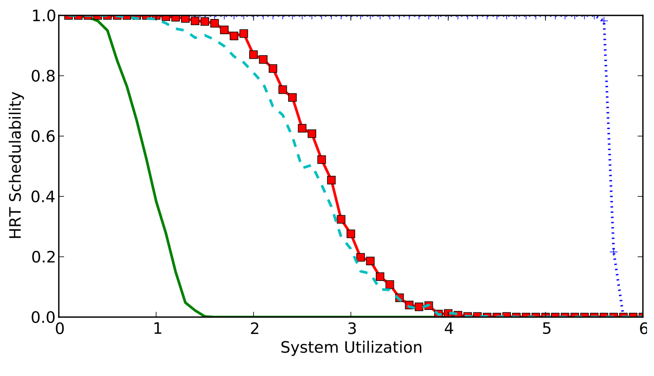
Period: uni-long, Task utilization: uni-medium,
Memory utilization: bimo-light



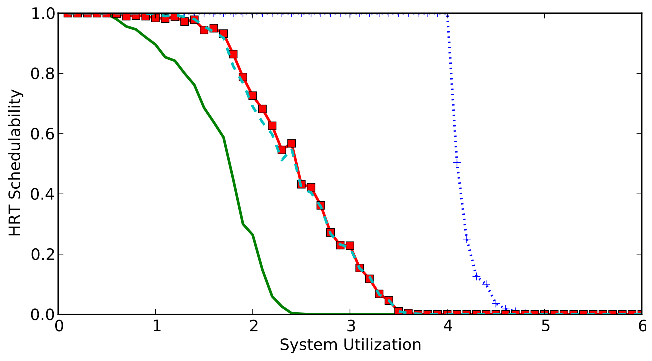
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: bimo-light



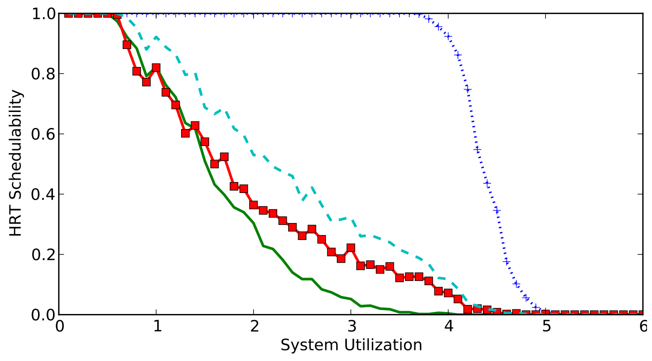
Period: uni-moderate, Task utilization: uni-heavy,
Memory utilization: bimo-light



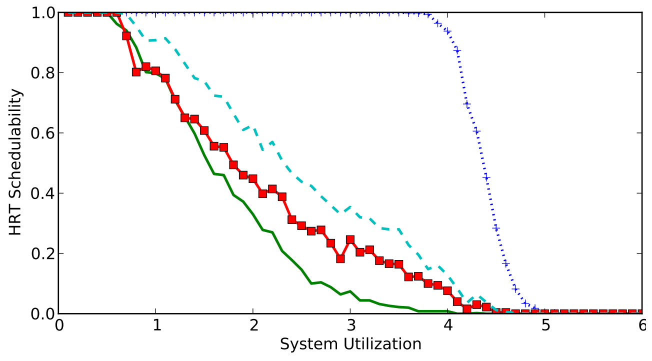
Period: uni-short, Task utilization: uni-light,
Memory utilization: bimo-light



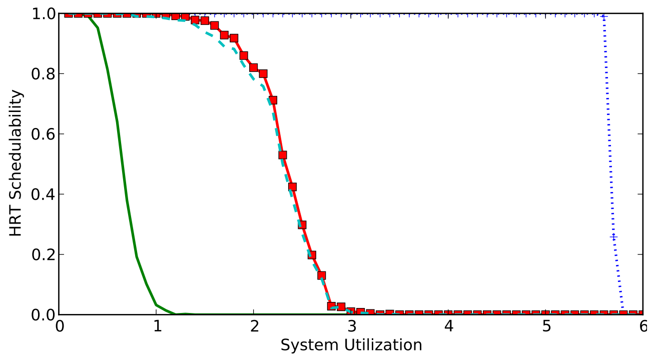
Period: uni-short, Task utilization: uni-medium,
Memory utilization: bimo-light



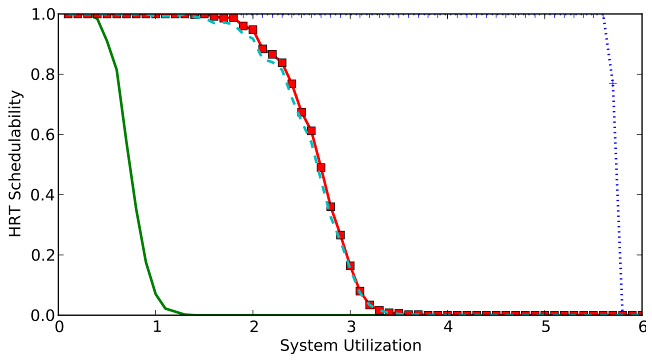
Period: uni-moderate, Task utilization: uni-heavy,
Memory utilization: bimo-medium



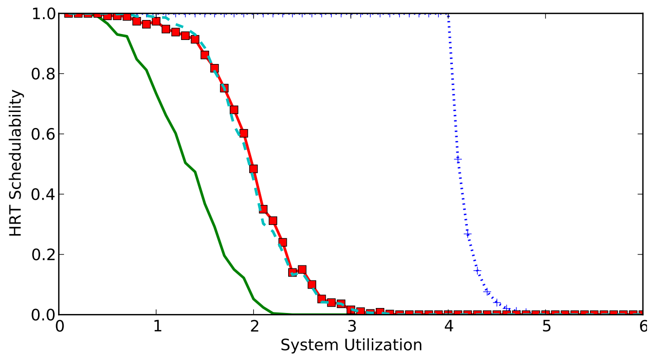
Period: uni-long, Task utilization: uni-heavy,
Memory utilization: bimo-medium



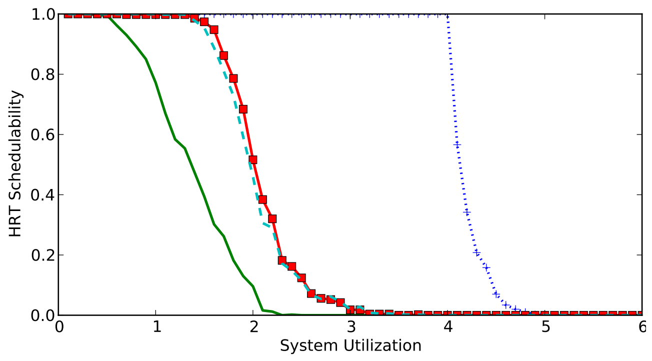
Period: uni-moderate, Task utilization: uni-light,
Memory utilization: bimo-medium



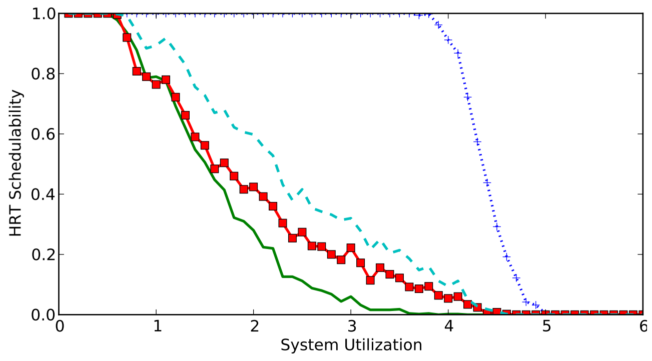
Period: uni-long, Task utilization: uni-light,
Memory utilization: bimo-medium



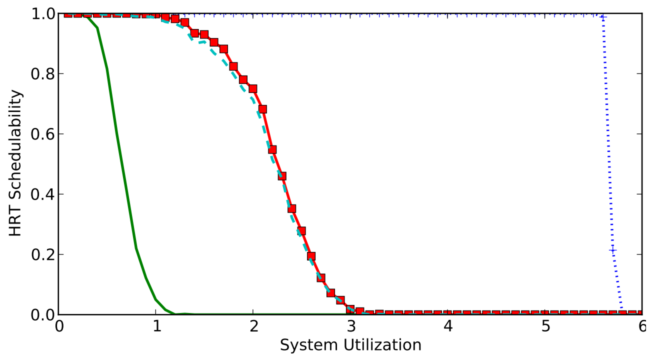
Period: uni-moderate, Task utilization: uni-medium,
Memory utilization: bimo-medium



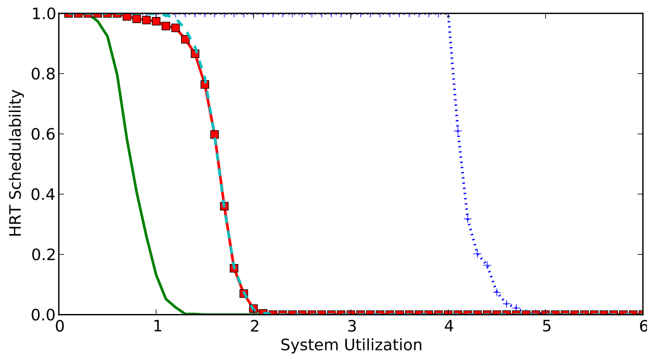
Period: uni-long, Task utilization: uni-medium,
Memory utilization: bimo-medium



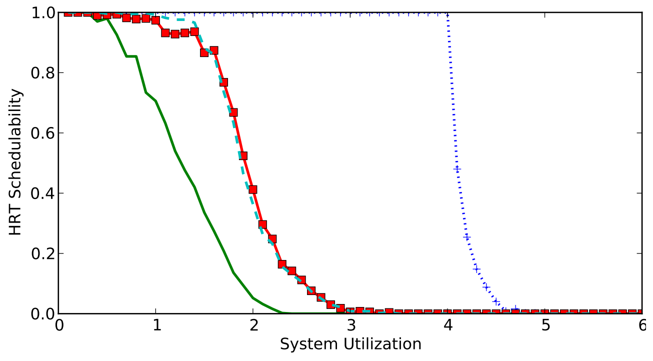
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: bimo-medium



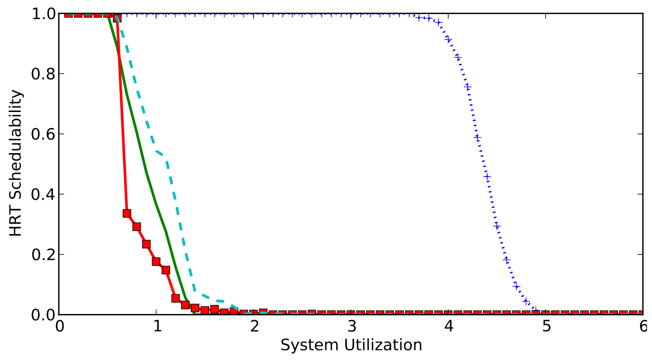
Period: uni-short, Task utilization: uni-light, Memory utilization: bimo-medium



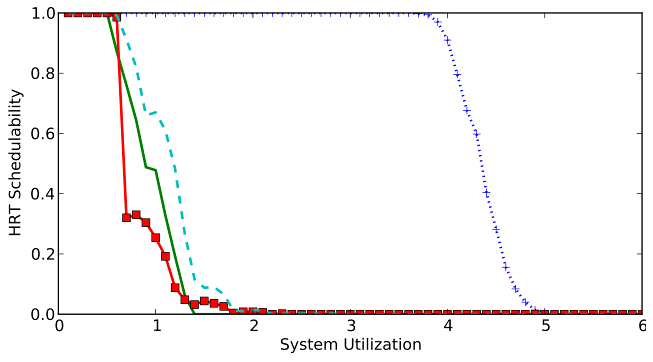
Period: uni-long, Task utilization: uni-medium, Memory utilization: const-heavy



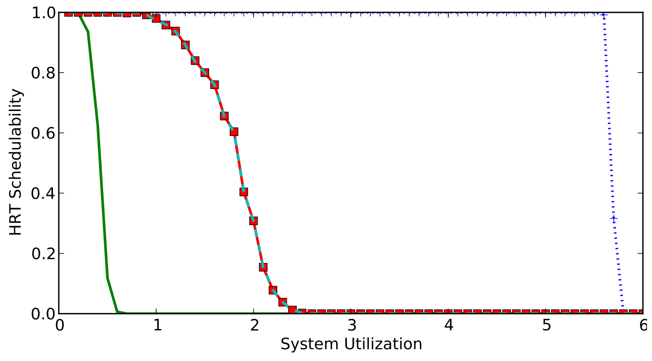
Period: uni-short, Task utilization: uni-medium, Memory utilization: bimo-medium



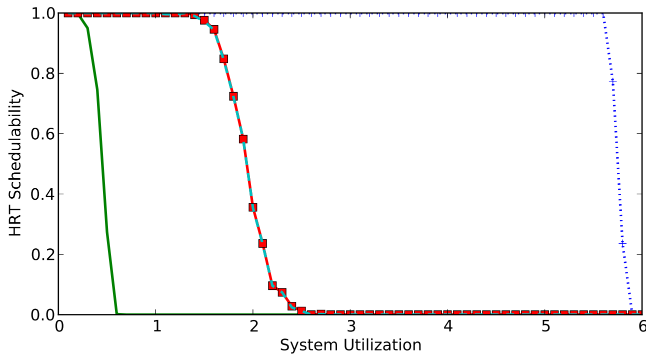
Period: uni-moderate, Task utilization: uni-heavy, Memory utilization: const-heavy



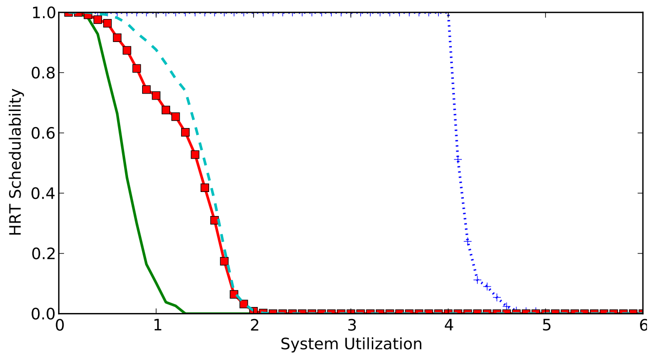
Period: uni-long, Task utilization: uni-heavy, Memory utilization: const-heavy



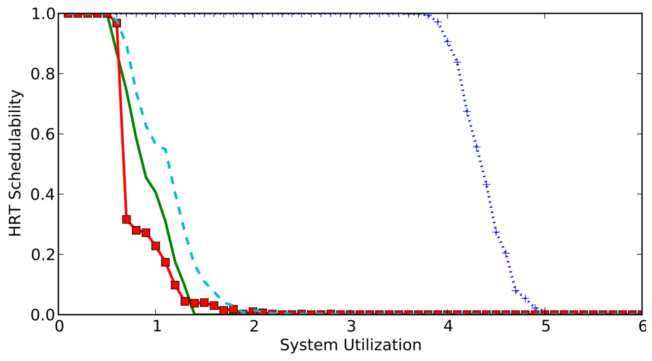
Period: uni-moderate, Task utilization: uni-light, Memory utilization: const-heavy



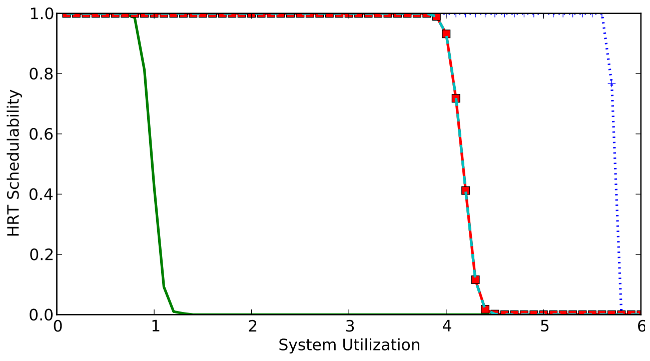
Period: uni-long, Task utilization: uni-light, Memory utilization: const-heavy



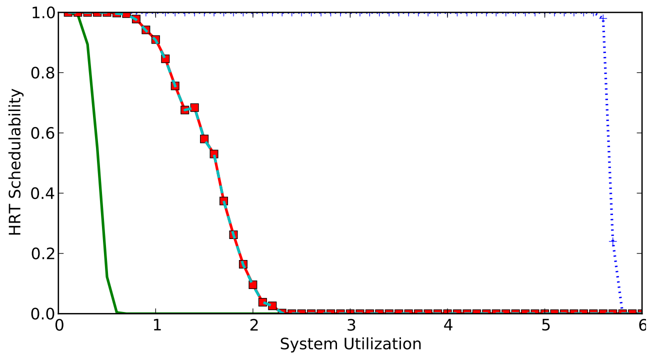
Period: uni-moderate, Task utilization: uni-medium, Memory utilization: const-heavy



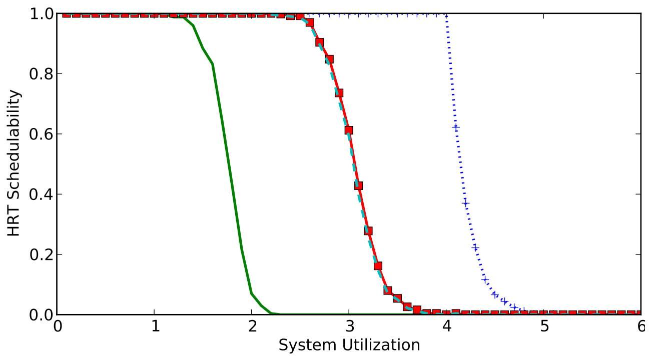
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: const-heavy



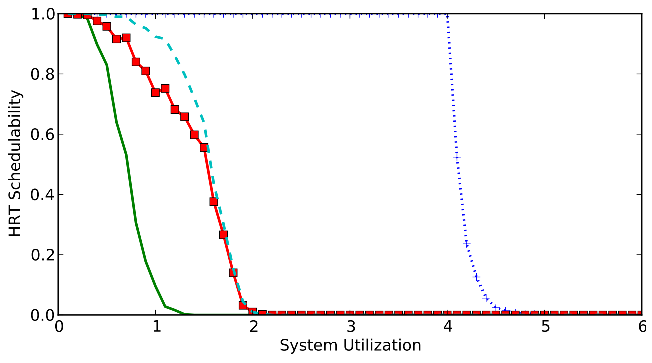
Period: uni-long, Task utilization: uni-light,
Memory utilization: const-light



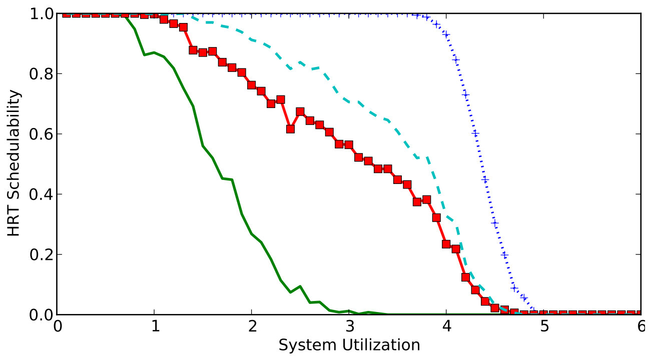
Period: uni-short, Task utilization: uni-light,
Memory utilization: const-heavy



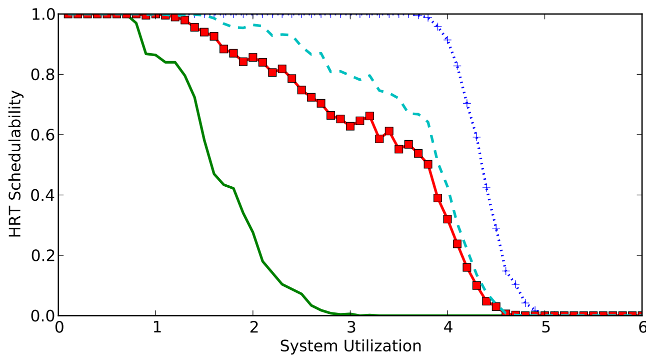
Period: uni-long, Task utilization: uni-medium,
Memory utilization: const-light



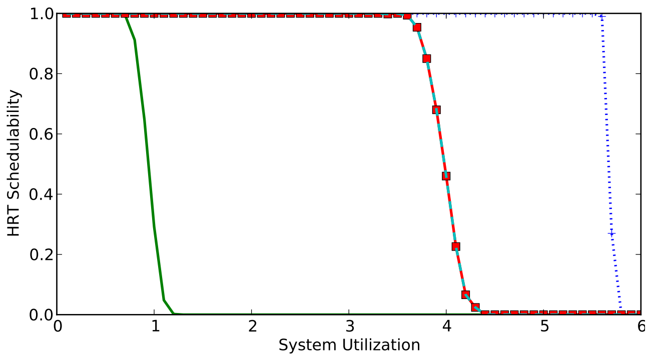
Period: uni-short, Task utilization: uni-medium,
Memory utilization: const-heavy



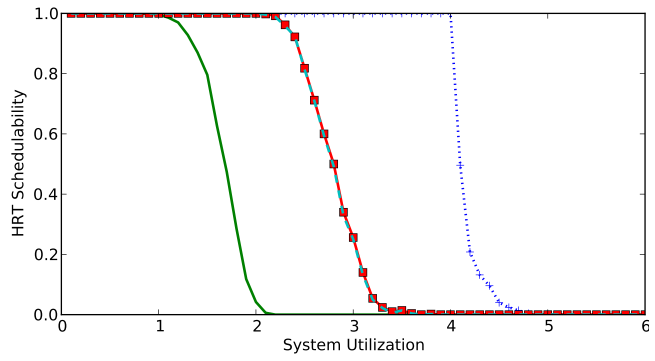
Period: uni-moderate, Task utilization: uni-heavy,
Memory utilization: const-light



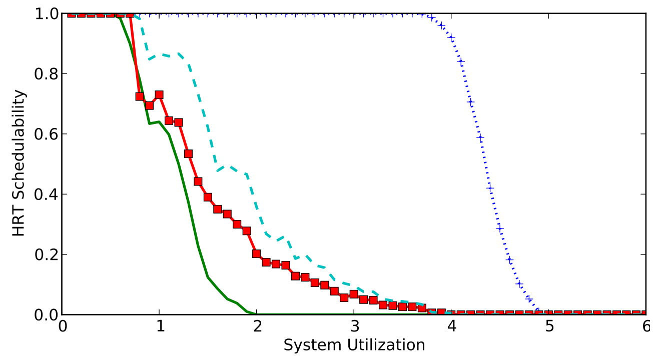
Period: uni-long, Task utilization: uni-heavy,
Memory utilization: const-light



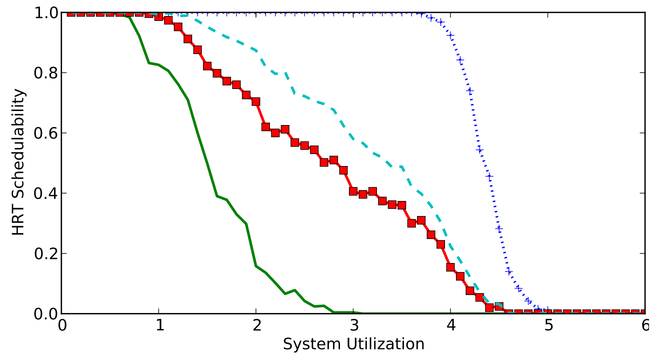
Period: uni-moderate, Task utilization: uni-light,
Memory utilization: const-light



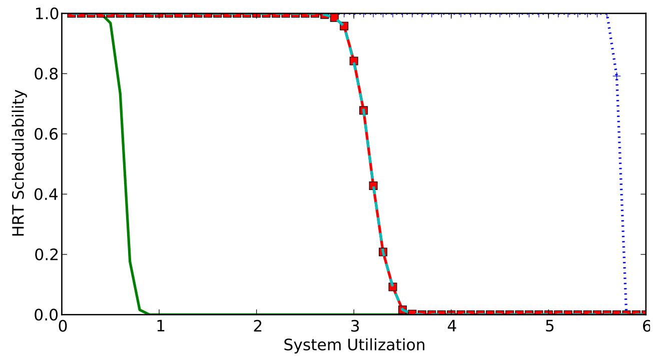
Period: uni-moderate, Task utilization: uni-medium,
Memory utilization: const-light



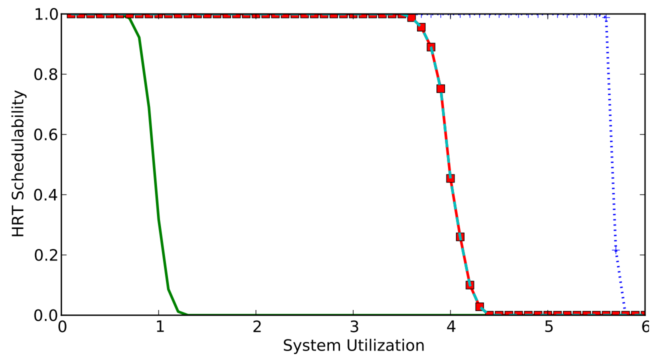
Period: uni-long, Task utilization: uni-heavy,
Memory utilization: const-medium



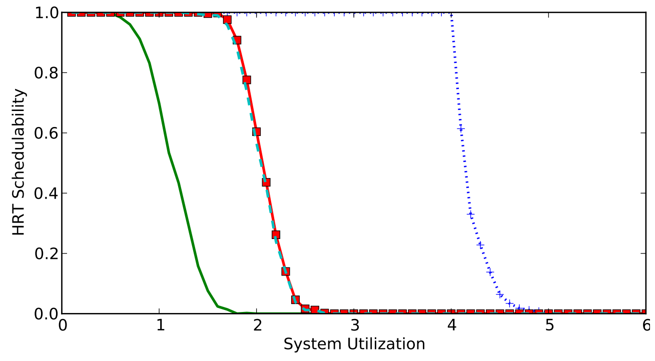
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: const-light



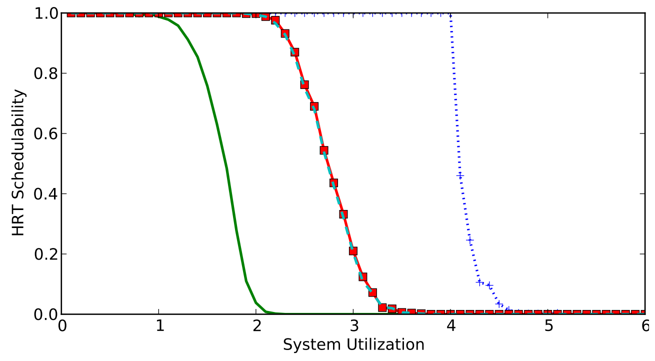
Period: uni-long, Task utilization: uni-light,
Memory utilization: const-medium



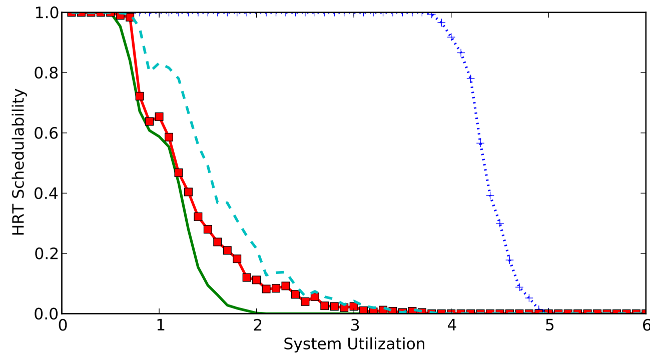
Period: uni-short, Task utilization: uni-light,
Memory utilization: const-light



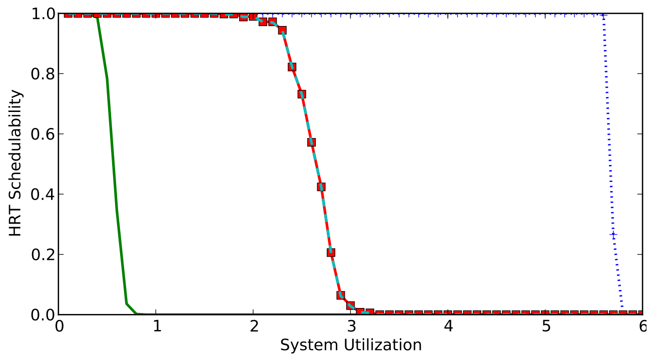
Period: uni-long, Task utilization: uni-medium,
Memory utilization: const-medium



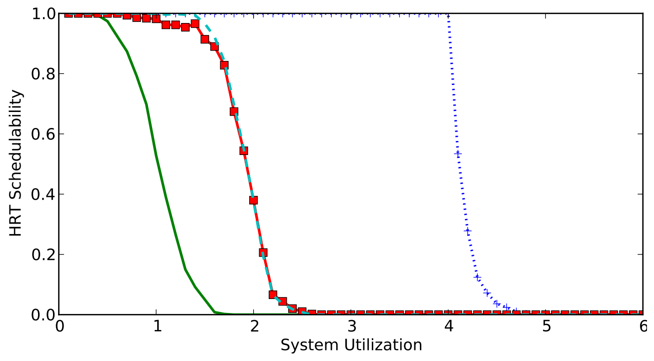
Period: uni-short, Task utilization: uni-medium,
Memory utilization: const-light



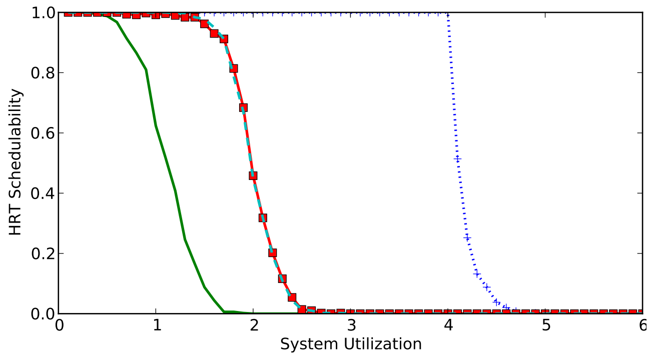
Period: uni-moderate, Task utilization: uni-heavy,
Memory utilization: const-medium



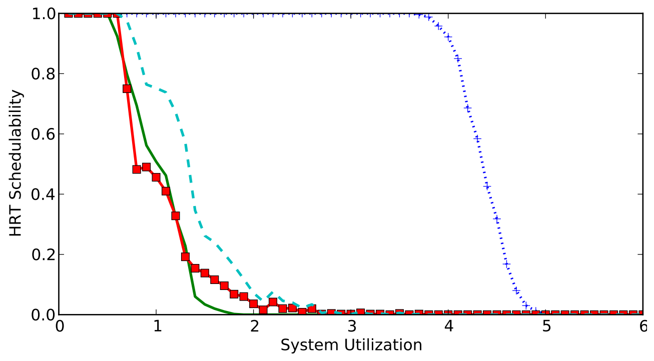
Period: uni-moderate, Task utilization: uni-light,
Memory utilization: const-medium



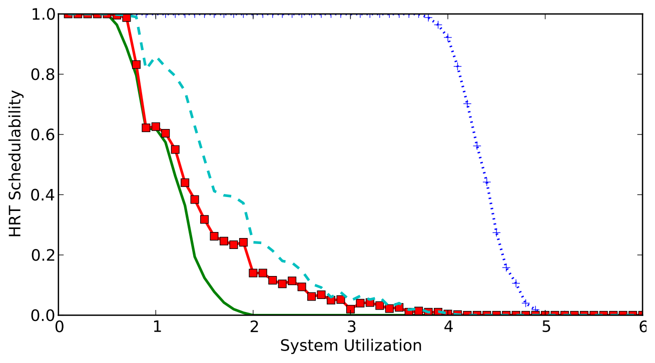
Period: uni-short, Task utilization: uni-medium,
Memory utilization: const-medium



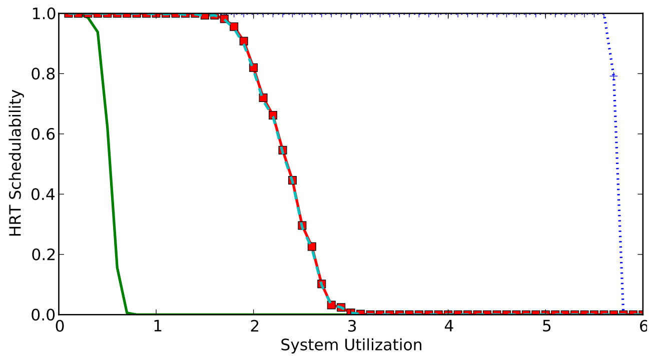
Period: uni-moderate, Task utilization: uni-medium,
Memory utilization: const-medium



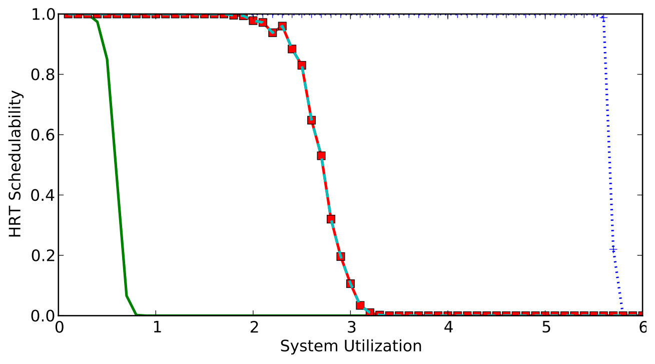
Period: uni-long, Task utilization: uni-heavy,
Memory utilization: uni-heavy



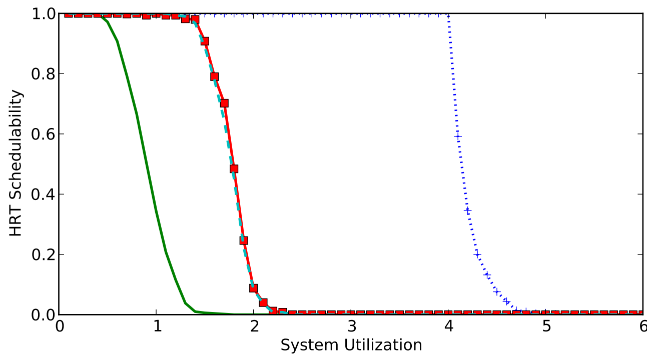
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: const-medium



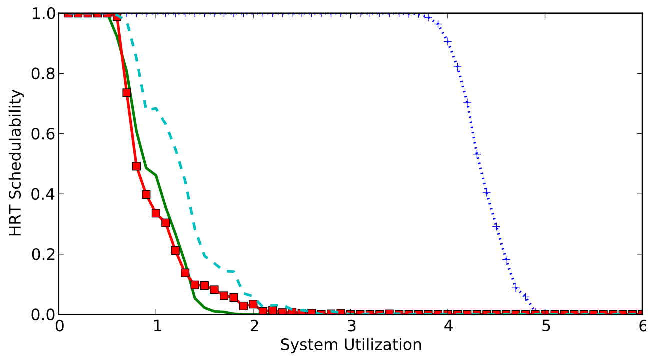
Period: uni-long, Task utilization: uni-light,
Memory utilization: uni-heavy



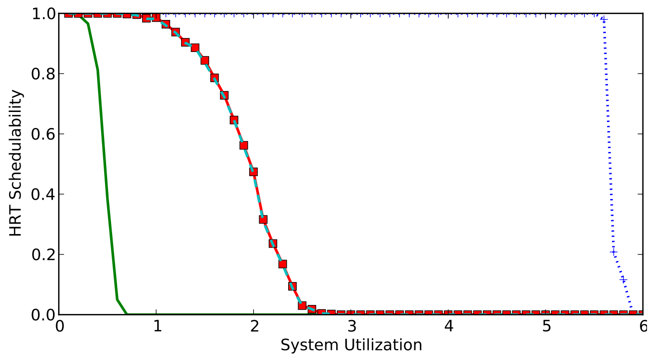
Period: uni-short, Task utilization: uni-light,
Memory utilization: const-medium



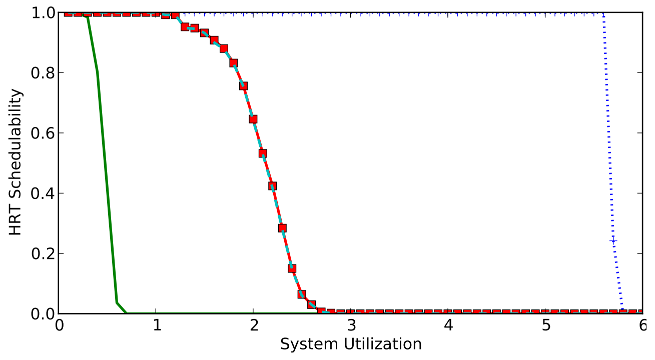
Period: uni-long, Task utilization: uni-medium,
Memory utilization: uni-heavy



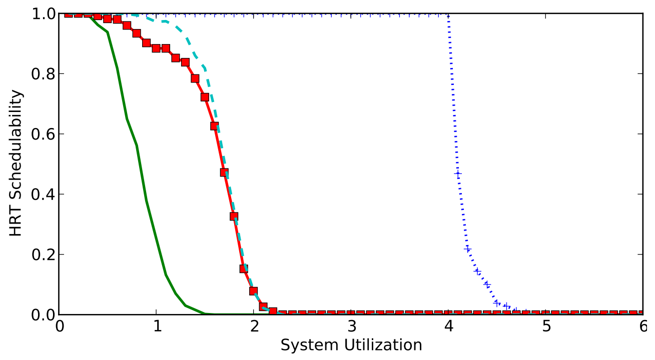
Period: uni-moderate, Task utilization: uni-heavy,
Memory utilization: uni-heavy



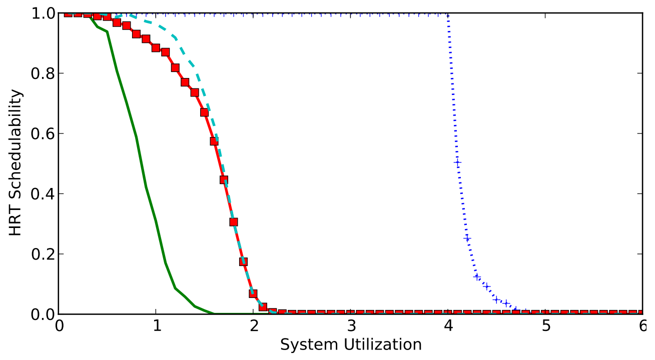
Period: uni-short, Task utilization: uni-light,
Memory utilization: uni-heavy



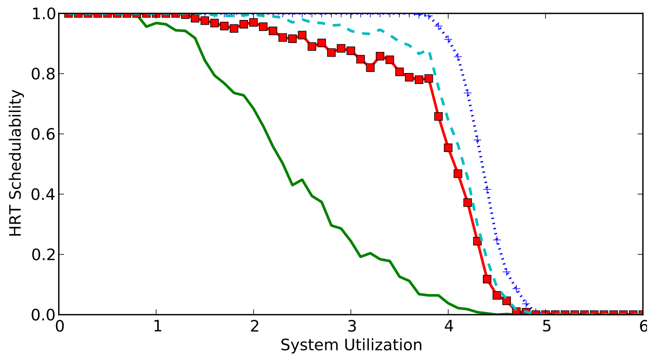
Period: uni-moderate, Task utilization: uni-light,
Memory utilization: uni-heavy



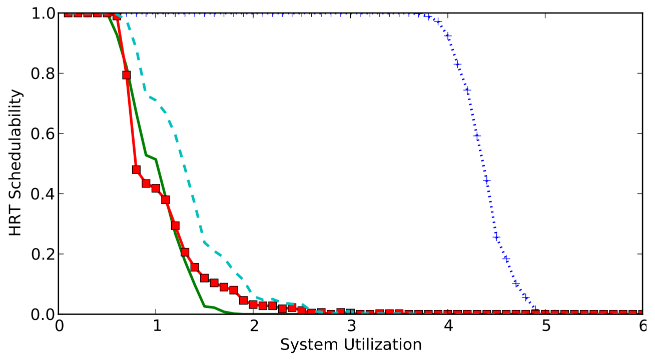
Period: uni-short, Task utilization: uni-medium,
Memory utilization: uni-heavy



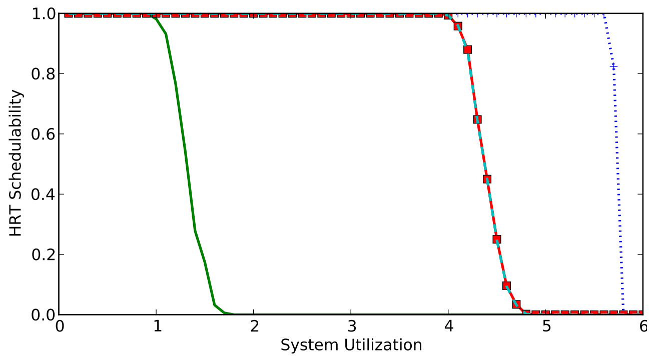
Period: uni-moderate, Task utilization: uni-medium,
Memory utilization: uni-heavy



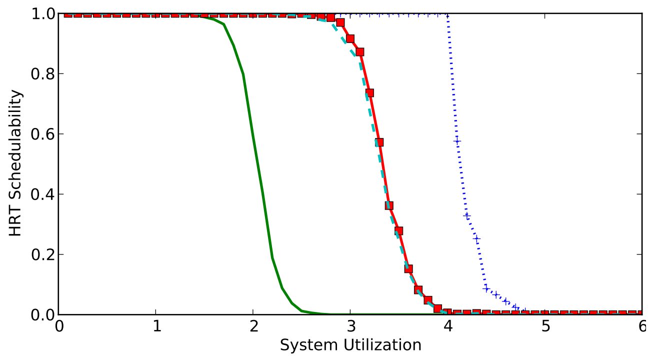
Period: uni-long, Task utilization: uni-heavy,
Memory utilization: uni-light



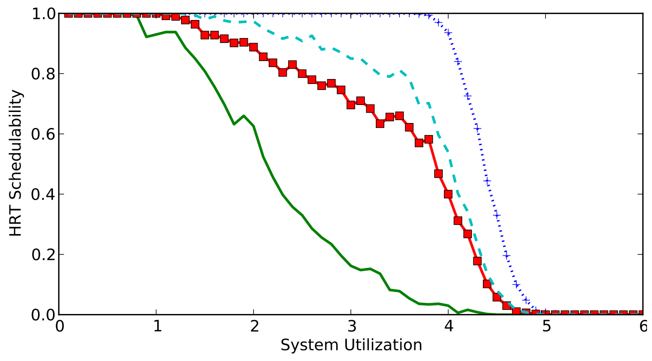
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: uni-heavy



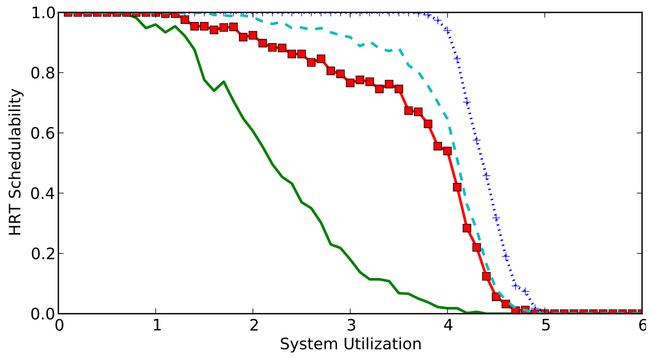
Period: uni-long, Task utilization: uni-light,
Memory utilization: uni-light



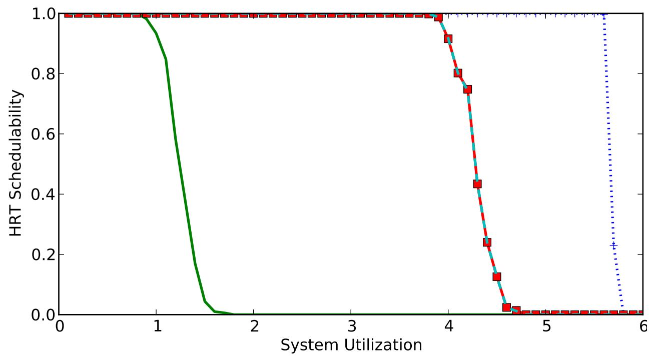
Period: uni-long, Task utilization: uni-medium,
Memory utilization: uni-light



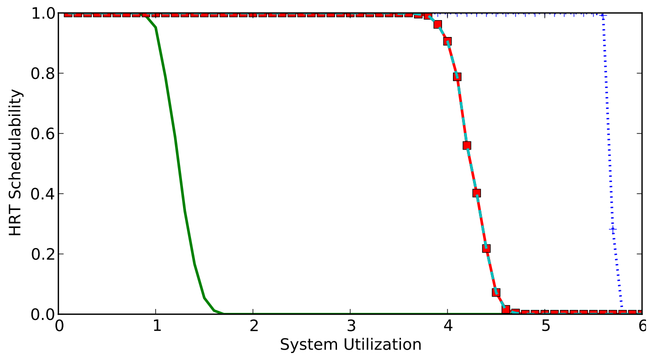
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: uni-light



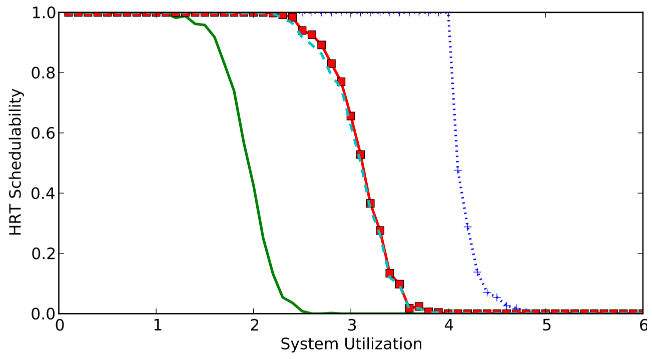
Period: uni-moderate, Task utilization: uni-heavy,
Memory utilization: uni-light



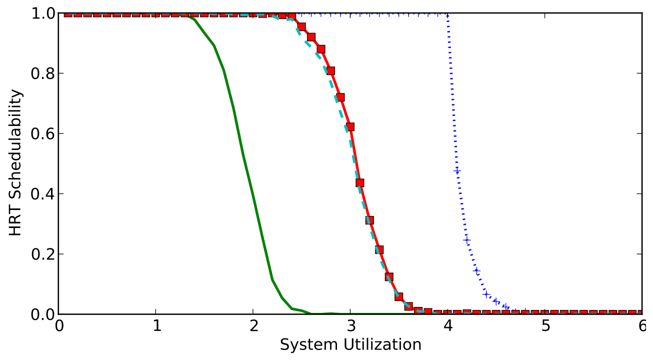
Period: uni-short, Task utilization: uni-light,
Memory utilization: uni-light



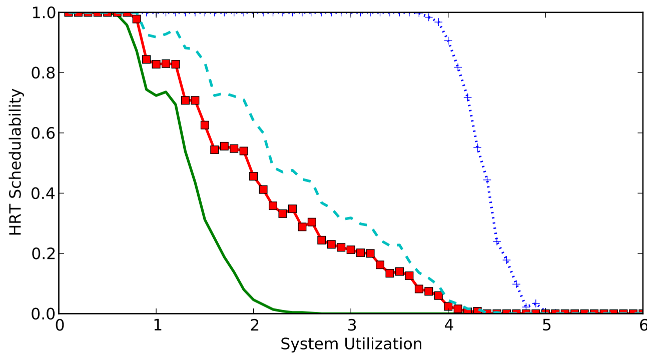
Period: uni-moderate, Task utilization: uni-light,
Memory utilization: uni-light



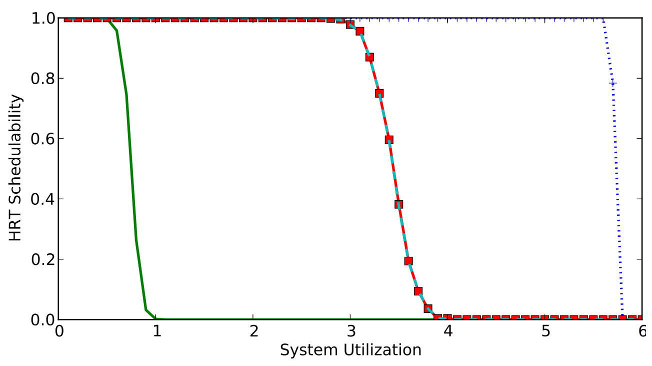
Period: uni-short, Task utilization: uni-medium,
Memory utilization: uni-light



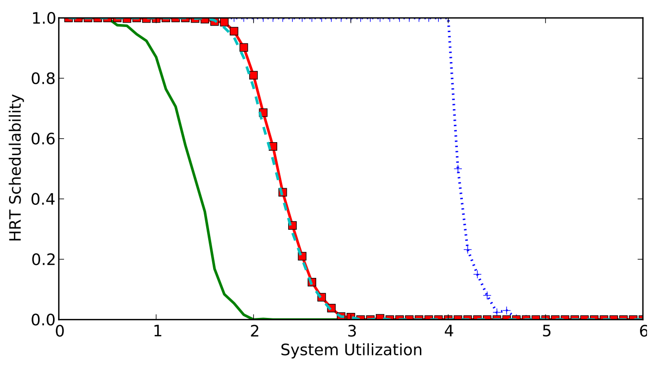
Period: uni-moderate, Task utilization: uni-medium,
Memory utilization: uni-light



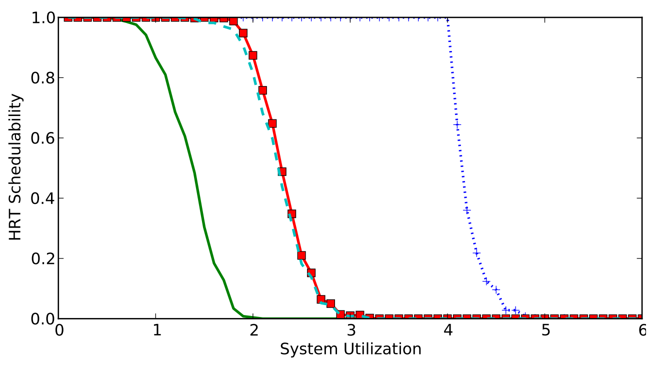
Period: uni-long, Task utilization: uni-heavy,
Memory utilization: uni-medium



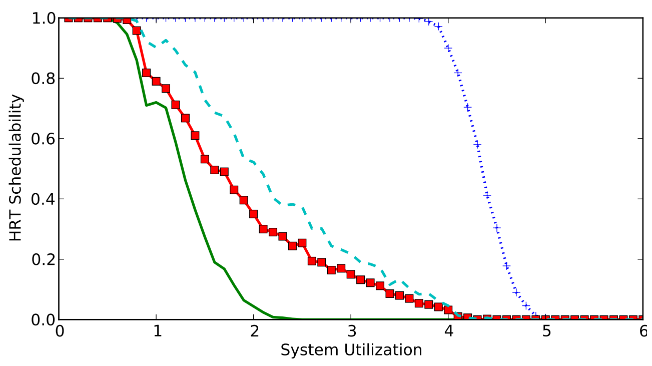
Period: uni-long, Task utilization: uni-light,
Memory utilization: uni-medium



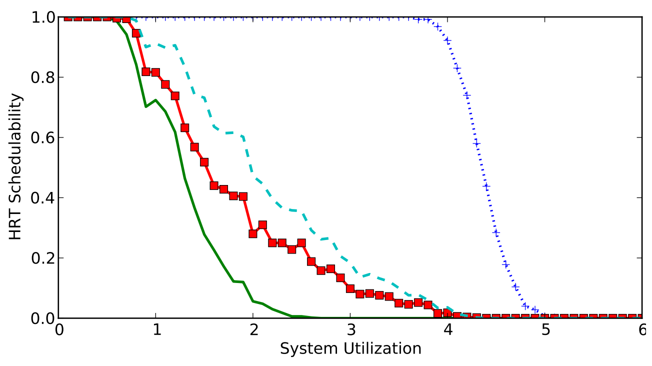
Period: uni-moderate, Task utilization: uni-medium,
Memory utilization: uni-medium



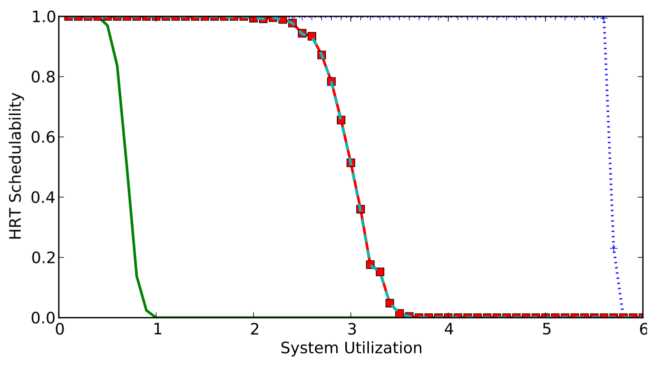
Period: uni-long, Task utilization: uni-medium,
Memory utilization: uni-medium



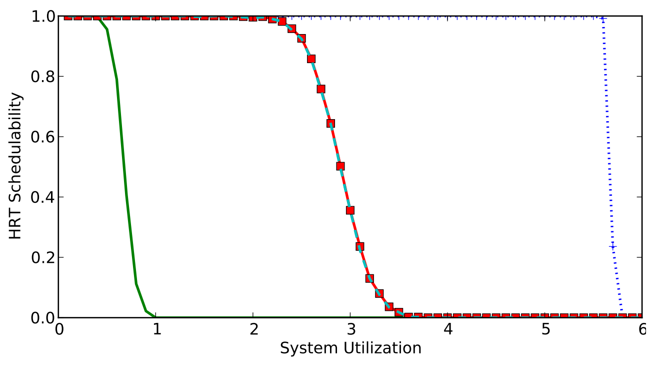
Period: uni-short, Task utilization: uni-heavy,
Memory utilization: uni-medium



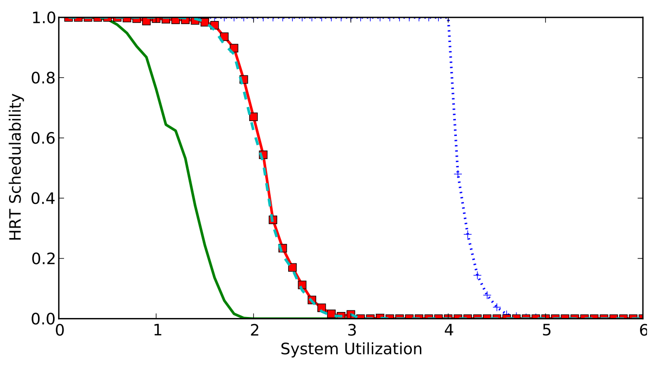
Period: uni-moderate, Task utilization: uni-heavy,
Memory utilization: uni-medium



Period: uni-short, Task utilization: uni-light,
Memory utilization: uni-medium



Period: uni-moderate, Task utilization: uni-light,
Memory utilization: uni-medium



Period: uni-short, Task utilization: uni-medium,
Memory utilization: uni-medium