Resource Sharing among Real-Time Components under Multiprocessor Clustered Scheduling

Farhang Nemati and Thomas Nolte
Mälardalen Real-Time Research Centre
Mälardalen University
P.O. Box 883, SE-721 23, Västerås, Sweden
{farhang.nemati, thomas.nolte}@mdh.se

***Abstract*—In this paper we generalize our recently presented synchronization protocol (MSOS) for resource sharing among independently-developed real-time systems (real-time components) on multi-core platforms. Each component is statically allocated on a dedicated subset of processors (cluster) whose tasks are scheduled by its own scheduler. In this paper we focus on multiprocessor global fixed priority preemptive scheduling algorithms to be used to schedule the tasks of each component on its cluster. Sharing the local resources (only shared by tasks within a component) is handled by the Priority Inheritance Protocol (PIP). For sharing the global resources (shared across components) we have studied the usage of FIFO and Round-Robin queues for access across the components and the usage of FIFO and prioritized queues within components for handling sharing of these resources. We have derived schedulability analysis for the different alternatives and compared their performance by means of experimental evaluations. Finally, we have formulated the integration phase in the form of a nonlinear integer programming problem whose techniques can be used to minimize the total number of required processors by all components.**

I. Introduction

Looking at industrial systems, to speed up their development, it is not uncommon that large and complex systems are divided into several semi-independent subsystems (components) which are developed in parallel. In order to guarantee correct function of these systems, scheduling techniques are used to enforce predictable execution of subsystems. However, the emergence of multi-core architectures introduce challenges in allowing for an efficient and predictable execution of industrial software systems.

Two main approaches for scheduling real-time systems on multiprocessors (multi-cores) exist; global and partitioned scheduling [1], [2]. Under global scheduling, e.g., Global Earliest Deadline First (G-EDF), tasks are scheduled by a single scheduler and each task can be executed on any processor. Under partitioned scheduling, tasks are statically assigned to processors and tasks within each processor are scheduled by a uniprocessor scheduling protocol, e.g., Rate Monotonic (RM) or Earliest Deadline First (EDF). The generalization of global and partitioned scheduling algorithms is called clustered scheduling [3], [4], in which tasks are statically assigned to a subset (a cluster) of processors, and within each cluster tasks are scheduled using a global scheduling algorithm.

When the components co-execute on a shared multi-core platform they may share resources that require mutual exclusive access. To our knowledge there is no work on handling resource sharing among real-time components where each component is allocated on a cluster.

Allocation of real-time components on a multi-core architecture may have the following alternatives: (i) one processor includes only one component, (ii) one processor may contain several components, (iii) a component may be allocated on multiple processors. In a recent work [5] we have studied and developed a synchronization protocol for the first alternative which is called Multiprocessors Synchronization protocol for real-time Open Systems (MSOS). For the second alternative, the techniques developed for uniprocessors can be used, e.g., the methods presented in [6] and [7], by which the second alternative becomes similar to the first alternative. Generalization of MSOS to the third alternative, where each component is allocated on a cluster, is the objective of this paper.

The contributions of this paper are: (1) We develop a synchronization protocol for resource sharing among real-time components on a multi-core platform, where each component is allocated on multiple dedicated processors (cores). We have named the new protocol as Clustered MSOS (C-MSOS). (2) Given a real-time component, we derive an *interface-based schedulability condition* for C-MSOS. The interface abstracts the information regarding resource sharing of a component. We show that for schedulability analysis of a component there is no need for detailed information from other components, e.g., scheduling protocol or priority setting policy of other components. (3) We formulate the integration of components as a nonlinear integer programming problem for which the algorithms in this domain can be used to minimize the total number of required processors for all components.

A. Related Work

Clustered scheduling techniques have been developed for multiprocessors (multi-cores) [3], [8]. However, they assume tasks to be independent and have not studied sharing of mutually exclusive resources.

A non-exhaustive set of existing approaches for handling resource sharing on multiprocessor platforms includes; Dis-

This work was partially supported by the Swedish Foundation for Strategic Research (SSF), the Swedish Research Council (VR), and Mälardalen Real-Time Research Centre (MRTC)/Mälardalen University.

tributed Priority Ceiling Protocol (DPCP) [9], Multiprocessor PCP (MPCP) [9], Multiprocessor SRP (MSRP) [10], Flexible Multiprocessor Locking Protocol (FMLP) [11]

Recently, Brandenburg and Anderson [12] presented a new locking protocol, called O(m) Locking Protocol (OMLP) which is an *suspension-oblivious* protocol. Under a suspension-oblivious locking protocol, the suspended jobs are assumed to occupy processors and thus blocking is counted as demand. In this paper we focus on *suspension-aware* locking synchronization in which suspended jobs are not assumed to occupy processors.

Easwaran and Andersson proposed a synchronization protocol [13] under the global fixed priority scheduling protocol. In the paper, the authors have derived schedulability analysis of the Priority Inheritance Protocol (PIP) under global scheduling algorithms and proposed a new protocol called P-PCP which is a generalization of PIP. For suspension-aware resource sharing under global scheduling policies, this is the only work that provides a schedulability test, hence in our paper we use their schedulability test and assume that within a component local resources are accessed using PIP.

In all the aforementioned existing synchronization protocols on multiprocessors it is assumed that the tasks of one single real-time system are scheduled on a multiprocessor platform. C-MSOS, however, allows a component to use its own scheduling policy and it abstracts the timing requirements regarding global resources shared by the component in its interface, hence, it is not required to reveal its task attributes to other components which it shares resources with. Recently, in industry, co-existing of several separated components (systems) on a multi-core platform (called virtualization) has been considered to reduce the hardware costs [14]. C-MSOS seems to be a natural fit for synchronization under virtualization of real-time components on multi-cores where each component is allocated on multiple processors.

II. System and Platform Model

We assume that the multiprocessor platform is composed of m identical, unit-capacity processors (cores) with shared memory. We consider a set of real-time components, i.e., real-time (sub)systems, aimed to be allocated on the multiprocessor (multi-core) platform. A real-time component consists of a set of real-time tasks. A component may also include constitute components (i.e., hierarchical components), however in this paper we focus on components composed of tasks only. Each component is allocated on a dedicated subset of processors, called *cluster*. Each component has its local scheduler (which can be any multiprocessor global scheduling algorithm, e.g., G-EDF). The jobs generated by tasks of a component can migrate among the processors within its cluster, however migration of jobs among clusters is not allowed. In this paper we focus on schedulability analysis for the global fixed priority preemptive scheduling algorithm.

A component C_k consists of a task set denoted by τ_{C_k} which includes n_k sporadic tasks $\tau_i(T_i, E_i, D_i, \rho_i, \{Cs_{i,q,p}\})$ where T_i denotes the minimum inter-arrival time between two successive jobs of task τ_i with worst-case execution time E_i, relative deadline D_i and ρ_i as its unique *base priority*. A task τ_i has a higher priority than another task τ_j if $\rho_i > \rho_j$. The priority of a job of a task may temporarily be raised by a synchronization protocol which is denoted as the *effective priority*. The tasks in component C_k may share a set of mutually exclusive resources R_{C_k} which are protected using semaphores. The set of shared resources (R_{C_k}) consists of two sets of different types of resources; *local* and *global* resources. A local resource is only shared by tasks within the same component (i.e., intra-component resource sharing) while a global resource is shared by tasks from more than one component (i.e., inter-component resource sharing). The sets of local and global resources accessed by tasks in component C_k are denoted by $R^L_{C_k}$ and $R^G_{C_k}$ respectively. The set of critical sections, in which task τ_i requests resources in R_{C_k} is denoted by $\{Cs_{i,q,p}\}$, where $Cs_{i,q,p}$ is the the worst case execution time of p^{th} critical section of task τ_i in which the task uses resource $R_q \in R_{C_k}$. We define $Cs_{i,q}$ to be the worst case execution time of the longest critical section in which τ_i uses R_q. We also denote $CsT_{i,q}$ as the maximum total amount of time that τ_i uses R_q, i.e., $CsT_{i,q} = \sum Cs_{i,q,p}$. The set of tasks in component C_k sharing R_q is denoted by $\tau_{q,k}$, and $n_{i,q}$ is the total number of critical sections of task τ_i in which it accesses resource R_q. In this paper, we focus on non-nested critical sections. We also assume constrained-deadline tasks (i.e., $D_i \leq T_i$ for any τ_i). A job of task τ_i is specified by J_i and the utilization factor of τ_i is denoted by u_i where $u_i = \frac{E_i}{T_i}$.

Component C_k will be allocated on a cluster comprised of m_k processors; $m_k^{(min)} \leq m_k \leq m_k^{(max)}$ where $m_k^{(min)}$ and $m_k^{(max)}$ are the minimum and maximum number of processors required by C_k respectively. To efficiently determine the number of processors which C_k will be allocated on (i.e., m_k) in the integration phase, is one of the objectives in this paper.

III. Resource Sharing

In a component-based manner the global resource requirements of a component should be encapsulated in its *interface* (Definition 3). Furthermore, the interface should also provide information about the maximum time duration that each global resource can be held by the component. The tasks within a component should not need any detailed information about the tasks (e.g., deadlines, periods, etc.) from other components, neither do they need to be aware of the scheduling algorithms or synchronization protocols in other components.

Definition 1: *Resource Hold Time* (RHT) of a global resource R_q by task τ_i in component C_k, assuming that C_k is allocated on m_k processors, is denoted by $RHT_{q,k,i}(m_k)$ and is the maximum duration of time that the global resource R_q can be locked by τ_i. Consequently, the resource hold time of a global resource R_q by component C_k (i.e., the maximum duration of time that R_q is locked by any task in C_k) denoted by $RHT_{q,k}(m_k)$, is as follows:

$$RHT_{q,k}(m_k) = \max_{\tau_i \in \tau_{q,k}} \{RHT_{q,k,i}(m_k)\} \tag{1}$$

The concept of resource hold times for compositional real-time applications on uniprocessors was first studied in [15]. In our recent work [5] we extended this concept to multi-core (multiprocessor) platforms to calculate resource hold times of global resources under multiprocessor partitioned scheduling. In this paper we further extend RHT's to multiprocessor clustered scheduling.

Definition 2: *Maximum Resource Wait Time* (RWT) for a global resource R_q in component C_k, denoted as $RWT_{q,k}$, is the worst-case duration of time that any task τ_i within C_k can be delayed by other components (i.e., R_q is held by tasks from other components) whenever τ_i requests R_q.

Definition 3: *Component Interface:* A component C_k is abstracted and represented by an interface denoted by $I_k(Q_k(m_k)\,,\,Z_k(m_k)\,,\,m_k^{(min)}\,,\,m_k^{(max)})$.

Global resource requirements of C_k are encapsulated in the interface by $Q_k(m_k)$ which is a set of resource requirements that have to be satisfied for C_k to be schedulable on m_k ($m_k^{(min)} \leq m_k \leq m_k^{(max)}$) processors.Each requirement $r_i(m_k)$ in $Q_k(m_k)$ is represented as a linear inequality which indicates that an expression of the maximum resource wait times of one or more global resources should not exceed a value $g_i(m_k)$, e.g., $r_1(m_k) \stackrel{def}{=} 4RWT_{2,k} + 3RWT_{3,k} \leq g_1(m_k)$. Each requirement is extracted from one task requesting at least one global resource (Section VII). Thus, the number of requirements equals to the number of tasks in component C_k that may request global resources. A formal definition of the requirements is as follows:

$$Q_k(m_k) = \{r_i(m_k) \,:\, \tau_i \text{ shares global resources}\} \quad (2)$$

where

$$r_i(m_k) \stackrel{def}{=} \sum_{\substack{\forall R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} \alpha_{i,q} RWT_{q,k} \leq g_i(m_k) \quad (3)$$

where $\alpha_{i,q}$ is a constant, i.e., it only depends on internal parameters of C_k (Section VII).

A global resource requirement (in $Q_k(m_k)$) of a component C_k is extracted from the schedulability analysis of the component in isolation, i.e., to extract the requirements of a component, no information from other possible existing components (on the same multi-core platform) is required.

$Z_k(m_k)$ in the interface, $I_k(Q_k(m_k), Z_k(m_k))$, represents a set $Z_k(m_k) = \{Z_{q,k}(m_k)\}$ where $Z_{q,k}(m_k)$ is the *Maximum Component Locking Time* ($MCLT$). $Z_{q,k}(m_k)$ represents the maximum duration of time that C_k (allocated on m_k processors) can delay the execution of any task τ_x in any component C_l ($l \neq k$) whenever τ_x requests R_q, i.e., any time any task in C_l requests R_q its execution can be delayed by C_k for at most $Z_{q,k}(m_k)$ time units.

IV. Locking Protocol for Real-Time Components under Clustered Scheduling

In this section we generalize our recently proposed locking protocol, MSOS (Multiprocessors Synchronization protocol for real-time Open Systems) [5], to real-time components (applications) that are allocated on multiple processors. Under MSOS each component is assumed to be allocated on one dedicated processor. In this paper we generalize MSOS such that a component can be allocated on one cluster. Thus the tasks within each component have to be scheduled using a global scheduling policy and local resources are to be handled using a locking protocol under global scheduling policies. We call the generalized protocol C-MSOS (Clustered MSOS).

We assume that the Priority Inheritance Protocol (PIP) for multiprocessors is used for sharing local resources among tasks of a component. We extend the schedulability analysis presented in [13] such that it is compatible with C-MSOS. First we review the characteristics of PIP for multiprocessors as described in [13].

A. PIP on Multiprocessors

Assume that a task set is scheduled on a multiprocessor composed of m processors, and that shared resources are handled by PIP. Whenever a job J_i is blocked on a resource which is locked by another job J_j with a lower base priority than J_i, the effective priority of J_j is raised to the priority of J_i if the effective priority of J_j is not already higher than the priority of J_i. In this case, J_i is said to be *directly blocked* [13] by J_j if J_i is among the m highest priority jobs.

Under PIP, besides direct blocking, a job J_i can also incur interference from other lower priority jobs whose effective priorities have been raised above J_i's priority. Furthermore, J_i may incur extra interference from higher priority jobs when they have locked a resource that J_i has requested and J_i is among the m highest priority jobs.

B. General Description of C-MSOS

Under C-MSOS, sharing local resources is handled by multiprocessor PIP. Each global resource is associated with a *global queue* in which components requesting the resource are enqueued. Since prioritizing the components may not be possible, the global queues can be implemented in either FIFO or Round-Robin manner. In [5] we only studied FIFO-based global queues. In this paper we study both types.

Within a component the jobs requesting a global resource are enqueued in a *local queue*. The blocking time on global resources should only depend on the duration of *global critical sections* (gcs) in which jobs access global resources. This bounds blocking times on global resources as a function of (length and number of) global critical sections only. Thus the priority of jobs accessing global resources should be boosted to be higher than any base priority within the component. The *boosted priority* of any job of task τ_i requesting any global resource equals to $\rho^{max}(C_k) + 1$, where $\rho^{max}(C_k) = \max\{\rho_i | \tau_i \in C_k\}$. Boosting the priority of a job when it executes within a gcs ensures that it can only be preempted by jobs within gcs's.

C. C-MSOS Rules

The C-MSOS request rules are as follows:
Rule 1: Access to the local resources is handled by PIP (Section IV-A).
Rule 2: When a job J_i within a component C_k requests a global resource R_q the priority of J_i is increased immediately to its boosted priority (i.e., $\rho^{max}(C_k) + 1$).
Rule 3: If global resource R_q is free, access to R_q is granted to J_i. If R_q is locked (by a local job or another component); (i) if the global queues are FIFO-based a placeholder for C_k is added to the global queue of R_q, and
(ii) if the global queues are Round-Robin-based (e.g., the global queues can be implemented as a ring queue in which each component has one placeholder) C_k's placeholder is set to an appropriate value. For Round-Robin global queues there will be at most one placeholder per each component in any global queue while a FIFO global queue may contain more than one placeholder for any component sharing the corresponding resource. Locally (for both types of global queues) J_i is located in the local queue of R_q and suspends.
Rule 4: When a global resource R_q becomes available to component C_k the eligible job (e.g., the one at the top of the local queue if the local queue is a FIFO queue) is granted accesses to R_q.
Rule 5: When J_i is granted to access R_q all processors of the component may be busy by other jobs executing global resources other than R_q. The jobs that are granted access to global resources are enqueued in a FIFO queue denoted by $allResourcesQ$. Obviously jobs in $allResourcesQ$ are granted access to different global resources and it does not contain more than one job per each global resource. At the time J_i is granted access to R_q, if all processors are occupied by other jobs accessing other global resources, J_i is added to $allResourcesQ$. As soon as an executing job releases a global resource (it enters a non-critical section) it will be preempted by the job (say J_x) at the top of $allResourcesQ$ (if any), and J_x will hold the global resource it has been granted access to and it will be removed from $allResourcesQ$.
Rule 6: When J_i releases R_q;
(i) in the case of using FIFO global queues, the placeholder of C_k from the top of the global FIFO queue of R_q will be removed and R_q becomes available to the component whose placeholder is now at the top of R_q's global queue,
(ii) in the case of using Round-Robin global queues, J_i is removed from the local queue and R_q becomes available to the next component whose placeholder is set. If the local queue is empty the placeholder of C_k is reset (e.g., the placeholder is set to 0).

V. Schedulability Analysis

In this section we extend the response time analysis for multiprocessor PIP in [13] to be applicable to C-MSOS.

A. Schedulability Analysis of PIP

Easwaran and Andersson [13] have shown that under multiprocessor PIP the response time of any task τ_i denoted by RT_i can be calculated as follows:

$$\begin{aligned} RT_i = \quad & E_i + DB_i + Ihp_i^{(dsr)} + Ihp_i^{(osr)} \\ & + Ihp_i^{(nsr)} + Ilp_i \end{aligned} \tag{4}$$

where

- DB_i upper bounds the direct blocking (on local resources) that τ_i incurs,
- $Ihp_i^{(dsr)}$ is an upper bound for the amount of time that tasks with a higher base priority than τ_i lock (local) resources shared by τ_i (*direct shared resources*),
- $Ihp_i^{(osr)}$ is an upper bound for the amount of time that tasks with a higher base priority than τ_i may lock (local) resources not shared by τ_i (*other shared resources*),
- $Ihp_i^{(nsr)}$ is an upper bound for the amount of time that tasks with a higher base priority than τ_i execute in their non-critical sections, i.e., they do not hold any resource (*no shared resource*),
- Ilp_i upper bounds the amount of time that tasks with a lower base priority than τ_i execute with a higher effective priority than τ_i.

All the aforementioned factors that contribute to response time of τ_i, except $Ihp_i^{(nsr)}$, are delays inherent in local resources. Thus, for the sake of simplicity we rewrite Equation 4 as follows:

$$RT_i = E_i + Ihp_i^{(nsr)} + I^{local}(\tau_i) \tag{5}$$

where $I^{local}(\tau_i) = DB_i + Ihp_i^{(dsr)} + Ihp_i^{(osr)} + Ilp_i$.

To upper bound the worst-case interference from any task τ_j to task τ_i in the interval RT_i Easwaran and Andersson have presented a worst case execution pattern [13]. In this pattern, during the interval RT_i, the carry-in job of τ_j executes as late as possible and all following jobs execute as early as possible. This pattern was fist proposed by Bertogna and Cirinei [16] and later extended by Easwaran and Andersson to maximize the total interference from a *certain portion* x (e.g., critical sections) of execution time of any job τ_j to τ_i in RT_i. In the extended pattern, x time units of execution time of the carry-in job appears as late as possible and the x time units of execution time of all the following jobs (of τ_j in interval RT_i) appear as early as possible (Figure 1). In this worst-case execution pattern Easwaran and Andersson [13] have shown that in any interval t the total execution of x units of jobs of any task τ_j is maximized as follows:

$$W_j(t,x) = x\,N_j(t,x) + \min\{x, t - x + D_j - T_j N_j(t,x)\} \tag{6}$$

where $N_j(t,x) = \left\lfloor \frac{t-x+D_j}{T_j} \right\rfloor$.

Based on this worst-case execution pattern Easwaran and Andersson have calculated DB_i, $Ihp_i^{(dsr)}$, $Ihp_i^{(osr)}$, $Ihp_i^{(nsr)}$ and Ilp_i(for details about the calculations please read [13]).

1) Improved Response Times for m_k Highest Priority Tasks: Easwaran and Andersson in [13] have further improved the computation of the response times for m_k highest priority tasks. The improved response times of m_k highest priorities

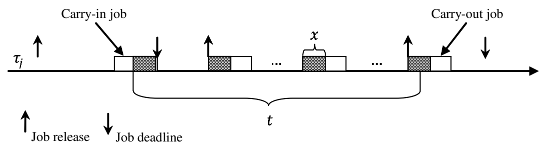

Fig. 1. Worst-case execution pattern regarding giving importance to a certain portion of execution time.

is calculated as follows (for details about the rationale behind the improved response times please read [13]):

$$RT_i = \begin{cases} E_i + DB_i + Ihp_i^{(dsr)} & |\tau_H(\tau_i)| < m_k \\ E_i + DB_i + Ihp_i^{(dsr)} & \\ + Ihp_i^{(osr)} + Ihp_i^{(nsr)} + Ilp_i & \text{Otherwise} \end{cases} \tag{7}$$

where $|\tau_H(\tau_i)|$ is the number of tasks with priority higher than that of τ_i.

B. Schedulability Analysis of C-MSOS

1) Computing Resource Hold Times: In this section we determine the calculation of resource hold times of tasks and components. We assume that component C_k is allocated on m_k processors. Any job of task τ_i (in C_k) which is granted access to a global resource R_q can only be delayed by other jobs accessing global resources other than R_q because the boosted priority of a job which is granted access to a global resource is higher than any base priority of other jobs within its consisting component. At the time R_q becomes available to any job of τ_i, in the worst case all other jobs (which share other global resources) have been granted access to their requested global resources before τ_i. However, at any time if the number of those jobs that are ahead of τ_i is less than m_k, they do not interfere with τ_i's job. Thus an upper bound of the delay that τ_i incurs by other tasks when it is granted access to R_q is denoted by $H_{i,q}$ and can be calculated as follows:

$$H_{i,q}(m_k) = \frac{\sum_{\substack{\tau_j \neq \tau_i \\ \wedge\, \tau_j \in \tau_{l,k}}} \left(\max_{R_l \in R^G_{C_k},\, l \neq q} \{Cs_{j,l}\} \right)}{m_k} \tag{8}$$

τ_i itself will hold R_q for at most $Cs_{i,q}$ time units. Thus the maximum duration of time that τ_i can lock R_q can be calculated as follows:

$$RHT_{q,k,i}(m_k) = Cs_{i,q} + H_{i,q}(m_k) \tag{9}$$

As shown in Equation 1 the resource hold time of a resource in a component is the longest resource hold time among all tasks sharing the resource. Equation 8 shows that $H_{i,q}(m_k)$ is the same for any task τ_i sharing global resource R_q. Thus, the resource hold time of a global resource will be the resource hold time of a task that has the longest gcs (in which it accesses R_q) among all tasks sharing the resource, i.e., to compute RHT of R_q it is enough to calculate the RHT of the task possessing the longest gcs in which it accesses R_q.

Looking at Equations 8 and 9, it is clear that all parameters except m_k are constants. Thus $RHT_{q,k,i}(m_k)$ and consequently $RHT_{q,k}(m_k)$ is a function of m_k.

2) Computing Maximum Resource Wait Times: Each time a component C_k requests a global resource R_q it can be blocked by each component C_l up to $Z_{q,l}(m_l)$ time unites. Thus the worst-case waiting time ($RWT_{q,k}$) for C_k to wait until R_q becomes available is bounded as a summation of all $MCLT$'s of other components on R_q:

$$RWT_{q,k} = \sum_{l \neq k} Z_{q,l}(m_l) \tag{10}$$

Calculation of $MCLT$'s for components depends on the type of global queues. In the case of using FIFO global queues, whenever a component C_l requests a global resource R_q the worst case happens when all tasks from component C_k sharing R_q have issued requests before C_l (i.e., are already in the global FIFO). On the other hand each task in C_k may hold R_q up to $RHT_{q,k,i}(m_k)$ time units, hence we can set $Z_{q,k}(m_k)$ as follows:

$$Z_{q,k}(m_k) = \sum_{\substack{\forall \tau_i,\ \tau_i \in \tau_{q,k} \\ \wedge\ R_q \in R^G_{C_k}}} RHT_{q,k,i}(m_k) \tag{11}$$

If the global queues are of type Round-Robin, each component can have at most one placeholder in each global queue of global resources that it shares, e.g., whenever a global resource R_q is released by a job in component C_k, the resource R_q should become available to the next component even if there are jobs waiting for R_q in the local queue of R_q. Thus when C_l is waiting for resource R_q, it may wait for component C_k for at most $\max\{RHT_{q,k,i}\}$ time units:

$$Z_{q,k}(m_k) = RHT_{q,k}(m_k) \tag{12}$$

3) Response Times under C-MSOS: Under C-MSOS, the response time of any task τ_i in component C_k, besides the factors mentioned in Section V-A, also depends on interference with other jobs regarding global resource sharing, i.e., interference with other jobs within the same component as well as the other components. The execution (of any job) of τ_i may further be delayed by the following factors, denoted as *global factors*:

- The tasks with base priority lower than τ_i that may lock global resources shared by τ_i. DB_i^G (*direct global blocking*) denotes an upper bound on the amount of time (i.e., workload) that these tasks lock global resources shared by τ_i in interval RT_i.
- The tasks with higher (base) priority than τ_i that access global resources shared by τ_i. An upper bound of the amount of time that these tasks may delay τ_i during interval RT_i is denoted by $Ihp_i^{(dsgr)}$ (*direct shared global resources*).
- The tasks with priority lower than τ_i that may access any global resource. The tasks holding global resources

may delay the execution of any task since their effective priority is boosted to be higher than any task's priority. $Ilp_i^{(gr)}$ (*global resources*) denotes an upper bound on the amount of time that jobs of these tasks execute with boosted priority in interval RT_i.
- The components other than C_k whose tasks may lock global resources shared by τ_i. RB_i (*remote blocking*) is an upper bound on the amount of time that (tasks in) those components lock global resources shared by τ_i during interval RT_i.

Considering these interferences under C-MSOS, the response time of task τ_i is calculated as follows:

$$\begin{aligned} RT_i = \; & E_i + Ihp_i^{(nsr)} + I^{local}(\tau_i) \\ & + DB_i^G + Ihp_i^{(dsgr)} + Ilp_i^{(gr)} + RB_i \end{aligned} \tag{13}$$

Computing the global factors: We now compute the global factors that may delay the execution (of any job) of task τ_i in component C_k. We use the dispatch pattern in Figure 1 and the definition of workload in Equation 6 to calculate these factors.

(i) Computing DB_i^G:

- *for FIFO-based local queues:* whenever a job J_i of τ_i in component C_k requests a global resource R_q, in the worst case all the lower priority jobs in component C_k sharing R_q may have requested it before J_i and are located in the local FIFO queue before J_i. Thus DB_i^G can be calculated as follows:

$$DB_i^G = \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} \Big(n_{i,q} \sum_{\substack{\rho_j < \rho_i \\ \wedge\ \tau_j \in \tau_{q,k}}} Cs_{j,q} \Big) \tag{14}$$

- *for priority-based local queues:* whenever a job J_i of τ_i in component C_k requests a global resource R_q, it may happen that a lower priority job in component C_k has locked R_q. However, J_i will not be delayed by lower priority jobs requesting R_q that request R_q after J_i does. Thus DB_i^G can be calculated as follows:

$$DB_i^G = \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} n_{i,q} \max_{\substack{\rho_j < \rho_i \\ \wedge\ \tau_j \in \tau_{q,k}}} \{Cs_{j,q}\} \tag{15}$$

(ii) Computing $Ihp_i^{(dsgr)}$:

- *for FIFO-based local queues:* in addition to lower priority jobs, in the worst case all higher priority jobs will be located before J_i in the local FIFO queue whenever J_i requests R_q. Thus $Ihp_i^{(dsgr)}$ is calculated as follows:

$$Ihp_i^{(dsgr)} = \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} \Big(n_{i,q} \sum_{\substack{\rho_j > \rho_i \\ \wedge\ \tau_j \in \tau_{q,k}}} Cs_{j,q} \Big) \tag{16}$$

- *for priority-based local queues:* whenever a job J_i of τ_i in component C_k requests a global resource R_q, it may happen that all the higher priority jobs sharing R_q, also request R_q. When R_q is locked by these higher priority jobs, more of theses higher priority jobs may arrive and request R_q. Thus $Ihp_i^{(dsgr)}$ is calculated as follows:

$$Ihp_i^{(dsgr)} = \sum_{\rho_j > \rho_i} W_j \Big(RT_i, \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \{\tau_i, \tau_j\} \subset \tau_{q,k}}} CsT_{j,q} \Big) \tag{17}$$

(iii) Computing $Ilp_i^{(gr)}$: upper bounds the amount of time that the jobs of tasks with lower priority than τ_i delay the execution of any job of τ_i, when they execute with their boosted priority (i.e., when holding global resources). The jobs contributing to $Ilp_i^{(gr)}$ also include all the jobs of lower priority tasks that share global resources with τ_i. For example suppose that a task τ_x with lower priority than that of τ_i shares a global resource R_q with τ_i. The longest gcs of τ_x (i.e., $Cs_{x,q}$) in which it shares R_q with τ_i contributes to DB_i^G (i.e., whenever τ_i requests R_q, τ_x may have requested it before). On the other hand, all gcs's of τ_x may also repeatedly interfere with τ_i when the jobs of τ_i execute in their non-critical sections. However, if there are less than m_k jobs executing at the same time, the lower priority jobs executing within their gcs's do not interfere with the executing job of τ_i. Thus we can calculate $Ilp_i^{(gr)}$ as follows:

$$Ilp_i^{(gr)} = \frac{\sum_{\rho_j < \rho_i} W_j \Big(RT_i, \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_j \in \tau_{q,k}}} CsT_{j,q} \Big)}{m_k} \tag{18}$$

Please notice that calculation of DB_i^G, $Ihp_i^{(dsgr)}$ and $Ilp_i^{(gr)}$ does not depend on the type of global queues (i.e., FIFO or Round-Robin), hence those calculations are valid for both types. However, the execution delay introduced to tasks from other components with which they share global resources is calculated differently depending on the type of global queues as explained in the following:

(iv) Computing RB_i for FIFO global queues:

- *for FIFO-based local queues:* whenever a job of task τ_i requests a global resource R_q, in the worst case all local tasks (in the same component as τ_i) as well as all tasks from other components have requested R_q before τ_i. However, the execution delays introduced by the local tasks regarding shared global resources are considered in DB_i^G and $Ihp_i^{(dsgr)}$. On the other hand, τ_i will wait for any component C_l for at most $Z_{q,l}$ time units (whenever it requests R_q) because for FIFO global queues $Z_{q,l}$ is the summation of resource hold times of all tasks from component C_l that share R_q (Equation 11). Consequently τ_i will wait up to $RWT_{q,k} = \sum_{l \neq k} Z_{q,l}$ time units considering all other components sharing R_q. Thus for FIFO global queues RB_i is calculated as follows:

$$RB_i = \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} n_{i,q} RWT_{q,k} \tag{19}$$

- *for priority-based local queues:* In [5] for the case of using priority-based local queues for accessing global resources, we have shown that whenever a job J_i requests a global resource R_q each request to R_q from a higher priority job J_j will introduce an extra $RWT_{q,k}$ to J_i. The rationale is similar here as well, i.e., each gcs of J_j in which it requests R_q, may delay J_i for another $RWT_{q,k}$. Similar to the calculation of workload of a task during an interval by Equation 6, the maximum number of gcs's of a task τ_j during any interval t can be calculated as following:

$$Ngcs_j(t,E_i) = N_j(t,E_i) + \left\lceil \frac{\min\{E_i, t-E_i+D_j-T_jN_j(t,E_i)\}}{E_i} \right\rceil \quad (20)$$

where the first term $N_j(t,E_i)$ is the number of jobs that totally locate in t (their arrival and deadline locates within t) plus the carry-in job. The second term equals to 1 if there is a carry-out job within t, and it equals 0 otherwise. Besides each $RWT_{q,k}$ introduced because of requests of higher priority jobs, the request of J_i to R_q itself may also wait for R_q up to $RWT_{q,k}$ time units. Thus, for the case that both the global queues and local queues for accessing global resources are FIFO-based, RB_i can be calculated as follows:

$$RB_i = \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} \Big(n_{i,q} + \sum_{\substack{\rho_j > \rho_i \\ \wedge\ \tau_j \in \tau_{q,k}}} n_{j,q} Ngcs_j(RT_i, E_i) \Big) RWT_{q,k} \quad (21)$$

(v)Computing RB_i for Round-Robin global queues:

- *for FIFO-based local queues:* every time a job of task τ_i from component C_k requests global resource R_q, the worst case happens when all local tasks have requested R_q before τ_i and globally in the worst case per every request to R_q, C_k has to wait for all other components. Thus, in the worst case every request before τ_i's as well as τ_i's own request to R_q have to wait $RWT_{q,k} = \sum_{l \neq k} Z_{q,l}$ time units until R_q becomes available. The maximum number of requests in the local queue of global resource R_q is the number of tasks sharing the resource which is denoted by $|\tau_{q,k}|$. Considering that the durations of time that local tasks hold R_q are counted in DB^G_i and $Ihp^{(dsgr)}_i$, we can calculate RB_i for Round-Robin global queues as follows:

$$RB_i = \sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} n_{i,q} |\tau_{q,k}| RWT_{q,k} \quad (22)$$

- *for priority-based local queues:* This case is similar to the case of using priority-based local queues with FIFO-based global queues. This means that a request of any job J_i to a global resource R_q may incur $RWT_{q,k}$ time units per each request to R_q from higher priority jobs sharing R_q. Furthermore, each request of J_i, itself may wait at most $RWT_{q,k}$ time unit for R_q to be released by other components accessing R_q. Thus, RB_i for the case of using Round-Robin-based global queues and priority-based local queues to access global resources can also be calculated by Equation 21.

Looking at Equations 19 and 22 it may seem that the value of remote blocking (RB_i) under Round-Robin global queues is always greater than that under FIFO global queues as each maximum resource wait time, for Round-Robin queues, is multiplied by the number of tasks sharing the global resource (e.g., $|\tau_{q,k}|$). This is not true, because maximum resource wait times are calculated differently depending on the type of global queues; comparing Equations 11 and 12 shows that under Round-Robin global queues maximum component locking times and consequently maximum resource wait times (Equation 10) are smaller than that under FIFO global queues. Thus in different situations (e.g., the number of tasks sharing a global resource) remote blocking under either type of global queues can be larger than that under the other one.

C. Improved Response Times under C-MSOS

Easwaran and Andersson in [13] have shown that under PIP, the response time of any task τ_i among the m_k highest priority tasks only depends on E_i, DB_i and $Ihp^{(dsr)}_i$ which are the worst-case execution time of τ_i and the factors with regarded to the local resources that τ_i shares. These factors represent sequential executions and they do not depend on the number of processors available to C_k. However, as shown in [13] the other factors (i.e., $Ihp^{(nsr)}_i$, $Ihp^{(osr)}_i$ and Ilp_i) are affected by the number of processors and they do not affect response time of τ_i if τ_i is among the m_k highest priority tasks. In this section we present how the response times for some of the m_k highest priority tasks under C-MSOS can be improved.

Under C-MSOS, besides the mentioned sequential factors, the factors DB^G_i, $Ihp^{(dsgr)}_i$ and RB_i regarding the global resources accessed by τ_i are also sequential. This means that when τ_i is waiting for a locked global resource the waiting cannot be reduced even if there is a free processor in C_k. Thus the factors DB^G_i, $Ihp^{(dsgr)}_i$ and RB_i contribute to the response time of τ_i even if τ_i is among the m_k highest priority tasks in C_k (i.e., $|\tau_H(\tau_i)| < m_k$). However, a job J_i generated by τ_i can execute in parallel with other jobs accessing global resources that are not requested by J_i. Hence, $Ilp^{(gr)}_i$ will not affect RT_i if there are enough processors.

Thus, if the number of tasks with higher priority than that of τ_i plus the number of tasks with lower priority than that of τ_i and that share any global resources is less than m_k, the execution of τ_i will never be delayed except the times it is waiting for a locked resource. Thus we can rewrite the response time calculation in Equation 13 for task τ_i as follows, where $|\tau^G_L(\tau_i)|$ denotes the number of tasks with priority lower than that of τ_i that share any global resources:

If $|\tau_H(\tau_i)| + |\tau_L^G(\tau_i)| < m_k$:

$$RT_i = E_i + DB_i + Ihp_i^{(dsr)} + DB_i^G + Ihp_i^{(dsgr)} + RB_i \quad (23)$$

otherwise

$$\begin{aligned} RT_i = {} & E_i + DB_i + Ihp_i^{(dsr)} + DB_i^G + Ihp_i^{(dsgr)} + RB_i \\ & + Ihp_i^{(osr)} + Ihp_i^{(nsr)} + Ilp_i + Ilp_i^{(gr)} \end{aligned} \quad (24)$$

VI. Extracting Interfaces

A component C_k is abstracted by its interface I_k, which consists of four elements; $Q_k(m_k)$, $Z_k(m_k)$, $m_k^{(min)}$ and $m_k^{(max)}$ (Definition 3). In Section V-B2 we have shown how to calculate the elements of $Z_k(m_k)$ (e.g., $Z_{q,k}(m_k)$ for resource R_q) for FIFO and Round-Robin global queues (Equations 11 and 12 respectively). In this section we determine how to extract the requirements in $Q_k(m_k)$ as well as $m_k^{(min)}$ and $m_k^{(max)}$ by means of schedulability test of C-MSOS.

A. Deriving Requirements

As shown in Equation 3, a requirement in $Q_k(m_k)$ specifies that a linear expression whose variables are the maximum resource wait times of one or more global resources should not exceed a value which is a function of m_k, e.g., $g_i(m_k)$. Each requirement is derived from the schedulability analysis of one task that shares any global resources and each task sharing global resources produces one requirement.

To guarantee schedulability of a component C_k on m_k processors, for any task τ_i in C_k, condition $RT_i \le D_i$ has to be satisfied. Looking at the calculation of response times under C-MSOS (Section V-B3), the response time of tasks that do not share any global resources is only dependent on the local factors, i.e., for task τ_i the only factor in RT_i that needs information from other components (other than τ_i's consisting component) is the remote blocking factor RB_i. If τ_i does not share any global resources then $RB_i = 0$ because it will not be blocked on any global resource. Intuitively the response time of such task can be calculated without any requirement on external factors. On the other hand, if τ_i shares global resources it may incur remote blocking. However, the amount of remote blocking that τ_i can tolerate is limited and it should not exceed a value that makes τ_i to miss its deadline.

The maximum acceptable response time of τ_i denoted by $RT_i^{(max)}$, is when it equals its deadline, i.e., $RT_i^{(max)} = D_i$. During interval $RT_i^{(max)}$ (or D_i) the delay introduced by local factors and global factors except remote blocking RB_i is constant which means that they can be calculated without any requirement on external factors from other components. The remaining slack (if any) can be taken as the maximum tolerable remote blocking. Thus the maximum remote blocking $RB_i^{(max)}$ that τ_i can tolerate is calculated as follows:

$$RB_i^{(max)} = RT_i^{(max)} - internal_i(m_k)$$

where

$$\begin{aligned} internal_i(m_k) = {} & E_i + DB_i + Ihp_i^{(dsr)} + DB_i^G + Ihp_i^{(dsgr)} \\ & + Ihp_i^{(osr)} + Ihp_i^{(nsr)} + Ilp_i + Ilp_i^{(gr)} \end{aligned}$$

by replacing $RT_i^{(max)}$ with D_i:

$$RB_i^{(max)} = D_i - internal_i(m_k) \quad (25)$$

The terms in $internal_i(m_k)$ can intuitively be calculated using their corresponding equations in Section V by replacing RT_i with D_i where it is applicable.

Looking at the calculation of RB_i in Equations 19 and 22 for FIFO and Round-Robin global queues respectively we can rewrite the calculation of RB_i as follows:

$$RB_i = \sum_{\substack{R_q \in R_{C_k}^G \\ \wedge\ \tau_i \in \tau_{q,k}}} \alpha_{i,q} RWT_{q,k} \quad (26)$$

where $\alpha_{i,q} = n_{i,q}$ for FIFO and $\alpha_{i,q} = n_{i,q}|\tau_{q,k}|$ for Round-Robin queues respectively. In both cases, $\alpha_{i,q}$ is a constant, i.e., it depends only on the local factors.

Considering $RB_i \le RB_i^{(max)}$ and by combining Equations 25 and 26 we can derive the following requirement:

$$\sum_{\substack{R_q \in R_{C_k}^G \\ \wedge\ \tau_i \in \tau_{q,k}}} \alpha_{i,q} RWT_{q,k} \le D_i - internal_i(m_k) \quad (27)$$

The requirement derived in Equation 27 adheres the definition of a requirement in Equation 3 ($g_i(m_k) = D_i - internal_i(m_k)$).

The discussion in Section V-C for improvement of response times can also be applied here to improve (reduce) $internal_i(m_k)$ and consequently improve (relax) the requirement in Equation 27 for some of the tasks among the m_k highest priority tasks sharing global resources:
If $|\tau_H(\tau_i)| + |\tau_L^G(\tau_i)| < m_k$:

$$internal_i(m_k) = E_i + DB_i + Ihp_i^{(dsr)} + DB_i^G + Ihp_i^{(dsgr)} \quad (28)$$

otherwise

$$\begin{aligned} internal_i(m_k) = {} & E_i + DB_i + Ihp_i^{(dsr)} + DB_i^G + Ihp_i^{(dsgr)} \\ & + Ihp_i^{(osr)} + Ihp_i^{(nsr)} + Ilp_i + Ilp_i^{(gr)} \end{aligned} \quad (29)$$

As it can be seen in Equation 28, $internal_i(m_k)$ cannot further be reduced even if m_k is increased since none of its terms are dependant on the number of processors that C_k is allocated on.

B. Determine Minimum and Maximum Required Processors

In this section we derive the calculations to determine $m_k^{(min)}$ and $m_k^{(max)}$ for component C_k in its interface.

$m_k^{(min)}$ is the minimum number of processors required by C_k such that it is schedulable. Obviously $\lceil U_k \rceil \le m_k^{(min)}$ where $\lceil U_k \rceil = \sum_{\tau \in \tau_{C_k}} u_i$.

Theorem 1. *Under C-MSOS, the minimum number of required processors for component C_k to be schedulable, can be* *achieved by setting $RB_i = 0$ for any task τ_i sharing global resources, and is calculated as follows:*

$$m_k^{(min)} = \min_{\substack{m_x \geq \lceil U_k \rceil \\ \wedge\, C_k \text{ is schedulable on } m_x}} \{m_x\} \tag{30}$$

Proof: If task τ_i does not share any global resources, its response time depends only on internal factors (i.e., no information from other components is needed). We assume that C_k is allocated on m_k processors. Considering the calculations for the factors regarding local resources [13], among the factors that RT_i depends on E_i, DB_i and $Ihp_i^{(dsr)}$ do not depend on m_k while $Ihp_i^{(osr)}$, $Ihp_i^{(nsr)}$ and Ilp_i monotonically decrease when m_k increases, and consequently RT_i monotonically decreases by increasing m_k. Thus, when increasing m_k, suppose that m_x is the first number of processors for which RT_i is decreased enough such that $RT_i \leq D_i$. In this case τ_i will not need further increase of the number of processors since it is already schedulable by m_x processors.

If τ_i shares global resources, besides the aforementioned factors, its response time also depends on DB_i^G, $Ihp_i^{(dsgr)}$, $Ilp_i^{(gr)}$ as well as remote blocking RB_i. Looking at the calculations for these factors in Section V-B, DB_i^G and $Ihp_i^{(dsgr)}$ do not depend on m_k while $Ilp_i^{(gr)}$ monotonically decrease with increasing m_k. RB_i depends on information from components other than C_k. Setting $RB_i = 0$ means that no other component, whose tasks may access global resources shared by τ_i, co-execute with C_k. Thus, τ_i does not need to tolerate any remote blocking. Let us denote $RT_i^0(m_l)$ as the response time of τ_i where C_k is allocated on m_l processors and $RB_i = 0$, and denote $RT_i^{>0}(m_l)$ when $RB_i > 0$. We assume that m_x is the smallest number of processors for which $RT_i^0(m_x) \leq D_i$. Suppose that there exist a m_y such that if C_k is allocated on m_y processors the following statement is true: $(m_y < m_x) \wedge\ (RB_i > 0) \wedge\ (RT_i^{>0}(m_y) \leq D_i)$. From one side the response time of τ_i will increase if the remote blocking factor RB_i is increased and from the other side the response time of τ_i also monotonically increase by reducing the number of processors, hence $RT_i^0(m_y) \leq RT_i^{>0}(m_y)$. Considering that we have supposed $RT_i^{>0}(m_y) \leq D_i$ it turns out that $RT_i^0(m_y) \leq D_i$ which is in contradiction with the assumption that m_x is the minimum number of processors on which $RT_i^0(m_x) \leq D_i$.

Thus setting $RB_i = 0$ for any task τ_i sharing global resources, $m_k^{(min)}$ is the smallest m_x ($m_x \geq \lceil U_k \rceil$) number of processors on which $RT_l \leq D_l$ for any task τ_l in C_k. ∎

$m_k^{(max)}$ is the maximum number of processors required for C_k to be schedulable, i.e., further increasing the number of processors for C_k does not improve the schedulability of any component. In a component C_k the tasks that do not share any global resources do not benefit (from the schedulability point of view) from increasing the number of processors from $m_k^{(min)}$ since these tasks are already schedulable on $m_k^{(min)}$ processors. However, (Equation 27) for any task τ_i sharing global resources, the requirement extracted from τ_i will be relaxed by increasing m_k, i.e., τ_i can tolerate more remote blocking (from other components) which benefits other components sharing global resources with τ_i. Thus $m_k^{(max)}$ is the maximum number of processors where at least one requirement in $Q_k(m_k)$ is relaxed, i.e., allocating C_k on m_h where $m_h > m_k^{(max)}$ does not further relax any requirement hence no component will benefit from C_k being allocated on m_h compared to the case where C_k is allocated on $m_k^{(max)}$.

Theorem 2. *Under C-MSOS, $m_k^{(max)} = |\tau_H(\tau_{min})| + 1$, where τ_{min} is the task with minimum priority among all tasks sharing any global resources.*

Proof: In Section VI-A we have showed that for a task τ_i sharing any global resources, if $|\tau_H(\tau_i)| + |\tau_L^G(\tau_i)| < m_k$, $internal_i(m_k)$ cannot further be decreased by increasing m_k (i.e., the requirement extracted from τ_i cannot further be relaxed). When increasing m_k, as soon as all tasks sharing global resources are among the m_k highest priority tasks, condition $|\tau_H(\tau_i)| + |\tau_L^G(\tau_i)| < m_k$ will hold for any task sharing global resources. This is because if a task τ_i (sharing any global resources) is among the m_k highest priority tasks any task in $\tau_H(\tau_i)$ will also be among them. Furthermore since all tasks sharing global resources are among the m_k highest priority tasks any task in $\tau_L^G(\tau_i)$ as well as τ_i itself are also among them, thus $|\tau_H(\tau_i)| + |\tau_L^G(\tau_i)| < m_k$ holds for all these tasks. Hence, by definition $m_k = m_k^{(max)}$.

On the other hand the last task sharing global resources that ends up in the m_k highest priority tasks will be τ_{min}. As soon as τ_{min} ends up in m_k priority tasks, these m_k tasks will only consist of all tasks in $\tau_H(\tau_{min})$ and τ_{min} itself. Thus $m_k = |\tau_H(\tau_{min})| + 1$. ∎

VII. Minimizing the Number of Required Processors for all Components

In this section we will show that using the information in the interfaces of components the integration of all the real-time components on a multiprocessor platform can be formulated as a Nonlinear Integer Programming (NIP) problem [17]. By formulating the integration phase as a NIP problem, by means of the techniques in this domain [17] we can minimize the number of required processors on which all components will be schedulable.

A typical model of a NIP problem is represented as follows: For n number of integer variables $x_1, \cdots, x_n$, there is an objective function $f : R^n \longrightarrow R$ to be minimized (or maximized):

$$Minimize\ \ f(x_1, \cdots, x_n) \tag{31}$$

With a set of (nonlinear) inequality constraints g and a set of (nonlinear) quality constraints h formed as follows:

$$\begin{array}{l} g_i(x_1, \cdots, x_n) \leq b_i, \ \ i = 1, \cdots, i = p \\ h_j(x_1, \cdots, x_n) = c_j, \ \ j = 1, \cdots, i = q \end{array} \tag{32}$$

If the objective function f or some of the constraints g_i are nonlinear, the problem is a NIP problem. An optimal solution $(\bar{x}_1, \cdots, \bar{x}_n)$ is a solution for which all constraints hold and the output of the objective function is minimized.

Our goal is to minimize the number of total required processors by all components in the integration phase. Thus, assuming that there are n components to be integrated on a multiprocessor platform, the objective function will be formed as follows:

$$Minimize\ \ f(m_1, \cdots, m_n) = m_1 + \cdots + m_n \tag{33}$$

where m_i is the number of processors that C_i will eventually be allocated on.

We rewrite the model of a requirement r_i (Equation 27) in the requirement set $Q_k(m_k)$ of a component C_k:

$$\sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} \alpha_{i,q} RWT_{q,k} \leq D_i - internal_i(m_k)$$

It can be shown that by replacing the terms of $internal_i(m_k)$ with their calculations from corresponding Equations in [13] and Equations 14, 16, and 18 in this paper, it can be simplified as follows:

$$internal_i(m_k) = \beta_i + \frac{\delta_i}{m_k} \tag{34}$$

where β_i and δ_i are constant numbers, i.e., they depend only on the internal parameters of C_k.

Thus we can rewrite requirement r_i as follows:

$$\sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} \alpha_{i,q} RWT_{q,k} \leq d_i - \frac{\delta_i}{m_k} \tag{35}$$

where $d_i = D_i - \beta_i$.

Shown in Equation 10, $RWT_{q,k}$ is the summation of $Z_{q,s}(m_s)$'s ($s \neq k$) and $Z_{q,s}(m_s)$'s in turn as shown in Equation 11 (we consider FIFO global queues without loss of generality) depend on RHT's. Furthermore, similar to the simplification of $internal_i(m_k)$ in Equation 34, the calculation of $RHT_{q,s,i}(m_s)$ can be simplified as follows:

$$RHT_{q,s,i}(m_s) = \sigma_i + \frac{\gamma_i}{m_s} \tag{36}$$

where $\sigma_i = Cs_{i,q}$ and $\gamma_i = \sum_{\tau_j \neq \tau_i} \Big(\max_{\substack{R_l \in R^G_{C_s},\ l \neq q \\ \wedge\ \tau_j \in \tau_{l,s}}} \{Cs_{j,l}\} \Big) / m_s$ which are also constants.

Thus by combining Equations 10, 11, and 36, we can rewrite Equation 35 as follows (e.g., for FIFO queues):

$$\sum_{\substack{R_q \in R^G_{C_k} \\ \wedge\ \tau_i \in \tau_{q,k}}} \alpha_{i,q} \sum_{l \neq k} \sum_{\substack{\forall \tau_i,\ \tau_i \in \tau_{q,l} \\ \wedge\ R_q \in R^G_{C_l}}} (\sigma_i + \frac{\gamma_i}{m_l}) \leq d_i - \frac{\delta_i}{m_k} \tag{37}$$

Finally, it is easy to see that Equation 37 can be rewritten in the following form:

$$\sum \frac{c_l}{m_l} \leq b_i \tag{38}$$

The requirement in Equation 38 is a nonlinear inequality constraint which adheres to the form of constraint for a NIP problem (Equation 32). Thus every requirement in $Q_k(m_k)$ of every component C_k will generate a nonlinear inequality constraint. Furthermore, every component C_k generates the inequality constraint $m_k^{(min)} \leq m_k \leq m_k^{(max)}$ which can be divided into two inequalities $m_k \geq m_k^{(min)}$ and $m_k \leq m_k^{(max)}$. Obviously $m_1, \cdots$, and m_k are integers, thus the integration of the real-time components on a multiprocessor platform under C-MSOS can be modeled as a NIP problem.

VIII. Experimental Evaluation

We have performed experimental evaluation to investigate the performance of C-MSOS for its four different alternatives where global queues are FIFO-based or Round-Robin-based and the local queues (to access global resources) are FIFO-based or Priority-based.

A. Experiment Setup:

To determine the performance of all four alternatives we tested the schedulability of a set of randomly generated components on a multiprocessor platform under each alternative and according to different settings. For each setting, the number of components was varied from 2 to 22, and each component was allocated on 3 or 5 processors (processors per component). The number of components sharing each global resource was chosen between 2 and 12 (components per resource), and the number of tasks per each component sharing a global resource was varied from 2 to 12 (tasks per component per resource). For each component a task set was randomly generated where the utilization of each task was randomly chosen between 0.01 and 0.1, and its period was randomly chosen between $10ms$ and $100ms$, and the execution time of the task was calculated based on its utilization and period. For each component, tasks were generated until the utilization of the component reached a cap or a maximum number of 30 tasks were generated. The utilization cap of a component was set to be the number of processors of the component multiplied by 0.4. A task included up to 4 critical sections, and the total number of shared global resources was 8 or 16. The length of global critical sections ranged from $10\mu s$ to $150\mu s$. For each setting we generated 1000 platforms.

B. Results:

First we performed the experiments for the platforms that consisted of similar components, e.g., all the components sharing the global resources had the same number of tasks sharing each global resource (the number of tasks per component per resource were the same), etc. The performance of C-MSOS for its different alternatives, according to the number of components on the platform, the number of components sharing each resource, the number of tasks per component sharing each resource, and the length of critical sections per task, is illustrated in Figures 2, 3, 4, and 5 respectively. In this case (where the components are similar), the overall results show that C-MSOS mostly performs better if the local

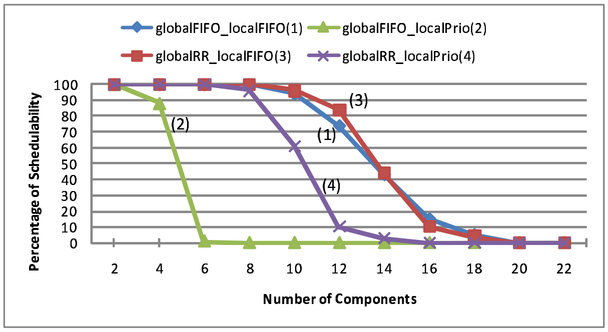

Fig. 2. Schedulability of C-MSOS by increasing the number of components on the platform. 3 processor per component, 8 global resources each shared by half of the components from which 4 tasks share the resource, up to 4 critical sections per task each with length of 40 μs.

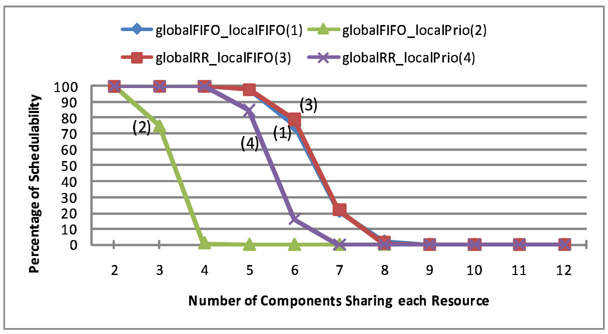

Fig. 3. Schedulability of C-MSOS by increasing the number of components sharing each resource. 12 component, 3 processor per component, 8 global resources each shared, 4 tasks per component sharing a global resource, up to 4 critical sections per task each with length of 40 μs.

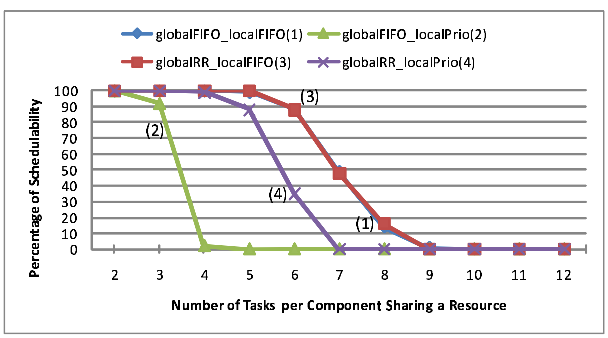

Fig. 4. Schedulability of C-MSOS by increasing the number of tasks per component per resource. 12 component, 3 processor per component, 8 global resources each shared by 4 components, up to 4 critical sections per task each with length of 40 μs.

queues are FIFO-based. When using FIFO-based local queues, C-MSOS performs similar for both FIFO-based and Round-Robin-based global queues. However, using prioritized local queues, C-MSOS mostly performs better by using Round-Robin-based global queues.

Second, we performed experiments where each generated platform consisted of components with different degree of resource sharing. This is closer to reality where components may differ in their settings, e.g., the number of tasks per component sharing a global resource can be different for

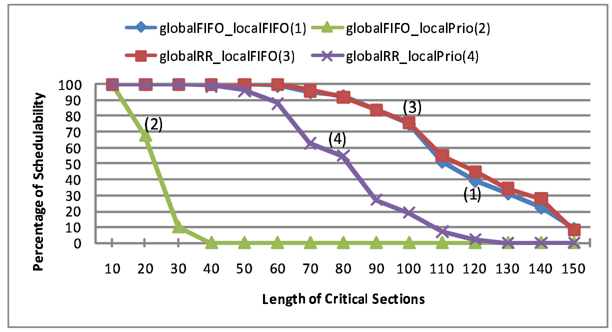

Fig. 5. Schedulability of C-MSOS by increasing the length of critical sections (in μs). 12 component, 3 processor per component, 8 global resources each shared by 4 components from which 4 tasks share the resource, up to 4 critical sections per task.

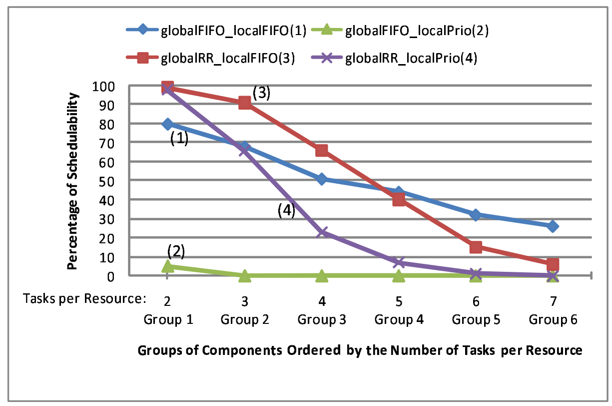

Fig. 6. Schedulability under C-MSOS for components with different number of tasks per resource. 12 component, 3 processor per component, 8 global resources each shared by 6 components, up to 4 critical sections per task each with length of 80 μs.

different components. Looking at the schedulability analysis in Section V-B3, an important factor for which the different alternatives of C-MSOS may perform differently is the degree of resource sharing in each component, e.g., a component may benefit better under an alternative of C-MSOS depending on the number of its tasks that share each global resource. We performed experiments in which each generated platform consisted of components with different number of tasks sharing each global resource. we generated 1000 platforms consisting of 12 components. For each platform we divided its 12 components into 6 groups (2 components per group); where beginning from the first group to the sixth group they included 2, 3, 4, 5, 6, and 7 tasks sharing each resource respectively. The results in Figure 6 illustrates the average percentage of schedulable components of each group under different alternatives of C-MSOS. As shown in Figure 6, for any type of components the alternative of C-MSOS where the global queues are FIFO-based and the local queues are priority-based (FP) is always outperformed by other alternatives. In fact this alternative (FP) was never better than any other alternatives for any settings. Furthermore, the components that share global resources, and include only 2 tasks per each shared global resource, perform better under the both alternatives that use Round-Robin-based global queues (RF and RP) compared to the alternative where both global and local queues are FIFO-based (FF). The alternative RF (Round-Robin global queues and FIFO local queues) performs better for the components that have less than 5 tasks per each global resource they share while FF (FIFO global queues and FIFO local queues) alternative performs better for the component with 5 and more tasks per each global resource they share. Alternative FF performs better than RP even for components with 3 and more tasks per each global resource the components share. Given any type of global queues, all types of components benefit more from FIFO-based local queues rather than priority-based queues, i.e., FF and RF always outperform FP and RP respectively.

IX. Conclusion

In this paper, we have generalized our recently proposed protocol (MSOS) [5] which handles resource sharing among real-time components on a multi-core platform where each component is allocated on one dedicated processor. In this paper we have developed a new locking protocol (C-MSOS) to handle resource sharing among components where each component is statically allocated on multiple dedicated processors (one cluster). We have also assumed that the tasks within each component are scheduled using global fixed priority preemptive scheduling policy.

In C-MSOS each component is abstracted and represented by an interface which abstracts the information about global resources it shares with other components. Furthermore, the interface includes a set of requirements that should be satisfied for the component to be schedulable when it co-executes with other components on a shared multi-core platform. This offers the possibility to develop different real-time components in parallel and independently and their schedulability analysis can be performed and abstracted in their interfaces.

In the future we plan to implement C-MSOS under real-time operating systems (RTOS) and study its performance. We also plan to study legacy real-time components and attempt to extract interfaces for them according the interface model of C-MSOS. C-MSOS is based on shared memory synchronization, hence an interesting future work is to study resource sharing among real-time components on a multiprocessor platform by means of message passing approaches.

References

[1] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *Handbook on Scheduling Algorithms, Methods, and Models*. Chapman Hall/CRC, Boca, 2004.

[2] U. Devi. Soft real-time scheduling on multiprocessors. In *PhD thesis, available at www.cs.unc.edu/~anderson/diss/devidiss.pdf*, 2006.

[3] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger. A hybrid real-time scheduling approach for large-scale multicore platforms. In *proceedings of 19th IEEE Euromicro Conference on Real-time Systems (ECRTS'07)*, pages 247–258, 2007.

[4] Theodore P. Baker and Sanjoy K. Baruah. Schedulability analysis of multiprocessor sporadic task systems. In *Handbook of Realtime and Embedded Systems*, 2007.

[5] F. Nemati, M. Behnam, and T. Nolte. Independently-developed real-time systems on multi-cores with shared resources. In *proceedings of the 23th IEEE Euromicro Conference on Real-time Systems(ECRTS'11), to appear*, 2011.

[6] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS'03)*, pages 2–13, 2003.

[7] M. Behnam, I. Shin, T. Nolte, and M. Nolin. SIRAP: a synchronization protocol for hierarchical resource sharingin real-time open systems. In *proceedings of the 7th ACM & IEEE International conference on Embedded software (EMSOFT'07)*, pages 279–288, 2007.

[8] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *proceedings of 20th IEEE Euromicro Conference on Real-time Systems (ECRTS'08)*, pages 181–190, 2008.

[9] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, 1991.

[10] P. Gai, G. Lipari, and M. D. Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *proceedings of 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, pages 73–83, 2001.

[11] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In *proceedings of 13th IEEE Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*, pages 47–56, 2007.

[12] B. Brandenburg and J. Anderson. Optimality results for multiprocessor real-time locking. In *proceedings of 31th IEEE Real-Time Systems Symposiumst (RTSS'10)*, pages 49–60, 2010.

[13] A. Easwaran and B. Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In *proceedings of 30th IEEE Real-Time Systems Symposium (RTSS'09)*, pages 377–386, 2009.

[14] C. Bialowas. Achieving Business Goals with Wind Rivers Multicore Software Solution. *Wind River white paper*.

[15] N. Fisher, M. Bertogna, and S. Baruah. Resource-locking durations in edf-scheduled systems. In *proceedings of 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'07)*, pages 91–100, 2007.

[16] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *proceedings of 22nd IEEE Real-Time Systems Symposium (RTSS'07)*, pages 149–160, 2007.

[17] D. Li and X. Sun. Nonlinear integer programming. Springer, 2006.