
Proceedings Paper:
Zhao, Shuai, Garrido, Jose, Burns, Alan orcid.org/0000-0001-5621-8816 et al. (1 more author) (2017) New Schedulability Analysis for MrsP. In: 2017 IEEE 23rd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA). IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 01 Sep 2017 IEEE, p. 1.

https://doi.org/10.1109/RTCSA.2017.8046311

New Schedulability Analysis for MrsP‡

Shuai Zhao*, Jorge Garrido†, Alan Burns*, Andy Wellings*
*Department of Computer Science, University of York, UK
†Departamento de Ingeniería Telemática (STRAST), Universidad Politécnica de Madrid (UPM), Spain

***Abstract*— In this paper we consider a spin-based multiprocessor locking protocol, named the Multiprocessor resource sharing Protocol (MrsP). MrsP adopts a helping-mechanism where the preempted resource holder can migrate. The original schedulability analysis of MrsP carries considerable pessimism as it has been developed assuming limited knowledge of the resource usage for each remote task. In this paper new MrsP schedulability analysis is developed that takes into account such knowledge to provide a less pessimistic analysis than that of the original analysis. Our experiments show that, theoretically, the new analysis offers better (at least identical) schedulability than the FIFO non-preemptive protocol, and can outperform FIFO preemptive spin locks under systems with either intensive resource contention or long critical sections.**

The paper also develops analysis to include the overhead of MrsP's helping mechanism. Although MrsP's helping mechanism theoretically increases schedulability, our evaluation shows that this increase may be negated when the overheads of migrations are taken into account. To mitigate this, we have modified the MrsP protocol to introduce a short non-preemptive section following migration. Our experiments demonstrate that with migration cost, MrsP may not be favourable for short critical sections but provides a better schedulability than other FIFO spin-based protocols when long critical sections are applied.

I. Introduction

The transition from uniprocessors to multiprocessors has been taking place over the last few years to meet the increasing demand for computation power [2]. However, moving to multiprocessor platforms breaks most of the well-known locking protocols and schedulability analysis approaches that are used on uniprocessor platforms, which can only manage resources that are accessed from one processor (*local resources*). With this transition, resource sharing technology that can control resources shared by tasks from different processors (*global resources*) has received much attention. However, as a relatively new area, some of the multiprocessor locking protocols either lack efficient schedulability analysis support or have analysis with considerable pessimism [14].

In this paper we focus on a FIFO spin-based multiprocessor locking protocol, named the Multiprocessor resource sharing Protocol (MrsP) [11]. In MrsP, a helping mechanism is adopted where the resource holder can migrate to be helped under certain situations. Our work starts by reducing the existing pessimism in MrsP response-time analysis [11]. Then we develop new analysis that bounds the migration cost incurred with use of the protocol. By integrating the two analysis approaches, we present a more complete MrsP schedulability analysis. Based on the new combined analysis, a set of experiments are performed to evaluate the efficacy of MrsP.

‡This work has been partially funded by the Spanish National R&D&I plan (project M2C2, TIN2014-56158-C4-3-P).

978-1-5386-1898-1/17/ $31.00 © 2017 IEEE

A. Background and Motivation

The original MrsP analysis provides an acceptable worst-case response time for each task based on limited knowledge of the system. It does this by assuming that, in the worst case, each time a task tries to access a resource, it can be delayed (blocked) once from each remote processor that contains tasks requesting the same resource. If full details of the system and the individual behaviour of each remote task is known then *holistic blocking analysis* [8] can be used. This reduces the pessimism of the original analysis as not all resource access will result in the worst-case blocking. However, as shown by Wieder and Brandenburg [29], *holistic blocking analysis* still suffers from considerable pessimism due to the approach of inflating a task's computation time with its resource accessing time (and potential delay). In the same paper, a new analysis framework for spin locks is developed based on mixed-integer linear-programming (ILP), which further reduces the pessimism.

Unfortunately, the ILP-based analysis does not consider any helping-based locking protocols so that it cannot be applied to MrsP directly. In addition, although the ILP-based analysis provides a valuable uniform analysis tool for a variety of spin locking protocols, when focusing at one protocol, say MrsP, the analysis is relatively complex and expensive due to the use of linear-programming. Further, although the ILP-based analysis can be applied to MrsP with modifications, this would not consider the overheads of the helping mechanism. Consequently, we aim to develop new MrsP analysis that inherits the format of its original analysis in order to obtain a simpler schedulability test but with the same degree of pessimism that the ILP-based analysis achieves. In addition, as in MrsP helping is provided by migrations, we aim to reduce and to bound the migration cost that each resource-requesting task can incur and integrate this into our new analysis to provide a more complete migration-aware schedulability analysis test.

B. Related Work

On uniprocessor platforms, locking protocols have been developed and well-practiced for decades. The Non Preemptive Protocol provides the most straightforward protection to resource-requesting tasks and can be applied with any scheduling scheme. The Priority Inheritance Protocol [24] is

suitable for fixed-priority systems and inspired the creation of protocols that are agreed as the best practice for resource sharing control on uniprocessor platforms — Priority Ceiling Protocol (PCP) [24], the Stack Resource Protocol (SRP) [4] and the Deadline Floor Protocol [9]. These protocols guarantee that a task can only suffer from one blocking, as well as avoiding deadlocks and carrying low run-time overheads [14].

On multiprocessors, MPCP [22] applies a limited migration facility, where the resource holding task should explicitly migrate to a processor before it can acquire the resource. Later, the protocol was renamed as DPCP [20], [21] with the notion of the local agent, which is a remote task that can execute a global resource on behalf of other tasks. MSRP [16] is a spin-based FIFO locking protocol. Under MSRP tasks become non-preemptable while waiting for, and executing with, the resource. With the FIFO non-preemptive approach, MSRP guarantees resource execution progress and is supported by sufficient schedulabiltiy analysis [16]. Later, Wieder and Brandenburg's work [8], [29] gave a more precise schedulability test for MSRP. More recently, Biondi et al [6] presented the first analysis for nested resource access for FIFO spin locks, which can also be directly applied to MSRP. In contrast, the FMLP [7] protocol introduced the notion of resource groups, where resources are grouped based on the length of resources. FIFO spin locks are used to protected short resources while semaphores are adopted to protect long resources. SPEPP [26] and M-BWI [15], [19] apply the notion of a helping mechanism, where a waiting task can execute on behalf of the preempted resource holder. More recently, a protocol named RNLP [28], [27] has been developed to support nested resource requests by applying a token mechanism and a set of request satisfaction mechanisms that can fit into different system models. In [18], a new multiprocessor task partitioning and resource allocating algorithm is proposed to offer a guaranteed speedup of 11-6/(m+1), where m is the number of processors in the system.

As MrsP is a relatively new protocol, the directly related work of MrsP is limited. In [13], Catellani et al. proved that MrsP can be effectively implemented inside the Litmus kernel [12], [8] and RTEMS [23] with acceptable overheads. In addition, several approaches to implement the MrsP primitives are discussed along with the challenges and issues for supporting the required functionalities of the helping mechanism. In [10], MrsP is applied to Ada with a prototype outside-kernel implementation. In [17], a complete definition of nested resources behaviour in MrsP is presented with sufficient analysis based on the original MrsP Response Time Analysis (RTA).

C. System and Task Model

In this work we apply a similar system model to that presented in MrsP's original paper [11]. We assume a fully partitioned multiprocessor system with m processors (P_1 to P_m). We employ a fixed priority scheduling policy and a general constrained *sporadic task model*. Each task in the system, say τ_x, has a priority $Pri(\tau_x)$, a response time R_x, a period T_x, a deadline D_x, a pure worst-case computation time C_x without accessing any resources and a worst-case execution time $\widehat{C_x}$, where $\widehat{C_x}$ is the sum of the pure computation time of task τ_x (C_x) and the time it spends accessing each resource on its and others behalf. A task can generate a bounded set of sequential jobs but only one job can be executable at a time. In this work a higher priority value represents a higher priority.

In the system there exist a set of resources R (e.g., data structures and I/O devices) that are shared by tasks mutual exclusively. As presented in the MrsP paper, two functions are applied to describe the relation between tasks and resources: function $F(\tau_x)$ returns a set of resources that are used by τ_x and function $G(r^k)$ gives a set of tasks that request r^k. For each resource r^k, c_x^k denotes the worst-case execution time when task τ_x accesses r^k. In addition, a task τ_x may request a resource r^k a number of times in one release, which is denoted by N_x^k. In this work, however, we assume (as the original MrsP analysis does) that the accessing time to resource r^k is identical for each task. Hence, the parameter c^k is used in the entire work to denote the critical section length of resource r^k. This assumption is not fundamental but eases presentation.

In this work we assume that a task can only access one resource at a time. That is, we will only focus on non-nested resource accesses. We acknowledge that nested resource access is highly relevant. However, due to the complexity of the topic in this paper and the page limit, this paper focuses on presenting our results for the non-nested case. The results of our research into nested resource access is presented in an independent paper [17].

D. Blocking in Multiprocessors

We inherit the classification of blocking effects from [29]: *spin delay* and *arrival blocking*. A task can incur spin delay when (1) being blocked *directly* by remote requests when accessing a global resource and (2) being blocked *indirectly* by a local high priority task that holds a global resource. A task, say τ_x, can incur arrival blocking if there exists a local lower priority task that requests a resource with current active priority equal or higher than τ_x's priority. For global resources, τ_x can incur remote blocking as the lower priority task can be delayed by remote requests. The arrival blocking occurs before the execution of τ_x and can only happen once. Note that under MrsP a task that incurs *direct spin delay* can still execute on behalf of other resource requesting tasks but is not executable when incurring *indirect spin delay*. Thus, in MrsP, we identify three blocking effects that a task can incur: *direct spin delay*, *indirect spin delay* and *arrival blocking*.

II. MrsP

MrsP [11] is a spin-based locking protocol for partitioned fixed-priority systems. This protocol is created as a variant of MSRP: spin locks are adopted and resources are served in a FIFO order. However, unlike the non-preemptive approach, tasks under MrsP can be preempted during spinning and executing with a resource. Under MrsP, each resource has a ceiling priority on each processor that contains tasks requesting the resource. The ceiling priority on a given processor is set to be

the highest priority of the requesting tasks. Once a task tries to access a resource, it raises its priority to the ceiling during spinning and accessing the resource. With FIFO spinning, the length of the resource's waiting queue is the number of processors that contain tasks that request the resource. Yet spinning and executing with resources only at the ceiling level can lead to a prolonged blocking time as the resource holder can be preempted by higher priority local tasks.

To bound the waiting time, a helping mechanism is introduced where a spinning task can undertake the associated computation of any other task that request the same resource. The helping mechanism allows tasks to help the preempted resource holder to make progress by using the wasted cycles. In the worst case, a resource-requesting task will execute all the critical section computations of tasks in the FIFO queue each time it tries to access the resource, which leads to a worst-case resource accessing time of the FIFO queue length multiplied by the cost for accessing the resource. In [11], two possible approaches to realise the helping mechanism are supported: a task migration approach and a duplicated execution approach. The duplicated execution approach can only be applied to stateless resources so that migrations are usually required when implementing MrsP in general. In this work we focus on MrsP with migrations.

A. Original Response Time Analysis

The MrsP analysis is extended from the uniprocessor Response Time Analysis (RTA) framework [3] for the PCP/SRP case with minor modifications to reflect the parallel access to global resources:

$$R_i = \widehat{C_i} + B_i + \sum_{\tau_j \in hpl(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \widehat{C_j}$$

The response time R_i for τ_i is determined by its execution time $\widehat{C_i}$, the maximum blocking time B_i and the interference from higher-priority tasks, where $hpl(i)$ gives a set of high priority local tasks. $\widehat{C_i}$ is further determined by:

$$\widehat{C_i} = C_i + \sum_{r^k \in F(\tau_i)} N_i^k e^k$$

where N_i^k is the number of times τ_i uses the resource and r^k represents each resource that τ_i requires. To reflect potential parallel access for resources, e^k is introduced to represent the full execution time of a resource.

$$e^k = |map(G(r^k))| c^k$$

With the FIFO spin approach and the helping mechanism, the resource accessing time can be safely bounded by the number of processors that contain tasks that request the resource, where map returns a set of processors where the given tasks are assigned to and $||$ gives the size of a given set.

$$B_i = max\left\{\widehat{e}, \widehat{b}\right\}$$

The blocking term B_i is determined by the maximum value between the maximum execution time of resources that are used by low priority tasks and at least one equal or higher priority task ($\widehat{e}$) and the maximum duration of non-preemptive sections incurring within the operating system ($\widehat{b}$).

B. Discussion

The original MrsP analysis applies a compact approach and can be used even with a limited knowledge of the system (e.g., the exact resource usage of remote tasks). However, this analysis may account for a critical section more than once. Consider a three processors single resource system in Figure 1, where each task is assigned with a priority of its index value (e.g., $Pri(\tau_5) = 5$) and requests resource r^1. Suppose that during the release of any task in the system, other tasks will only be released once. For τ_3, which requests r^1 3 times, it can incur direct spin delay 3 times from P_1 and 2 times from P_3 by remote requests with same color, where $\widehat{C_3} = C_3 + 3c^1 + 3c^1 + 2c^1 = C_3 + 8c^1$, including requests form τ_3 itself. Yet applying MrsP analysis to τ_3 will account for one extra critical section, where $\widehat{C_3} = C_3 + 3 \times 3c^1 = C_3 + 9c^1$ as the analysis assumes that each time τ_3 accesses r^1 it incurs blocking from both P_1 and P_3 i.e., processors with tasks requesting the same resource. As for task τ_2, $\widehat{C_2} = C_2 + 9c^1$ if MrsP analysis is applied, yet τ_2 will only incur direct spin delay by 2 remote requests from $P1$ and its third request will not incur spin delay at all because other remote requests delay τ_3 directly (thus block τ_2 indirectly) and will be accounted for as part of the high priority task interference of τ_2.

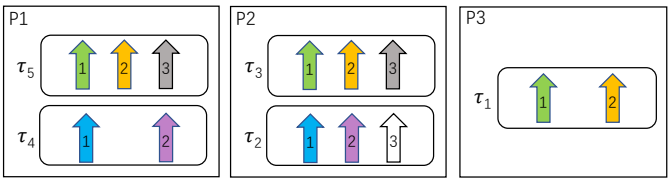

Fig. 1: Issues when bounding spin delay under MrsP

In addition, Wieder and Brandenburg [29] point out that inflating a task's computation time (i.e., using $\widehat{C}$ to bound the indirect spin delay) can also account for a request multiple times. Consider the same example, now we focus on τ_2 and assume that during τ_2's release τ_3 can be released (and preempt τ_2) 3 times so that $\left\lceil \frac{R_2}{T_3} \right\rceil = 3$ while other tasks are released once. Even with an accurate direct spin delay bounding, the interference of τ_2 is $3 \times \widehat{C_3}$, where $\widehat{C_3} = C_3 + 3c^1 + 5c^1$ as explained above. By doing this, the analysis assumes that each time when τ_3 is released in the context of τ_2, it can be blocked 5 times from P_1 and P_2, which is 15 blocking in total. However, as other tasks are released only once, there are at most 7 remote requests that can block τ_3's requests in its three releases so that 8 critical sections are over-calculated.

In the ILP-based analysis the spin delay is taken out of the tasks execution time (i.e., the task's computation time inflation approach is discarded) and all blocking effects are accounted in parameter B_i. With a set of constraints, the blocking effects for a task are bounded by the exact number of requests to each resource issued from each remote processor and local high

priority tasks with the principle that one remote request can only cause one blocking. For example, to analyse MSRP, an objective function is introduced as B_i to define the blocking variables (X^S for spin delay and X^A for arrival blocking) and 9 constraints are applied to bound these variables for each resource. In total, 30 constraints have being developed for eight spin locks. In addition, to accurately account for the number of requests that are issued during the release of τ_i, the *back to back hit* phenomenon is accounted for, where a task, say τ_x, can be released once during the lifetime of τ_i (i.e., $\left\lceil \frac{R_i}{T_x} \right\rceil = 1$) yet can cause one more blocking due to the resource accessing in its last release ($\left\lceil \frac{R_i+R_x}{T_x} \right\rceil = 2$). With such a design, the ILP-based analysis addressed the issues above and provides a less pessimistic as well as a more accurate analysis compared to other existing analysis [29].

III. Improving MrsP RTA

Due to the reasons stated in Section I-A, we created a new schedulability test explicitly for MrsP that overcomes the issues described in II-B but without the need for the potentially expensive ILP technique. Our improved MrsP RTA aims to provide an analysis with an identical degree of pessimism as the ILP-based analysis. In contrast to the ILP-based analysis, we aim to bound the three blocking effects identified in Section I-D separately and then fit them into the original MrsP RTA equations (without inflating the task's execution time) to facilitate the migration cost analysis in Section IV.

A. Modified MrsP Response-Time Equation

Equation 1 gives the response time of task τ_i, where the blocking effects are reflected by three parameter: E_i is the total resource accessing time of τ_i with direct spin delay accounted for; $I_{i,h}$ indicates the indirect spin delay incurred by τ_i from a local high priority task τ_h and the arrival blocking is accounted for in B_i. Note that in our new analysis, C_i is the pure computation time of τ_i without accessing any resource. Function $\left\lceil \frac{R_i}{T_h} \right\rceil \cdot C_h$ gives the pure computation interference from a local high priority task τ_h without accessing resources.

$$R_i = C_i + E_i + B_i + \sum_{\tau_h \in hpl(i)} \left(\left\lceil \frac{R_i}{T_h} \right\rceil \cdot C_h + I_{i,h} \right) \quad (1)$$

B. Direct and Indirect Spin Delay

We start by bounding the total resource accessing time with direct spin delay E and indirect spin delay I incurred by τ_i. These two equations share a similar format but take different inputs, as shown in equations 2 and 3, where $e_x^k(l,\mu)$ gives the accessing time (with direct spin delay) to resource r^k that task τ_x can incur within the duration l and a release jitter μ. By given different duration and jitter length, the function gives a different bounding as τ_x can be released a different number of times (so that a different number of requests) within the given duration. Accordingly, our analysis does not rely on inflating execution time.

$$E_i = \sum_{r^k \in F(\tau_i)} e_i^k(R_i, 0) \quad (2)$$

$$I_{i,h} = \sum_{r^k \in F(\tau_h)} e_h^k(R_i, R_h) \quad (3)$$

Equation 2 gives the total resource accessing time of τ_i. For τ_i itself, $l = R_i$ and $\mu = 0$ so that we will only account for resource requests in one release. As for the indirect spin delay (equation 3), $l = R_i$ and $\mu = R_h$ so that the back-to-back hit can be accounted for when computing the total number of requests issued from a high priority task τ_h to r^k in the context of τ_i (i.e., during τ_i's release). To facilitate the migration cost analysis, we analyse the resource accessing time of a task in each individual access so that $e_x^k(l,\mu)$ is further expanded as:

$$e_x^k(l,\mu) = \sum_{n=1}^{N_x^k(l,\mu)} e_x^k(l)(n) \quad (4)$$

where $N_x^k(l,\mu) = \left\lceil \frac{l+\mu}{T_x} \right\rceil \cdot N_x^k$ gives the number of requests τ_x can issue to resource r^k with the back to back hit and $e_x^k(l)(n)$ gives the time of τ_x's n-th access to r^k within a duration l.

To reflect the worst case scenario, a higher priority task should incur blocking before any low priority tasks do, as the spin delay incurred by high priority tasks is propagated to all local lower priority tasks as interference. Thus, when computing the direct spin delay that τ_x can incur for accessing r^k, the requests from a remote processor should delay τ_x's higher priority tasks a prior to τ_x, which leads to the following observations, where $Nh_x^k(l) = \sum_{\tau_h \in hpl(x)} N_h^k(l, R_h)$ gives the number of requests issued by local high priority tasks, $Np_m^k(l) = \sum_{\tau_j \in \tau(P_m)} N_j^k(l, R_j)$ gives the number of requests issued from a remote processor m, $\tau(P_m)$ gives a set of tasks allocated on processor m and $(f(x))_a$ denotes $max\{f(x), a\}$ for the ease of presentation.

Theorem 1. *The maximum number of requests on a remote processor m that **may** block τ_x directly for accessing r^k within the duration l is bounded by $NS_{x,m}^k(l) = (Np_m^k(l) - Nh_x^k(l))_0$.*

Proof. Let N_S^{may} denote the number of requests from a remote processor that may block τ_x. If $N_S^{may} > NS_{x,m}^k(l)$, then there exist remote requests that can block both τ_x and a higher priority task on τ_x's processor that requests r^k directly, which is not possible as one request can only cause one blocking. Otherwise (where $N_S^{may} < NS_{x,m}^k(l)$), certain requests that may block τ_x are not accounted for. □

Theorem 2. *The number of direct spin delays that τ_x **can** incur for accessing r^k from a remote processor m within the duration l and jitter μ is $min\{NS_{x,m}^k(l), N_x^k(l,\mu)\}$.*

Proof. Let N_S^{can} denote the number of spin delay that τ_x can incur. If $N_S^{can} = NS_{x,m}^k(l) \ \wedge \ NS_{x,m}^k(l) > N_x^k(l,\mu)$, there exists a remote request that can block τ_x multiple times. In contrast, where $N_S^{can} = N_x^k(l,\mu) \ \wedge \ N_x^k(l,\mu) > NS_{x,m}^k(l)$, there exist more than one requests on a remote processor that can block the same access of τ_x. Under MrsP, neither case is possible. □

To examine the blocking in each access, we assume that *the first access to a resource incurs as much spin delay as possible*. This assumption will not introduce any pessimism as the total spin delay a task can incur remains identical. Accordingly, equation $e_x^k(l)(n)$ can be constructed to compute the time for each access (see equation 5), where n is bounded to [1, $N_x^k(l,\mu)$] by equation 4 and one extra c^k is accounted for the access by τ_x itself. For the ease of presentation, let $(f(x))_a^b$ denote $min\{max\{f(x), a\}, b\}$, where a and b are positive integers with $a \leq b$.

$$e_x^k(l)(n) = \sum_{P_m \neq P(\tau_x)} (NS_{x,m}^k(l) - n + 1)_0^1 \cdot c^k + c^k \quad (5)$$

Proof. In τ_x's n-th access, requests from a remote processor m can block τ_x only if there still exists unaccounted requests on m i.e., $(NS_{x,m}^k(l) - n + 1)_0 >= 1$. Upon one access, there can be at most one request on a remote processor that can cause the spin delay and hence $(NS_{x,m}^k(l) - n + 1)_0^1$. □

With equations 4 and 5, the direct spin delay in E and the indirect spin delay I can be computed. As proved, our approach accounts for each critical section only once and does not rely on inflating task's computation time so that the issues discussed in Section II-B are addressed. In addition, with the back to back hit considered, the new equations provide less pessimism and more accurate spin delay bounding than that of the original MrsP analysis. In contrast to the ILP-based analysis (which only gives a total amount of spin delay for each task), our approach is able to compute the delay of each individual access. Analysing the spin delay of each access to each resource seems unnecessary for this work but is fundamental for the migration cost analysis in Section IV.

C. Arrival Blocking

The arrival blocking is accounted for by parameter B_i (equation 6), where $\widehat{e_i}$ gives the maximum arrival blocking that τ_i can incur and is calculated by equation 7.

$$B_i = max\{\widehat{e_i}, \widehat{b}\} \quad (6)$$

$$\widehat{e_i} = max\{|\alpha_i^k| \cdot c^k | r^k \in F^A(\tau_i)\} \quad (7)$$

Equation 7 firstly identifies resources that can cause τ_i to incur arrival blocking $F^A(\tau_i)$ and then gives the maximum blocking time among the resources in $F^A(\tau_i)$. Under MrsP, a resource r^k can cause arrival blocking to τ_i if it has a higher or equal ceiling priority on τ_i's processor $Pri(r^k, P(\tau_i))$ and will be accessed by local lower priority tasks τ_{ll}, where $F^A(\tau_i) \triangleq \{r^k | N_{ll}^k > 0 \wedge Pri(r^k, P(\tau_i)) \geq Pri(\tau_i)\}$.

The arrival blocking can be computed without the knowledge of the exact task that causes such a blocking. For any resource (either local or global) in $F^A(\tau_i)$, it can cause a local blocking of c^k. For a global resource r^k, there can be at most one request from each remote processor that can cause τ_i to incur arrival blocking transitively. Therefore, by identifying the number of such processors, the arrival blocking can be computed. Let $P(\tau_i)$ denote τ_i's processor and α_i^k be the set of processors with requests to r^k that cause arrival blocking to τ_i (including $P(\tau_i)$), where

$$\alpha_i^k \triangleq \{P_m | NS_{i,m}^k(R_i) - N_i^k > 0 \wedge P_m \neq P(\tau_i)\} \cup P(\tau_i) \quad (8)$$

Proof. Similar to the proof of equation 5, a request to r^k from a remote processor can block a lower priority task on τ_i's processor only if the remote request does not cause any delay yet (including τ_i) i.e., $NS_{x,m}^k(l) - N_i^k > 0$. Otherwise (where $NS_{x,m}^k(l) - N_i^k \leq 0$), this remote request (if any) will be calculated more than once because it is already accounted for in the spin delay of τ_i. □

With α_i^k computed for each resource in $F^A(\tau_i)$, the arrival blocking of τ_i is obtained with one extra access included to represent the local blocking, as shown in equation 7.

D. Summary

This concludes the new MrsP Response-Time Analysis. This analysis is independent of the priority assignment scheme and is not fixed to any specific hardware architecture. Similar to Wieder and Brandenburg's analysis, the blocking time of a given task in our analysis depends on the response time of potentially all tasks in the system. With an initial response time, say C_i, the analysis computes the blocking variables (E, I and B) and an updated response times of all tasks in the system iteratively and alternately until a fixed-point is reached.(i.e., the response time and the blocking variables remain the same after further calculations).

IV. Migrations in MrsP and Analysis

With MrsP applied, tasks are allowed to migrate with resources using the helping mechanism. If a resource holder is being preempted, it can migrate to a remote processor where there is an executing spinning task requesting for the same resource. Once being preempted again, the holder can then migrate either back to its original processor (if the preemptor is finished) or to another valid processor (if any). After the holder releases the resource, it will migrate back to its original processor (if necessary).

In theory, the helping mechanism is attractive because it guarantees an identical resource waiting queue as MSRP does. Previously, the migration cost has been treated as run-time overheads and not considered in schedulability analysis. However, in practice, migrations usually require updating operating system structures (e.g., run queues) and the reloading of caches, which carry non-negligible cost. Accordingly, once a resource holder requires migrations (i.e., is preempted), the migration cost that the holder incurs can increase the resource accessing time, which will reduce schedulability.

The analysis presented in Section III does not account for the overheads due to migrations. Indeed, we are not aware of any analysis for multiprocessor resource control that includes the overheads of protocols that support migration. Admittedly, the actual migration cost a task can incur largely depends on real hardware platforms and operating systems. Yet by treating the migration cost as a constant upper bound (e.g., mig in

our work), the maximum number of times a task can migrate during one access to a resource can be obtained and hence the migration cost can be bounded. In this section, we developed further analysis that bounds the migration cost in MrsP. The objective is to integrate this with the analysis in Section III to provide a migration-cost aware MrsP schedulablility analysis.

Before presenting the analysis, we discuss the difficulty in bounding the number of migrations that can occur and propose a modification to the MrsP helping mechanism to allow more efficient migrations.

A. The Problem of Frequent Migrations

The current definition of the MrsP helping mechanism carries a certain degree of pessimism under the situation where the resource holder is preempted and there are a large number of potential helpers each of which reside in processors where there is one or more high priority tasks with short periods. As we will show in Section IV-B, this can result in the resource holding task suffering frequent migrations. Under such a situation, the task can spend more time migrating than it does executing with the resource so that the resource accessing time can be significantly prolonged and the efficiency of the protocol can be undermined.

To avoid frequent migrations, we introduce a short non-preemptive section into the MrsP helping mechanism so that *upon each migration with a resource, the holder is allowed to execute non-preemptively (NP) immediately for a short time before it inherits the corresponding resource ceiling priority*. The NP-section can provide guaranteed progress to resource holders and can reduce the number of migrations effectively, especially when high priority tasks are released frequently. The only side effect of this approach is that any newly released high priority tasks have to cope with the cost of one NP section before it can preempt the holder and execute. However, the length of the NP section can be configured so that the high priority tasks are still able to meet their deadlines. As a default it can be the maximum time of the NP-sections in the hosting operating system (symbol $\widehat{b}$ in Section II). Our analysis presented below bounds the cost of the migration with this approach. In Section V, evidence is given to demonstrate improved efficiency when this approach is adopted.

B. Migration Cost Analysis

To capture the worst-case scenario, we assume that *a preempted resource holder can migrate to any valid processor* (i.e., a processor that has a task spinning for the resource or the holder's original processor). In addition, as shown in the analysis from Section III, for any resource-requesting task τ_x, it can incur a different amount of spin delay upon each access to a resource so that its migration targets can also be different during each resource access. Thus, the migration cost should be computed by each individual access to each resource. We firstly identify the set of migration targets for a given task τ_x.

Theorem 3. *In τ_x's n-th access to r^k within a duration l, the set of migration targets for τ_x is $mt_x^k(l)(n) \triangleq \{P_m | P_m \neq P(\tau_x) \wedge NS_{x,m}^k(l) - n + 1 > 0\} \cup P(\tau_x)$.*

Proof. A remote processor m is a valid migration target for τ_x's n-th access to r^k only if there exists a request to r^k from processor m that is not already accounted for during l (i.e., $NS_{x,m}^k(l) - n + 1 > 0$ from equation 5). In addition, τ_x's original processor should be included as τ_x may migrate back to $P(\tau_x)$ when it is preempted on a remote processor. □

When τ_x incurs arrival blocking by a low priority task, the blocking task may also incur migration cost, which in turn delays τ_x. The migration targets of the low priority task can be identified by the set α_x^k (the set of remote processors with requests that can cause τ_x to incur arrival blocking) in equation 8.

As the resource accessing task inherits the resource ceiling when accessing the resource, the potential preemptors on each migration target can be identified. With a given set of migration targets (denoted by mt) and a resource r^k, the migration targets with preemptors $mtp(mt, r^k)$ is:

$$mtp(mt, r^k) \triangleq \{P_m | P_m \in mt \wedge hpt(r^k, P_m) \neq \emptyset\} \quad (9)$$

where $hpt(r^k, P_m)$ gives a set of tasks on processor m that have a priority higher than the resource ceiling of r^k on P_m. Note $mtp(mt, r^k)$ is a subset of the given migration targets mt and can be empty.

As presented above, migration targets are identified based on whether there will be a request from the remote processor. Thus, on each migration target, there exists one request issued to the resource and they share the same set of migration targets. To bound the migration overhead a task τ_x can incur when accessing a resource, we examine the migration cost of each request issued from the migration targets. Let $Nmig$ be the number of potential migrations. We summarise the following observations where a limited number of migrations can be triggered when a request is issued from processor P_m to resource r^k with a given set of migration targets mt:

Lemma 1. *$Nmig = 0$ if $P_m \notin mtp(mt, r^k)$.*

Proof. The request issued from processor P_m incurs no migrations if there exists no preemptors on that processor. □

Lemma 2. *$Nmig = 0$ if $\{P_m\} = mt$.*

Proof. No matter how many times the request from P_m can be preempted on its processor, there will be no migrations if there exists no other migration targets. □

Lemma 3. *$Nmig = 2$ if $\{P_m\} = mtp(mt, r^k) \wedge |mt| > 1$.*

Proof. In the case where the request can only be preempted on its original processor P_m, the requesting task can migrate to other migration targets without further preemptions. Once the task releases the resource, it migrates back to P_m. □

In a more general case where there exist more than one migration targets with potential preemptors, the number of migrations have to be bounded by the release of all potential preemptors. Unfortunately, we are not able to track the state of the current processor of the resource holder constantly as no assumption can be made about the migration destination. Thus,

we have to assume that each release of the high priority task can cause a preemption with a subsequent migration. Because of this, *our analysis provides a safe upper bound of the migration cost rather than a precise worst-case bounding*. However, by applying the NP section and by identifying the exact set of migration targets, the pessimism of the analysis can be largely reduced, as shown in experiment (d) in Section V-B.

In the case where the resource-requesting task's processor $P_m \in mtp(mt, r^k) \wedge |mt| > 1$, the migration cost of that single request is bounded by the releases of high priority tasks on each migration target, denoted by $Mhp(mt, r^k)$, where mig denotes the overheads of one migration.

$$Mhp(mt, r^k) = mig \cdot \Big(\sum_{P_m \in mtp(mt,r^k)} \big(\sum_{\tau_h \in hpt(r^k, P_m)} \left\lceil \frac{c^k + Mhp(mt, r^k)}{T_h} \right\rceil \big) + 1 \Big) \tag{10}$$

The equation accounts for the total number of releases of all the potential preemptors on each migration targets within the duration of one resource computation time with migration cost considered $c^k + Mhp(mt, r^k)$. Through iteration, the equation can give a fixed migration cost that the requesting task can incur based on the given set of migration targets. To cope with the situation where the next holder needs to wait for the current holder to migrate away before it can acquire the resource, one extra migration is included.

On the other hand, with the NP section adopted, the migration cost in a single access can also be bounded by the length of the NP sections, denoted by Mnp^k (equation 11), where C_{np} represents the length of the NP section. Note that in our analysis we assume the length of NP section as a positive integer value (by default $C_{np} = \widehat{b}$)

$$Mnp^k = mig \cdot \left(\left\lceil \frac{c^k}{C_{np}} \right\rceil + 1 \right) \tag{11}$$

In the case where the holder can be preempted frequently, this equation can give a more acceptable number of migrations a holder can incur. Unlike equation 10, this equation does not rely on iterations as the NP section is for the resource execution only and does not include the cost of migrations. Therefore, $\left\lceil \frac{c^k}{C_{np}} \right\rceil$ provides an safe bounding on number of migrations with NP section applied. Combing equations 10 and 11, gives the following lemma, where the request is issued from processor m:

Lemma 4. *$Nmig = min\{Mhp(mt, r^k), Mnp^k\}$ if $P_m \in mtp(mt, r^k) \wedge |mtp(mt, r^k)| > 1$.*

Proof. In the case where $Mnp^k < Mhp(mt, r^k)$, the resource holder is protected by the NP section while some of the preemptions are delayed so that $Nmig = Mnp^k$. In contrast (where $Mhp(mt, r^k) \leq Mnp^k$) the holder often can execute for an amount of time longer than C_{np} after migrations without the effect of NP sections. Thus, $Nmig = Mhp(mt, r^k)$. □

Combining Lemma 1 to 4, we give the total migration cost a task can incur. In the worst-case, the task has to cope with the migration cost of all the requests in the FIFO queue. Let $Mig(mt, r^k)$ be the total migration cost that a task can incur for accessing r^k with a given set of migration targets mt:

$$Mig(mt, r^k) = \sum_{P_m \in mt} \begin{cases} 0, & \text{if } P_m \notin mtp(mt, r^k) \vee \{P_m\} = mt \\ 2 \cdot mig, & \text{if } \{P_m\} = mtp(mt, r^k) \wedge |mt| > 1 \\ min\{Mhp(mt, r^k), Mnp^k\}, & \text{otherwise} \end{cases} \tag{12}$$

With the migration cost analysis constructed, we integrate this with the analysis presented in Section III to form a complete MrsP schedulability analysis. Firstly, the migration cost should be integrated into the equation that bounds the spin delay (see equations 4 and 13). The set of migration targets are identified previously by $mt_x^k(l)(n)$.

$$e_x^k(l, \mu) = \sum_{n=1}^{N_x^k(l,\mu)} (e_x^k(l)(n) + Mig(mt_x^k(l)(n), r^k)) \tag{13}$$

In addition, the migration cost needs to be accounted for in equation 7, where the arrival blocking is bounded. The set of migration targets here are given by α_i^k. Equation 14 gives the arrival blocking with migration cost accounted for. In the case where r^k is a local resource, $Mig(\alpha_i^k, r^k) = 0$ as $\alpha_i^k = \{P(\tau_i)\}$.

$$\widehat{e_i} = max\{|\alpha_i^k| \cdot c^k + Mig(\alpha_i^k, r^k) | r^k \in F^A(\tau_i)\} \tag{14}$$

Finally, as we adopt the NP-section, an extra blocking effect should be accounted for. If the length of the NP section is configured as the maximum NP section length in the hosting operating system ($\widehat{b}$), no further modifications to the equations are required. Otherwise (where $C_{np} > \widehat{b}$), for any given task τ_i, it has the risk to incur such a blocking (denoted by $\widehat{np_i}$) as long as it has a priority equal or higher than the lowest ceiling priority of global resources on its processor:

$$\widehat{np_i} = \begin{cases} C_{np}, & \text{if } Pri(\tau_i) \geq min_{\{r^k \text{ is global}\}} Pri(r^k, P(\tau_i)) \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Same with the arrival blocking, such a blocking happens before the execution of τ_i and can only happen once, equation 6 should be modified to reflect this extra blocking.

$$B_i = max\{\widehat{e_i}, \widehat{np_i}, \widehat{b}\} \tag{16}$$

This concludes the work of MrsP Schedulability Analysis. In next section, a set of evaluations are performed to investigate the schedulability of MrsP.

V. Evaluation

In this section, we performed a set of experiments to compare the schedulability between MrsP and other FIFO spin lock-based protocols (both preemptive or non-preemptive models) with different configurations (e.g., work load and critical section length). We first investigate the theoretical schedulability and do not consider any overheads resulting

from the protocols. Thus, analysis in Section III is applied when evaluating MrsP with other protocols by assuming the migration cost is 0. We then study the impact of migration cost and compare the efficiency (and therefore the resulting schedulability) of the MrsP helping mechanism with (and without) the NP-section by the analysis created in Section IV-B.

The code for the evaluations performed in this section can be accessed via https://github.com/RTSYork/MrsPAnalysisTest. In this work, the new MrsP Response-Time Analysis presented in Section III and the complete schedulability test described in Section IV are implemented. In addition, a system generation tool is developed to generate systems with different tasks and resource usage configurations. To compare the schedulability of MrsP with other protocols, we integrate the implementation of the ILP-based analysis from the SchedCAT project [1] via JNI to provide the analysis of other FIFO spin locks.

A. System Setting

We consider platforms with $m = [2, 16]$ processors. On each processor, there can be up to 10 tasks, where $n = [1, 10]$. As we focus on fully partitioned systems, the tasks are generated on each individual processor without the need of partitioning. Periods of tasks on each processor are randomly chosen between $[1ms, 1000ms]$ in a log-uniform distribution fashion. In this evaluation, we assume that the deadline of the tasks are equal to their periods ($D = T$). The utilisation of each task is computed based on the UUnifast-Discard algorithm proposed by Bini and Buttazzo [5] and hence the execution time with resources ($\widehat{C}$) for each task can be computed. The system supports 1000 priority levels. The priority of each task in a processor is given using the rate-monotonic policy [25].

A wide range of critical section length (L): $[1\mu s, 15\mu s]$, $[15\mu s, 50\mu s]$, $[50\mu s, 100\mu s]$, $[100\mu s, 200\mu s]$ and $[200\mu s, 300\mu s]$ is supported. Then a real value parameter κ is introduced to specify the number of tasks on each processor that can access to resources (i.e., $\lfloor \kappa \cdot n \rfloor$), where $\kappa \in$ [0.0,1.0]. A task will issue requests to a number of randomly chosen resources (but limited by [1,m]). The number of requests is randomly decided between $[1, A]$, where $A = [1, 41]$. Let C_x^r be the total resource computation time of τ_x. For a given task τ_x, with its resource usage generated, the pure computation time C_x can be computed, where $C_x = \widehat{C}_x - C_x^r$. We enforce that $\widehat{C}_x - C_x^r \geq 0$.

In practice, modern operating systems and hardware (with a typical three-level cache topology) usually have a migration cost less than $10\mu s$, where the scheduling and context switch procedure will be invoked with the need for updating run-queue structure and cache. As observed by [13], the cost of one migration is 6000 ns in LITMUS^{RT} [12], [8] and is 5000 ns in RTEMS [23] on an Intel Quad Core i7-2670QM with 2.2GHz and a three-level cache memory. In this work we set the cost of one migration as 6000 ns (i.e., $mig = 6\mu s$).

B. Schedulability Evaluation

We investigate the schedulability of MrsP and other FIFO spin protocols under systems with various (a) work load on each processor n; (b) parallelism m; (c) critical section length L and (d) resource contention A. We focus on the FIFO spin locks to provide a view of how MrsP performs compared to the spin locks with similar features. The schedulability tests implement the following analysis: the original MrsP analysis [11] (MrsP-original); the original MSRP analysis [16] (MSRP); the ILP-based MSRP analysis [29] (FIFO-NP), the ILP-based FIFO preemptive spin locks analysis (FIFO-P) [29] and our new MrsP Response-Time Analysis (MrsP-new) from Section III. Each system setting is tested by 1000 systems.

(a) *Varying n and m:* With a low resource contention ($A = 2, \kappa = 0.4$) and short critical section length ($L = [1\mu s, 15\mu s]$), MrsP does not have an obvious schedulability difference between other spin locks, as shown in Figure 2. By incrementing n, the original analysis for MrsP (and MSRP) gives a much lower schedulability than that of our new analysis (and the FIFO-NP analysis). When further incrementing n, MrsP shows a slightly better schedulability than both FIFO-NP and FIFO-P do. A similar trend between MrsP and FIFO-NP is observed when increasing m (see Figure 3). However, FIFO-P in this experiment offers the best schedulability when $m \in \{8, 10, 12, 14\}$ due its relatively low arrival blocking. Yet with a further increasing of m , both MrsP and FIFO-NP give a better schedulability than that of FIFO-P (when $m \geq 16$) as the spin delay can be bounded to m.

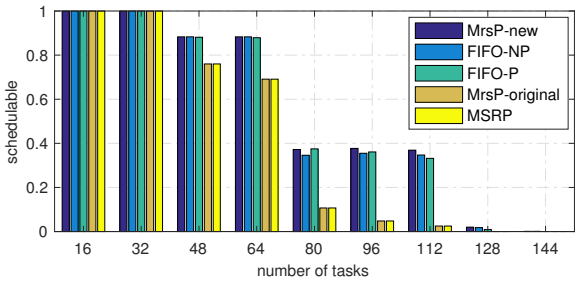

Fig. 2: Schedulability for $m = 16$, $U = 0.1n$, $\kappa = 0.4$, $A = 2$, $L = [1\mu s, 15\mu s]$, and m shared resources.

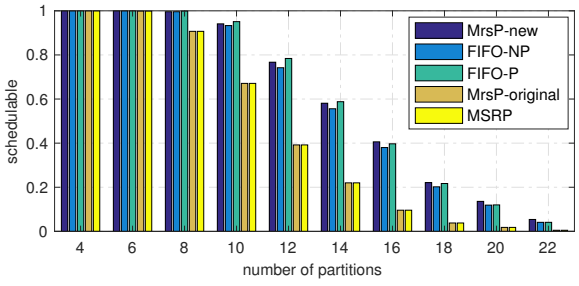

Fig. 3: Schedulability for $n = 5$, $U = 0.1n$, $\kappa = 0.4$, $A = 2$, $L = [1\mu s, 15\mu s]$, and m shared resources.

(b) *Varying A:* As shown in Figure 4, FIFO-P has the best schedulability and FIFO-NP is worse than MrsP and FIFO-P in the case where $A = 1$, as tasks incur limited preemptions within a short resource accessing time and such a cost is

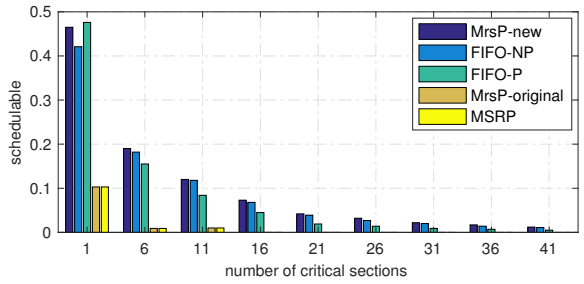

Fig. 4: Schedulability for $m = 16$, $n = 5$, $U = 0.1n$, $\kappa = 0.4$, $L = [1\mu s, 15\mu s]$, and m shared resources.

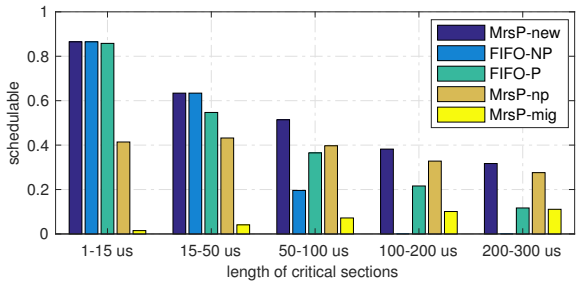

Fig. 6: Schedulability for $m = 16$, $n = 4$, $U = 0.1n$, $\kappa = 0.4$, $A = 3$ and m shared resources.

more likely to be less than the arrival blocking that tasks under FIFO-NP or MrsP can suffer. However, with a further increment (and an increased risk to be preempted), FIFO-P becomes the worst with a observable difference compared to the other two protocols.

(c) *Varying L:* With an increasing length of critical sections, we observed the schedulability of FIFO-NP locks decreases dramatically while MrsP provides the best schedulability among all tested locks (see Figure 5). With FIFO-NP, the highest priority tasks have to cope with the largest arrival blocking, and hence, can easily miss their deadlines if long critical sections are adopted. In contrast, although with a longer spin delay, FIFO-P locks only incur a local blocking so that can offer a higher schedulability than that of FIFO-NP. Under MrsP, tasks can incur a limited amount of arrival blocking due to the ceiling facility and can have a shorter spin delay than that of FIFO-P. Thus, MrsP can offer a better schedulability with long critical sections than both FIFO-NP and FIFO-P can achieve.

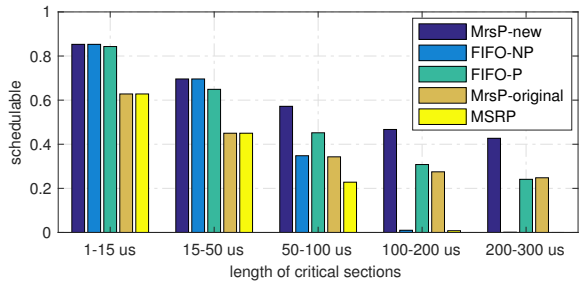

Fig. 5: Schedulability for $m = 16$, $n = 4$, $U = 0.1n$, $\kappa = 0.4$, $A = 3$ and m shared resources.

(d) *Migration Cost Analysis*: Now we study the impact of accounting for the overheads of migrations on the theoretical schedulability with the analysis in Section IV under various critical section length. In addition, we present evidence of an improved efficiency of MrsP by the controlled migration behaviour with the NP section adopted. The analysis used in this experiment is (1) the analysis without migration cost ("MrsP-new"); (2) the ILP-based FIFO-NP analysis; (3) the ILP-based FIFO-P analysis; (4) the migration cost analysis with the NP section adopted ("MrsP-np") and (5) the migration cost analysis without NP sections ("MrsP-mig"). The analysis MrsP-mig is modified from the analysis in Section IV by taking Mnp^k and $\widehat{np_i}$ out of equations 12 and 16 respectively. When MrsP-np is in use, the length of the NP sections are set differently based on the system settings, and hence is not presented. As described for, this length can be tuned for each individual system to achieve the best schedulability.

Compared to FIFO-NP and FIFO-P, MrsP with migration cost accounted for seems to be less favourable when applied to short critical sections (e.g., $15\mu s$) as one single migration costs $6\mu s$ in our case. This is proved by Figure 6, where MrsP-np provides a low schedulability with $L \in [1\mu s, 50\mu s]$. However, when $L > 50\mu s$, MrsP-np shows a better schedulability than both FIFO-NP and FIFO-P can achieve, which again proves that MrsP works better with long critical sections. In addition, we observe that with migration cost accounted for, there exist an obvious difference between the schedulability of MrsP-new and MrsP-np, which reveals the necessity to include such a cost into the schedulability analysis. Further, as observed, MrsP-mig (without the protection of NP sections) has a lower schedulability than that of MrsP-np (the one with NP sections applied) in all cases, which demonstrates an improved efficiency by integrating a short NP section into MrsP protocol.

C. Summary

From the experiments we observed that theoretically MrsP offers a better (at least identical) schedulability than MSRP in all cases because both protocols have an identical spin delay but MrsP guarantees a shorter arrival blocking at most times. In addition, as observed, both FIFO-NP and MrsP are less efficient than FIFO-P in systems with low resource contention or less partitions due to adopting either the non-preemptive accessing or the resource ceiling facility approach. With migration cost accounted for, the schedulability analysis of MrsP is significantly reduced and can be outperformed by protocols with no migrations. However, with long critical sections in use, both the theoretical RTA or the migration-cost-aware schedulability test provide clear evidence that MrsP outperforms both FIFO-NP and FIFO-P protocols.

Admittedly, one can argue that for long critical sections suspension-based locks should be applied. However, as re-

vealed by the experiments, both the FIFO-P and MrsP protocols can be considered applicable to long critical sections by offering an acceptable schedulability ratio, where MrsP gives a better schedulability.

VI. Conclusion

In this paper we developed a new Response-Time Analysis for MrsP that incorporates more knowledge of an application's behaviour than that previously assumed in the original work. The new analysis achieves an identical degree of pessimism as the ILP-based analysis does, which similarly requires such knowledge. Our new analysis is more in keeping with the original MrsP philosophy but without the need for the potentially expensive ILP techniques. Theoretically, the new MrsP analysis offers better (at least identical) schedulability than the FIFO non-preemptive spin-based locking protocol, and can outperformed FIFO preemptive spin locks under systems with an intensive resource contention.

This paper has also developed analysis to include the impact of migrations. Although MrsP's helping mechanism theoretically increases schedulability, our evaluation shows that this increase may be negated when the overheads of migration are taken into account. To mitigate this, we have modified the MrsP protocol to introduce a NP section following migration. This ensures that a preempted resource holding tasks can make progress and can only incur limited migrations. Our experiment reveals direct impact on the schedulability of MrsP with the migration cost considered and an improved efficiency with the integration of the NP sections. Most importantly, we demonstrate that with migration cost, MrsP may be less favourable for short critical sections but can offer a strong schedulability under long critical sections, where traditional FIFO spin locks have low schedulability.

Our future work will address non-uniform resource access times and migration-aware nested requests. In addition, we aim to explore and develop a technique to provide a more precise bounding for the migration cost analysis. For instance, migrations are not worth performing where the migration cost is bigger than the interference from a preemptor. Further, a study of the priority assignment rule and task allocation scheme that can benefit MrsP will be investigated.

References

[1] "SchedCAT: Schedulability test collection and toolkit". http://www.mpi-sws.org/bbb/projects/schedcat. Accessed: 2016-11-1.

[2] Jaume Abella, Francisco J Cazorla, Eduardo Quiñones, Arnaud Grasset, Sami Yehia, Philippe Bonnot, Dimitris Gizopoulos, Riccardo Mariani, and Guillem Bernat. Towards improved survivability in safety-critical systems. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 240–245. IEEE, 2011.

[3] Neil Audsley, Alan Burns, Mike Richardson, Ken Tindell, and Andy Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[4] Theodore P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.

[5] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[6] Alessandro Biondi, Björn B Brandenburg, and Alexander Wieder. A blocking bound for nested fifo spin locks. In *Real-Time Systems Symposium (RTSS), 2016 IEEE*, pages 291–302. IEEE, 2016.

[7] Aaron Block, Hennadiy Leontyev, Björn B Brandenburg, and James H Anderson. A flexible real-time locking protocol for multiprocessors. In *RTCSA 2007, 13th IEEE*, pages 47–56. IEEE, 2007.

[8] Björn B Brandenburg. *Scheduling and locking in multiprocessor real-time operating systems*. PhD thesis, University of North Carolina at Chapel Hill, 2011.

[9] Alan Burns, Marina Gutierrez, Mario Aldea Rivas, and Michael González Harbour. A deadline-floor inheritance protocol for EDF scheduled embedded real-time systems with resource sharing. *IEEE Transactions on Computers*, 64(5):1241–1253, 2015.

[10] Alan Burns and Andy Wellings. Locking policies for multiprocessor Ada. *ACM SIGAda Ada Letters*, 33(2):59–65, 2013.

[11] Alan Burns and Andy Wellings. A schedulability compatible multiprocessor resource sharing protocol– MrsP. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 282–291. IEEE, 2013.

[12] John M Calandrino, Hennadiy Leontyev, Aaron Block, UmaMaheswari C Devi, and James H Anderson. Litmusˆ rt: A testbed for empirically comparing real-time multiprocessor schedulers. In *Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International*, pages 111–126. IEEE, 2006.

[13] Sebastiano Catellani, Luca Bonato, Sebastian Huber, and Enrico Mezzetti. Challenges in the implementation of MrsP. In *Reliable Software Technologies–Ada-Europe 2015*, pages 179–195. Springer, 2015.

[14] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

[15] Dario Faggioli, Giuseppe Lipari, and Tommaso Cucinotta. The multiprocessor bandwidth inheritance protocol. In *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, pages 90–99. IEEE, 2010.

[16] Paolo Gai, Giuseppe Lipari, and Marco Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 73–83. IEEE, 2001.

[17] Jorge Garrido, Shuai Zhao, Alan Burns, and Andy Wellings. Supporting nested resources in MrsP. In *Ada-Europe International Conference on Reliable Software Technologies*. Springer, 2017.

[18] Wen-Hung Huang, Maolin Yang, and Jian-Jia Chen. Resource-oriented partitioned scheduling in multiprocessor systems: How to partition and how to share? In *Real-Time Systems Symposium (RTSS), 2016 IEEE*, pages 111–122. IEEE, 2016.

[19] Giuseppe Lipari, Gerardo Lamastra, and Luca Abeni. Task synchronization in reservation-based real-time systems. *Computers, IEEE Transactions on*, 53(12):1591–1601, 2004.

[20] Ragunathan Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Distributed Computing Systems, 1990. IEE Proceedings., 10th International Conference*, pages 116–123, 1990.

[21] Ragunathan Rajkumar. *Synchronization in real-time systems: a priority inheritance approach*, volume 151. Springer Science & Business Media, 2012.

[22] Ragunathan Rajkumar, Lui Sha, and John P Lehoczky. Real-time synchronization protocols for multiprocessors. In *RTSS*, volume 88, pages 259–269, 1988.

[23] C RTEMS. Users guide-edition 4.6. 5, for rtems 4.6. 5. *On-Line Applications Research Corporation (OAR)-http://www. 1tems. com*, 30, 2003.

[24] Lui Sha, Ragunathan Rajkumar, and John P Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on computers*, 39(9):1175–1185, 1990.

[25] Lui Sha, Ragunathan Rajkumar, and Shirish S Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, 1994.

[26] Hiroaki Takada and Ken Sakamura. A novel approach to multiprogrammed multiprocessor synchronization for real-time kernels. In *The 18th IEEE Real-Time Systems Symposium Proceedings*, pages 134–143, 1997.

[27] B Ward and J Anderson. Nested multiprocessor real-time locking with improved blocking. In *Proceedings of the 24th Euromicro conference on real-time systems*, 2012.

[28] Bryan C Ward and James H Anderson. Supporting nested locking in multiprocessor real-time systems. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 223–232. IEEE, 2012.

[29] Alexander Wieder and Björn B Brandenburg. On spin locks in autosar: Blocking analysis of fifo, unordered, and priority-ordered spin locks. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 45–56. IEEE, 2013.