Resource Freshness in Real Time Scheduling

Shyhtsun F. Wu *   *Gail E. Kaiser* †

Columbia University
Department of Computer Science
New York, NY 10027
Tel: 212-854-8182, Fax: 212-666-0140
{wu, kaiser}@cs.columbia.edu

Technical Report CUCS-014-92

April 29, 1992

Abstract

We define and study a *real-time producers/ consumers* problem, where the production/consumption relations are *nondeterministic*. In other words, a consumer can get one unit of resources from any one of the producers, and one producer can provide one unit of resources for any one of the consumers. Furthermore, a producer for one type of resources might be the consumer for another type of resources. A class of *fresh resource allocation policies* is developed not only to make resources *available* for periodic real-time tasks but also to control the *resource freshness*. We first model the real-time scheduling problem as a feasible *minimum cost flow (MCF)* problem. Then, by defining the *linear cost function* differently, we get resource allocation policies for making trade-offs between *resource freshness* and *task schedulability*. Finally, we show that our result can be used to extend the *priority ceiling protocol (PCP)* to work with *general unbounded semaphores*, where $Wait$ and $Signal$ operations need not be in pairs in one task.

Keywords: Real-Time Scheduling, Resource Freshness, Resource Allocation, Semaphores.

*Wu is supported in part by an IBM Fellowship and in part by the Center for Telecommunication Research.

†Kaiser is supported by National Science Foundation grants CCR-9106368, CCR-9000930 and CCR-8858029, by grants from AT&T, BNR, DEC, IBM and SRA, by the New York State Center for Advanced Technology in Computers and Information Systems and by the NSF Engineering Research Center for Telecommunications Research.

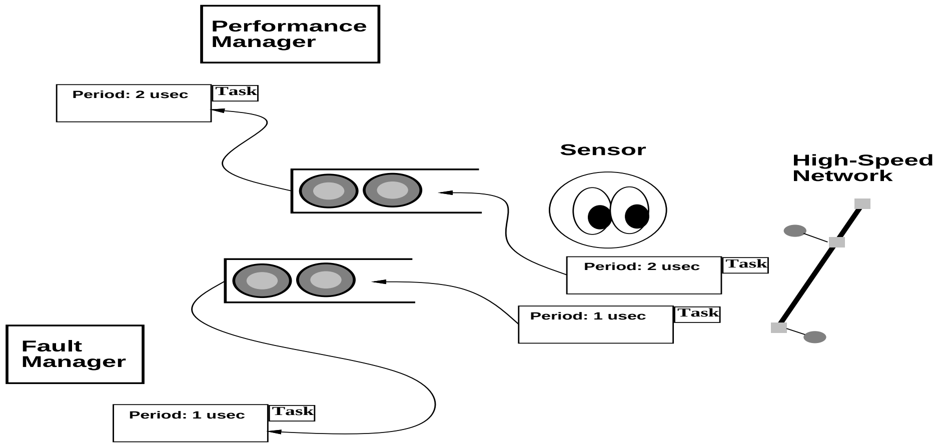

Figure 1: A Simple Network Monitoring System (4 Tasks)

1 Introduction

Imagine that *having breakfast* is a real-time task for an employee because he won't wake up until 8:00 a.m. each day, he has to attend an important group meeting at 9:00 a.m., and he gets less than an hour to drink some coffee and to swallow two donuts. Here, he has to worry about two things: first, he should not miss the meeting (i.e., he should make sure that he could finish the breakfast before 9:00 a.m.); second, both coffee and donuts are *fresh*, *i.e.,* the time between the production and the consumption of the coffee and donuts is bounded. Furthermore, the production processes for both coffee and donuts themselves have these two concerns, too: first, they have to beat the deadline to match the production plan; second, they need fresh materials to make fresh products.

We encounter a similar problem in real-time network monitoring [WK90]. In this domain, a network sensor is set up to trace different status variables of one high-speed network switch *periodically* for several network controllers. For example, a performance manager is interested in knowing whether the number of rejected packets is crossing the pre-set threshold or not in every two micro-seconds. A fault manager wants to know the operational status every one micro-second. Therefore, four periodic tasks are involved in this real-time monitoring system, as shown in figure 1. Here, the *network information* sent to the network managers is treated as resources. Thus, the freshness of resources is important because out-of-date information is not useful in monitoring a high-speed network. In a more complicated network monitoring system as in figure 2, the decisions made by one manager may also be treated as information resources by other managers.

We observe two properties about resources for both problems:

1. Resources must be available;

2. Resources must be *fresh*.

We transform these two properties into two optimization problems:

1. Given a set of tasks with certain resource constraints, what is the optimal schedule to achieve the highest processor utilization factor [LL73];

2. Given a set of tasks with certain resource dependency relations, what is the optimal schedule to achieve the best freshness for all relations under the assumption that the set of tasks is schedulable.

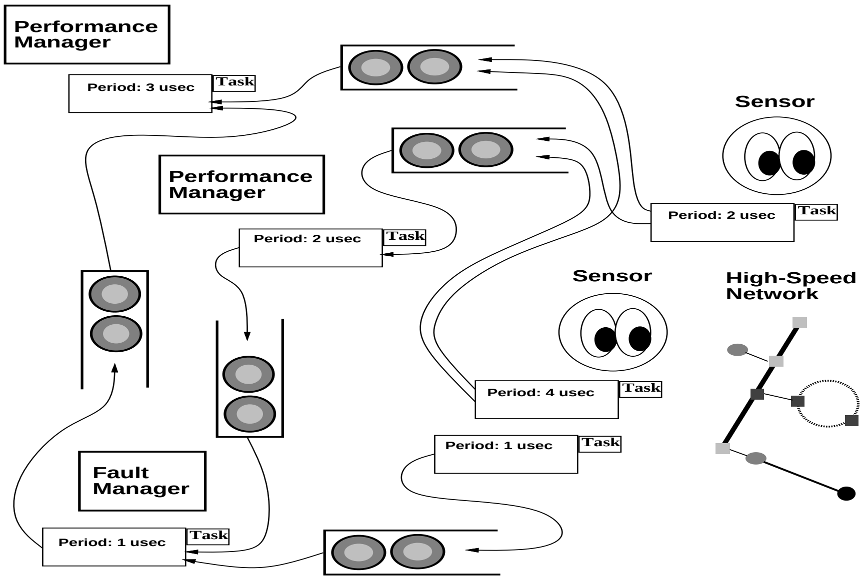

Figure 2: A More Complex Network Monitoring System (6 Tasks)

These two problems can not be optimized simultaneously, and optimizing one goal would usually hurt the other. For example, if the employee wants to get the optimally fresh donuts, he would *wait* in the donut shop for about 20 minutes for the newly baked donuts. However, if the employee wants to do a lot of other things from 8:00 a.m. to 9:00 a.m. each day, he should buy the donuts the day before and swallow two hard and cold donuts from his own refrigerator in the morning.

Thus, in this paper, we develop a class of *fresh resource allocation policies* that would allow us to play the trade-off between these two conflicting goals. In the next section, we formally define this real-time producers/consumers problem [1]. Then, we explain why the *priority ceiling protocol* doesn't work in this situation. In the following section, this problem is transformed into a *minimum cost flow (MCF)* problem. After defining a *resource balance* condition, we show, under that particular condition, this special MCF problem is feasible, *i.e.,* there is at least one solution. Next, a class of freshness resource allocation policies associated with the MCF problem are described. Finally, we show that our result can be used to extend the *priority ceiling protocol (PCP)* to work with general unbounded semaphores, where $Wait$ and $Signal$ operations are not in pairs in one task.

2 The Real-Time Producers/Consumers Problem

In this section, we formally define the problem as well as a few related concepts: *resource vector, task, occurrence, production/ consumption relations*, and *resource freshness.*

2.1 Tasks and Occurrences

A periodic task in a real-time producers/consumers problem with M types of resources has the following properties:

- Every occurrence of the task must be finished before its deadline;

[1]Please note that the producers/consumers problem we discuss here is different from the one in [JP91]. We address the difference in section 7.

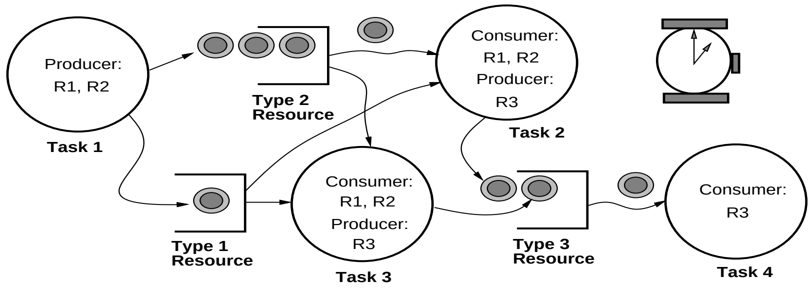

Figure 3: An Example of Real-Time Producers and Consumers

- If the task needs to consume some resources to finish its work, it must get the resources before it can finish the entire work;
- If the task would produce some resources for the system, it would produce them before finishing the entire work;
- The resource buffer size is infinite, and initially, a buffer could either be empty or have a finite amount of resources.

For example, in figure 3, we have three types of resource and four periodic tasks. Task τ_1 is a producer of resource types 1 and 2, while Task τ_2 is a consumer of resource types 1 and 2 as well as a producer of resource type 3.

Given these three properties, we can formally define what a periodic task is in a producers/consumers system:

Definition 1 (Resource Vector, $\mathcal{RV}$) *In a real-time system, if there are M distinct types of resource, then a resource vector $\mathcal{RV} = [RV_1, RV_2, \cdots, RV_m, \cdots, RV_M]$ is an M-tuple vector, where RV_m is the number of type m resources in $\mathcal{RV}$, and $RV_m \geq 0$ for $1 \leq m \leq M$ (you can not consume or produce a negative amount of resources).*

Definition 2 (Task, τ_i) *In a real-time producers/ consumers system, a periodic task τ_i is characterized with the following 6 attributes:*

t_i^1	:	*the starting time of τ_i's first occurrence (phase differences).*
T_i	:	*the periodic cycle length of τ_i.*
PRI_i	:	*the priority of τ_i.*
$WCET_i$	:	*the worst case execution time of τ_i.*
$\mathcal{P}_i$	:	*an $\mathcal{RV}$ for the resources produced by τ_i.*
$\mathcal{C}_i$	:	*an $\mathcal{RV}$ for the resources consumed by τ_i.*

Definition 3 (k-th occurrence of Task τ_i, J_i^k) *$\forall i, \tau_i$ has a sequence of occurrences. If k is the index of the sequence, then J_i^k is the k-th occurrence of τ_i. Furthermore, the starting time of J_i^k, $t_i^k = t_i^1 + [(k-1) \times T_i]$.*

For instance, in figure 3, we have only three different types of resource, which means all the $\mathcal{RV}$'s are 3-tuple vectors. Thus, since τ_3 consumes both type 1 and type 2 resource and only produces one unit of type 3 resource, $\mathcal{C}_3 = [1, 1, 0]$ and $\mathcal{P}_3 = [0, 0, 1]$. Similarly, we have $\mathcal{P}_1 = [1, 1, 0]$, $\mathcal{C}_1 = [0, 0, 0]$, $\mathcal{P}_2 = [0, 0, 1]$, $\mathcal{C}_1 = [1, 1, 0]$, $\mathcal{P}_4 = [0, 0, 0]$, and $\mathcal{C}_1 = [0, 0, 1]$,

2.2 Production/Consumption Relations and Resource Freshness

The key in a producer/consumer problem is the production/consumption relations between two tasks. For example, τ_3 and τ_4 have such a relation based on type 3 resource. At the same time, τ_2 and τ_4 also have the same relation. We describe these relations in the following definition:

Definition 4 (Production/Consumption Relation, $\pi^r(\tau_i, \tau_j)$**)** $\forall i, j, r,$

$$\pi^r(\tau_i, \tau_j) = 1 \Longleftrightarrow (\mathcal{P}_i[r] > 0) \wedge (\mathcal{C}_j[r] > 0),$$

where τ_i, τ_j are tasks (they could be the same task), and r is the resource type.

Thus, $\pi^2(\tau_1, \tau_4) = 0$ because $\mathcal{P}_1[2] = 1$ but $\mathcal{C}_4[2] = 0$. Similarly, $\pi^3(\tau_2, \tau_4) = 1$ because $\mathcal{P}_2[3] = 1$ and $\mathcal{C}_4[3] = 1$. However, $\pi^r(\tau_i, \tau_j) = 1$ doesn't necessarily imply that one occurrence J_i^k must consume the type r resource produced by another occurrence J_j^l, where k and l are occurrence indices. It only implies the possibility that the production/consumption relation is positive. For example, $\pi^3(\tau_2, \tau_4) = \pi^3(\tau_3, \tau_4) = 1$, but the occurrence J_4^k would consume the resource from either J_2^l or J_3^m, but not both. This nondeterministic situation, as we will point out in the next section, makes it hard for the scheduler to decide which task should go first. Therefore, in section 4.2, we show how to remove some relations and make the whole system deterministic. The removal process is based on a resource freshness property attached to each $\pi^r(\tau_i, \tau_j)$.

Please note that $\pi^r(\tau_i, \tau_j)$ states only the resource dependency relation between τ_i and τ_j. It does not specify which occurrences of both tasks could *potentially* have this relation, and in general, this relation can be applied to ANY two occurrences in τ_i and τ_j.

Definition 5 (Resource Freshness of $\{\pi^r(\tau_i, \tau_j), J_i^k, J_j^l\}$**,** $RF^r(J_i^k, J_j^l)$**)** $\forall i, j, k, l, r,$

$$\pi^r(\tau_i, \tau_j) = 1 \Longrightarrow RF^r(J_i^k, J_j^l) = t_{(r \rightsquigarrow J_j^l)} - t_{(J_i^k \rightsquigarrow r)},$$

where $t_{(r \rightsquigarrow J_j^l)}$ is the time J_j^l consumes r, and $t_{(J_i^k \rightsquigarrow r)}$ is the time J_i^k produces r.

3 Why doesn't PCP Work?

The Priority Ceiling Protocol (PCP) was developed for solving the *reusable resource sharing* problem where the total number for any type of resource existing in the system is *static* over time. For example, if the total number is one, then that unit of resource must either be free or held by one particular task, and this is true at any time. In other words, if a task grabs certain units of type m resource, then it must release the same amount of type m resource before the end of the task (in order to make sure the static property is true at ANY time). The original PCP [SRL90, Raj91, CL90] was proposed for binary semaphores, which implies the total number for any type of resource is statically one. Baker [Bak91] and Chen and Lin [CL91] provide protocols for total numbers greater than one.

The real-time producers/consumers problem defined in section 2 does not have that *static* property because a task is not required to produce the same amount of resources it has consumed earlier. Similarly, a task does not have to consume the same type of resources it produces. It can consume either NOTHING or SOMETHING ELSE before it produces a particular type of resources.

The lack of the static resource property makes it difficult to use PCP to solve this real-time producers/consumers scheduling problem because the priority inheritance relationship is nondeterministic. For example, task τ_i with priority PRI_i is blocked waiting for S type of resource, when both τ_j and τ_k with priority PRI_j and PRI_k, respectively, can produce S, and $PRI_i > PRI_j > PRI_k$. So, should τ_j or τ_k

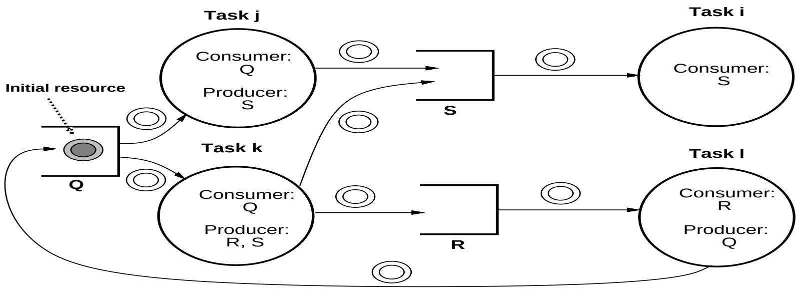

Figure 4: An Example with Four Tasks

inherit the priority PRI_i? In [CL91], it is proposed that the higher priority task (i.e., τ_j) should inherit the priority successfully. However, this strategy would work only when the static resource property holds. For instance, in figure 4, τ_j would consume $\{Q\}$ and produce $\{S\}$, while τ_k would consume $\{Q\}$ and produce $\{R, S\}$. Now, if we assume that initially the system has only one unit of Q, then τ_k should inherit PRI_i. Otherwise, τ_k and τ_l can never be executed because the only unit of Q would have been consumed by τ_j and neither τ_j nor τ_i produce Q.

4 Tranforming into a Minimum Cost Flow Problem

In order to solve the real-time producers/consumers problem, we transform it into a minimum cost flow problem first. Then, we develop solutions for that transformed problem. In this section, we show how to translate all the task occurrences in one hyper-cycle (denoted as T_{hyper}) into an AND/XOR graph. Then, the MCF constraints about the AND/XOR graph are built. Finally, we prove that the MCF setting is always feasible. This implies all the existing results about MCF can be used to solve our problem.

4.1 AND/XOR Graphs

An AND/XOR graph contains four types of nodes (figure 5):

$\boxed{^mAND_1}$ **:** This type of node has m input arcs and 1 output arc. The node will output one unit of resource to the only output arc if and only if all the input arcs have one unit coming in. Otherwise, the output flow is zero. We will use $\boxed{^mAND_1}$ to model the situation that a task occurrence needs m units of different types of resources to generate 1 unit of resource for the system.

$\boxed{^1AND_m}$ **:** This type of node has 1 input arc and m output arcs. The node will output one unit of resource to each of the output arcs if and only if the only input arc has one unit coming in. Otherwise, each of the output flows are zero. We will use $\boxed{^1AND_m}$ to model the situation that a task occurrence needs 1 unit of resource to generate m units of different types of resources for the system.

$\boxed{^mXOR_1}$ **:** This type of node has m input arcs and 1 output arc. The node will output one unit of resource to the only output arc if and only if exactly one of the input arcs has one unit coming in. Otherwise, the output flow is zero. We will use $\boxed{^mXOR_1}$ to model the situation that a task occurrence needs 1 unit of one particular type of resource, while m other different task occurrences can provide that kind of resource, to generate 1 unit of resource for the system.

$\boxed{^1XOR_m}$ **:** This type of node has 1 input arc and m output arcs. The node will output one unit of resource to exactly one of its output arcs if and only if the only input arc has one unit coming in.

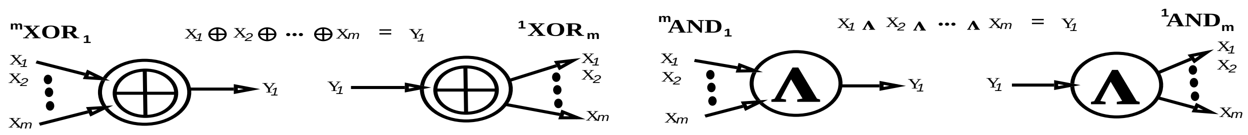

Figure 5: Four types of Nodes in an AND/XOR graph

Otherwise, all the output flows are zero. We will use $\boxed{{}^{1}XOR_m}$ to model the situation that a task occurrence needs 1 unit of resource to generate 1 unit of one particular type of resource, while m other different task occurrences would possibly need this generated resource.

Given a set of tasks, we can use these four types of nodes to model all the task occurrences in one hyper-cycle. The reason for modeling only one hyper-cycle is that if this hyper-cycle can be scheduled, then all the hyper-cycles in the system can be scheduled. The AND/XOR graph transformation process is described as the following algorithm[2]:

Algorithm 1 AND/XOR Graph Transformation

INPUT: $\forall i, j, k, r, 1 \leq k \leq \frac{T_{hyper}}{T_i}, J_i^k, \pi_r(\tau_i, \tau_j)$;

OUTPUT: *an AND/ XOR graph* $G = (V, E)$*;*

METHOD: *(Sketched)*

1. *For each task occurrence* J_i^k*:*
   (a) *We create a* $\boxed{{}^{1}AND_m}$ *node, where* m *is the number of non-zero elements in the* $\mathcal{P}_i$ *vector (i.e.,* $m = \sum_{\forall r} \mathcal{P}_i[r]$*).*
   (b) *Furthermore, we create a* $\boxed{{}^{m}AND_1}$ *node, where* m *is the number of non-zero elements in* $\mathcal{C}_i$ *(i.e.,* $m = \sum_{\forall r} \mathcal{C}_i[r]$*).*
   (c) *The* $\boxed{{}^{m}AND_1}$ *and* $\boxed{{}^{1}AND_m}$ *nodes link together using the "1" sides. (Please note that these two nodes may have different* m*.)*
   (d) *Then, for each of the non-zero elements in* $\mathcal{P}_i$*,* $\mathcal{P}_i[r]$*, we create a* $\boxed{{}^{1}XOR_m}$ *node with a label* r*, where* $m = \sum_{\forall j} \pi^r(\tau_i, \tau_j) \times \frac{T_{hyper}}{T_j}$*.*
   (e) *All the* $\boxed{{}^{1}XOR_m}$ *nodes link to the* $\boxed{m}$ *side of the* $\boxed{{}^{1}AND_m}$ *node.*
   (f) *Similarly, for each of the non-zero elements in* $\mathcal{C}_i$*,* $\mathcal{C}_i[r]$*, we create a* $\boxed{{}^{m}XOR_1}$ *node with a label* r*, where* $m = \sum_{\forall j} \pi^r(\tau_j, \tau_i) \times \frac{T_{hyper}}{T_j}$*.*
   (g) *All the* $\boxed{{}^{m}XOR_1}$ *nodes link to the* $\boxed{m}$ *side of the* $\boxed{{}^{m}AND_1}$ *node.*
2. *For each pair of* $\boxed{{}^{1}XOR_m}$ *and* $\boxed{{}^{m}XOR_1}$ *nodes in task occurrences* J_i^k *and* J_j^l*, respectively, we will put an edge if and only if:*
   (a) *the resource label of both nodes are the same, and*
   (b) *assuming that the resource label is* r*, then* $\pi^r(\tau_i, \tau_j) = 1$*.*

Example 1 *Consider the four tasks in figure 3, where we assume that* $T_1 = T_4 = 1$ *and* $T_2 = T_3 = 2$*. Then, since* $T_{hyper} = LCM_{1 \leq i \leq 4}\{T_i\} = LCM\{T_1, T_2, T_3, T_4\} = 2$*,* τ_1 *and* τ_4 *have two occurrences in one hyper-cycle, while* τ_2 *and* τ_3 *have only one. Therefore, the input set of occurrences is* $\{J_1^1, J_1^2, J_2^1, J_3^1, J_4^1, J_4^2\}$*. We get the AND/ XOR graph in figure 6.*

[2]From now on, we assume the elements in any resource vector $\mathcal{RV}$ are either 0 or 1. However, our result can be easily extended to drop the assumption.

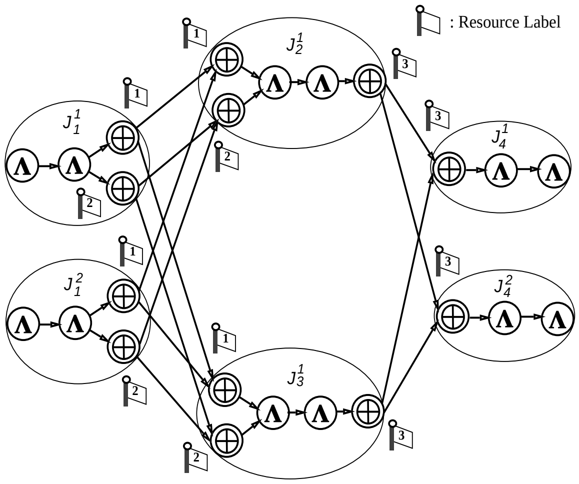

Figure 6: The AND/XOR Graph for all the Occurrences

4.2 MCF Problem Setting

The *minimum cost flow* problem [Roc84, BJS90] is to find a set of arc flows that would optimize a linear cost function under two types of constraints simultaneously:

- **The conservation of flow constraints:** for one particular node, the difference between the input and output flow is fixed;
- **The capacity constraints:** for one particular edge, the flow put on that edge is bounded.

Definition 6 (Minimum Cost Flow Problem) *minimized* $\sum_{(i,j)\in E} a_{ij}x_{ij}$*, subject to*

$$\sum_{\{j|(i,j)\in E\}} x_{ij} - \sum_{\{j|(j,i)\in E\}} x_{ji} = s_i, \forall i \in V, \quad b_{ij} \leq x_{ij} \leq c_{ij}, \forall (i,j) \in E,$$

where a_{ij}, b_{ij}, c_{ij}*, and* s_i *are given scalars.*

Our plan is to build an MCF problem on the AND/XOR graph we get in the previous section. The modeling is done the following way:

1. $\forall i, j, b_{ij} = 0, c_{ij} = 1$;
2. If a node i is $\boxed{{}^m AND_1}$, then $s_i = (1 - m)$;
3. If a node i is $\boxed{{}^1 AND_m}$, then $s_i = (m - 1)$;
4. If a node i is either $\boxed{{}^m XOR_1}$ or $\boxed{{}^1 XOR_m}$, then $s_i = 0$.

The first item in our modeling restricts the flow on each edge between 0 and 1. However, in the AND/XOR graph, since all the inputs are integers, the optimal feasible solution would contain only integers. Therefore, the values of all the x_{ij}'s are either 0 or 1. The rest of the items in the modeling follow the definitions of an AND/XOR graph. For example, for a $\boxed{{}^m AND_1}$ node, there are m input arcs and 1 output arc. Since this node must be executed, we need all the input arcs to have a flow with one unit of resource, *i.e.,* the in-flow is m and the out-flow is 1. Thus, the flow difference is $1 - m$ as described in item 2 above.

The intuition for x_{ij} in the real-time producers/consumers problem is whether job i is the producer of job j. If $x_{ij} = 1$, then the answer is "yes." And, the constraints in the MCF setting guarantee that all the resource dependency relations are *deterministic*, *i.e.*, one producer matches one consumer. Furthermore, if $x_{ij} = x_{jk} = 1$, we call job i is the *transitive producer* of job k.

Definition 7 (Transitive Producer, x^+_{ik}) *Job i is the transitive producer of job k, i.e., $x^+_{ik} = 1$, if and only if $x_{ik} = 1$ or $\exists j, (x^+_{ij} = 1) \wedge (x^+_{jk} = 1)$.*

4.3 Resource Balance

We have shown how to translate a set of real-time tasks into an AND/XOR graph as well as how to model an MCF problem on that particular graph. Before considering the *linear cost function*, we check whether this MCF problem is feasible. In this subsection, we introduce a sufficient condition for the MCF's feasibility.

Definition 8 (Type r Resource Balance Condition) *A set of tasks, $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, is type r resource balanced if and only if $\sum_{\forall i}(\frac{T_{hyper}}{T_i} \times (\mathcal{P}_i[r] - \mathcal{C}_i[r])) = 0$.*

Definition 9 (Global Resource Balance Condition) *$\mathcal{T}$ is globally resource balanced if and only if $\forall r$, $\mathcal{T}$ is type r resource balanced.*

The *type r resource balance condition* states that if we have k units of type r resource, then after one hyper-cycle, we still have k units. Furthermore, if, for all types of resource, we have this property, we get the *global resource balance condition*. In the following theorem, we show that this strong condition is sufficient for the MCF's feasibility.

Theorem 1 (Feasibility of MCF($\mathcal{T}$)) *$\mathcal{T}$ is globally resource balanced $\Longrightarrow$ MCF($\mathcal{T}$) is feasible.*

proof:

A $\boxed{{}^{m}AND_1}$ node is *marginal* if $m = 0$. Intuitively, a task $\tau_i \in \mathcal{T}$ has one marginal $\boxed{{}^{m}AND_1}$ node if and only if τ_i consumes nothing. Similarly, a $\boxed{{}^{1}AND_m}$ is *marginal* if $m = 0$. The intuition is if τ_i has one such $\boxed{{}^{1}AND_m}$ node, then τ_i produces nothing. Furthermore, one task must have either exactly one marginal $\boxed{{}^{m}AND_1}$ node, exactly one marginal $\boxed{{}^{1}AND_m}$ node, or no marginal nodes.

The marginal $\boxed{{}^{m}AND_1}$ nodes are all feasible because they are independent from all other nodes. The marginal $\boxed{{}^{1}AND_m}$ nodes are feasible if all the $\boxed{{}^{m}AND_1}$ nodes are feasible. In algorithm 1, we know that if the outputs of $\boxed{{}^{m}XOR_1}$ nodes are all "1," then the inputs as well as the outputs of all the non-marginal $\boxed{{}^{m}AND_1}$ nodes are also "1."

In steps 1a, 1b, and 1c of algorithm 1, we further know that the following statements are also true:

- the inputs and outputs of all the non-marginal $\boxed{{}^{1}AND_m}$ nodes are "1";
- the inputs of all the $\boxed{{}^{1}XOR_m}$ nodes are "1."

Now, we only need to prove the following two statements to prove the feasibility of MCF($\mathcal{T}$):

- exactly one input arc of any $\boxed{{}^{m}XOR_1}$ node is "1";

- exactly one output arc of any $\boxed{{}^1XOR_m}$ node is "1".

Please note, in steps 1d and 1f of algorithm 1, both $\boxed{{}^1XOR_m}$ and $\boxed{{}^mXOR_1}$ nodes have a *resource* label. Therefore, we can partition the nodes into M classes, where M is the number of total resource types.

Then, for each resource type r, because of the *type r resource balance condition*, we know that the number of r-labeled $\boxed{{}^mXOR_1}$ nodes is equal to the number of r-labeled $\boxed{{}^1XOR_m}$ nodes. Furthermore, by steps 2a and 2b of algorithm 1, from every $\boxed{{}^1XOR_m}$ node, there are edges to all the $\boxed{{}^mXOR_1}$ nodes, *i.e.*, we have a *clique* of size m. Thus, this subproblem becomes a feasible *assignment problem*.

Since, the global balance condition implies the type r resource balance condition, $\forall r$, the above two statements are true. □

For example, in figure 6, the set of tasks represented by the AND/XOR graph is globally resource balanced. We can easily check the following individual type resource balance conditions:

Occurrence Number: $T_{hyper}/T_1 = 2, T_{hyper}/T_2 = 1, T_{hyper}/T_3 = 1, T_{hyper}/T_4 = 2;$

Type 1 Resource: $\mathcal{P}_1[1] - \mathcal{C}_1[1] = 1, \mathcal{P}_2[1] - \mathcal{C}_2[1] = -1, \mathcal{P}_3[1] - \mathcal{C}_3[1] = -1, \mathcal{P}_4[1] - \mathcal{C}_4[1] = 0;$

$$\Longrightarrow 2 \times 1 + 1 \times (-1) + 1 \times (-1) + 2 \times 0 = 0.$$

Type 2 Resource: $\mathcal{P}_1[2] - \mathcal{C}_1[2] = 1, \mathcal{P}_2[2] - \mathcal{C}_2[2] = -1, \mathcal{P}_3[2] - \mathcal{C}_3[2] = -1, \mathcal{P}_4[2] - \mathcal{C}_4[2] = 0;$

$$\Longrightarrow 2 \times 1 + 1 \times (-1) + 1 \times (-1) + 2 \times 0 = 0.$$

Type 3 Resource: $\mathcal{P}_1[3] - \mathcal{C}_1[3] = 0, \mathcal{P}_2[3] - \mathcal{C}_2[3] = 1, \mathcal{P}_3[3] - \mathcal{C}_3[3] = 1, \mathcal{P}_4[3] - \mathcal{C}_4[3] = -1;$

$$\Longrightarrow 2 \times 0 + 1 \times 1 + 1 \times 1 + 2 \times (-1) = 0.$$

Since all three individual resource balance conditions are true, the set of tasks is globally balanced. Furthermore, one possible feasible solution of figure 6 is shown in figure 7. However, we do not know whether this solution is *optimal* unless we know the values of the a_{ij}'s.

In practice, we can make the MCF still feasible even if the *resource balance* condition does not hold. However, this requires that the production rate is higher than the consumption rate. The trick is to insert some "pseudo consumers" into the system, and make the whole system "balanced." Then, when we define the value a_{ij}'s, we can put "0" uniformally on those edges connecting these pseudo consumers.

5 Fresh Resource Allocation Policies or Playing a_{ij}

We have just shown a *feasible MCF* setting for the real-time producers/consumers problem. The major purpose of this setting is to remove the *nondeterministic resource dependency relations* from the AND/XOR graph (*i.e.,* from figure 6 to figure 7). After having such deterministic relations, the scheduler can decide the priorities among real-time periodic tasks. In this section, we develop two resource allocation policies based on the MCF setting. The first policy pessimistically assumes the scheduler is *vicious*, and therefore, we can not rely on the scheduler to achieve anything. In other words, we play a *non-cooperative game* with the scheduler to optimize our own interest – *resource freshness*. The second policy optimistically assumes a cooperative scheduler would help in achieving *optimal resource*

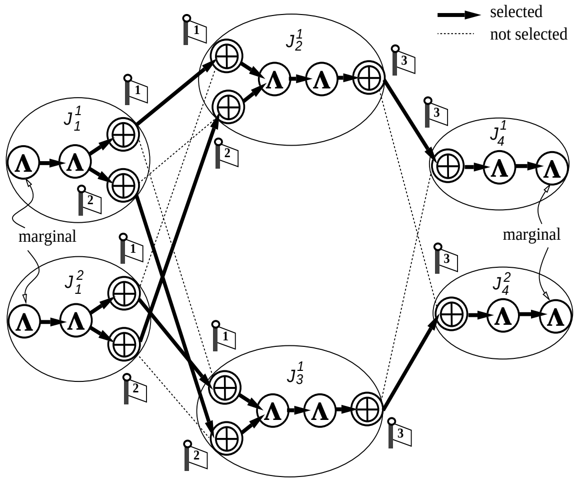

Figure 7: One Possible Feasible Solution

freshness. Thus, in order to make the consumed resources as fresh as possible, this policy would minimize the time period between the producer's starting time and the consumer's ending time. [3] For both policies discussed here, we assume that $\forall i, t_i^1$ is known.

5.1 OFUOS: Optimized Freshness Under Optimal Schedulability

Consider two job occurrences J_i^k and J_j^l, where $\exists r, \pi^r(\tau_i, \tau_j) = 1$. In Algorithm 1, there is an edge between these two occurrences because τ_j needs to consume r and τ_i can produce r. Furthermore, the edge connects one $\boxed{{}^1XOR_m}$ node and one $\boxed{{}^mXOR_1}$ node with the resource label r. Now, let's consider how to assign the value for $a_{J_i^k, J_j^l}$ in the following cases:

- If *the earliest possible time* for J_j^l to consume r, *i.e.,* $t_j^1 + (l-1) \times T_j$, is later than *the latest possible time* for J_i^k to produce r, *i.e.,* $t_i^1 + k \times T_i$, then it is obvious that the r generated by J_i^k can be consumed by J_j^l. Since we assume that the scheduler is *vicious*, the worst case freshness of this particluar r is the time between the earliest possible time for J_i^k to produce r, *i.e.,* $t_i^1 + (k-1) \times T_i$, and the latest possible time for J_j^l to consume r, *i.e.,* $t_j^1 + l \times T_j$. Therefore, in this case,

  $$a_{J_i^k, J_j^l} = \text{the freshness of } r = (t_j^1 + l \times T_j) - [t_i^1 + (k-1) \times T_i].$$

- If $(t_j^1 + (l-1) \times T_j)$ is earlier than $(t_i^1 + k \times T_i)$, then the r generated by J_i^k might not be able to be consumed by J_j^l. Remember we assume that the scheduler is *vicious* and it could schedule these two occurrences in this worst case. Furthermore, we can not ask J_j^l to *wait* for J_i^k because we don't want to rely on the scheduler at all.

  It turns out that J_j^l may consume one of the r's generated by J_i^k in previous hyper-cycles, *i.e.,* $J_i^{(k - n \times \frac{T_{hyper}}{T_i})}$, where n is the number of hyper-cycles that we need to *roll-back* in order to make:

  $$(t_j^1 + (l-1) \times T_j) \geq \{t_i^1 + [k - (n \times \frac{T_{hyper}}{T_i})] \times T_i\}.$$

  Similarly, the worst case freshness in this case would be:

  $$a_{J_i^k, J_j^l} = (t_j^1 + l \times T_j) - \{t_i^1 + [k - (n \times \frac{T_{hyper}}{T_i}) - 1] \times T_i\}.$$

[3] In order to have a simple design for the scheduler, we make sure that the producer's starting time must be earlier than the consumer's starting time.

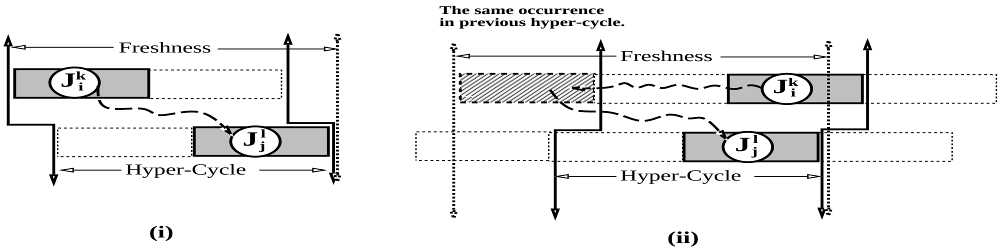

Figure 8: Two Cases in OFUOS

We summarize these two cases (figure 8) in the following policy:

Policy 1 (Optimized Freshness Under Optimal Schedulability, OFUOS) $\forall i, j, k, l,$

$$a_{J_i^k, J_j^l} = (t_j^1 + l \times T_j) - \{t_i^1 + [k - (\mathcal{N}_{J_i^k, J_j^l} \times \frac{T_{hyper}}{T_i}) - 1] \times T_i\},$$

where

$$\mathcal{N}_{J_i^k, J_j^l} = \min\{n \mid (n \geq 0) \wedge [(t_j^1 + (l-1) \times T_j] \geq [t_i^1 + (k - (n \times \frac{T_{hyper}}{T_i})) \times T_i)]\}.$$

Example 2 *Consider three tasks τ_1, τ_2, and τ_3. τ_1 and τ_2 produce one unit of r in each of their occurrences, while τ_3 consumes one unit of r periodically. Assume that $T_1 = 6, T_2 = 3, T_3 = 2$, and $t_1^1 = 0, t_2^1 = 1, t_3^1 = 2$. Since $T_{hyper} = LCM\{2, 3, 6\} = 6$, in one hyper-cycle, τ_1 has one occurrence J_1^1, τ_2 has two occurrences J_2^1 and J_2^2, and τ_3 has three occurrences, J_3^1, J_3^2, and J_3^3. So, we totally have 6 occurrences, and the corresponding AND/XOR graph is depicted in figure 9.*

Let's calculate two a_{ij}'s with OFUOS:

$a_{J_1^1, J_3^3}$: *Since J_1^1 produces r before time 6 and J_3^3 consumes r as early as time 6, too, $\mathcal{N}_{J_1^1, J_3^3} = 0$. Therefore,*

$$a_{J_1^1, J_3^3} = (2 + 3 \times 2) - (0 + (1 - 0 \times \frac{6}{6} - 1) \times 6) = 8 - 0 = 8.$$

This implies that if we let J_3^3 consume the r produced by J_1^1, the worst case freshness is "8". Intuitively, the answer is correct, as shown in figure 10, because J_1^1 may produce r at the very beginning, while J_3^3 potentially might consume that resource at the end of τ_3's cycle. Thus, the time difference is "8."

$a_{J_1^1, J_3^2}$: *Since J_1^1 produces r before time 6 and J_3^2 consumes r as early as time 4, too, $\mathcal{N}_{J_1^1, J_3^2}$ is 1. Therefore,*

$$a_{J_1^1, J_3^2} = (2 + 2 \times 2) - (0 + (1 - 1 \times \frac{6}{6} - 1) \times 6) = 6 - (-6) = 12.$$

This implies that if we let J_3^2 consume the r produced by J_1^1, the worst case freshness is "12". In figure 10, it is not always possible that J_3^2 can consume the r produced by J_1^1 because of the vicious scheduler. Therefore, for safety, J_3^2 has to consume r produced by J_1^1 in the previous hyper-cycle. Under the worst case, the time difference between the production and consumption is "12."

After calculating all the a_{ij}'s and running an MCF algorithm, we get the optimal solution $5 + 6 + 8 = 19$ as in figure 11: $J_1^1 \overset{r}{\rightsquigarrow} J_3^3$; $J_2^1 \overset{r}{\rightsquigarrow} J_3^2$; $J_2^2 \overset{r}{\rightsquigarrow} J_3^1$.

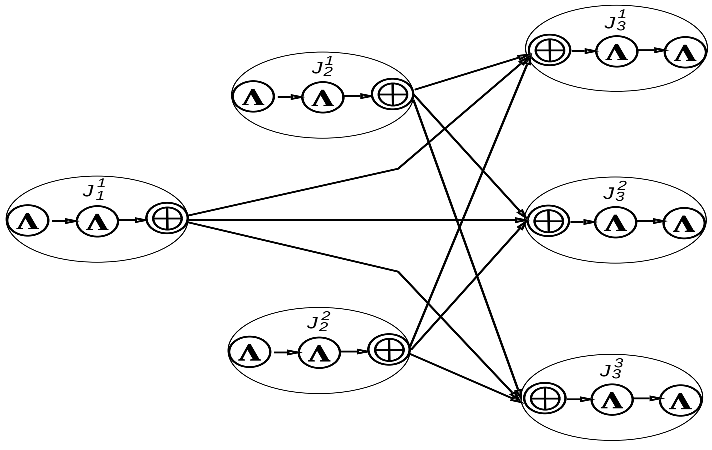

Figure 9: The AND/XOR Graph in the Second Example

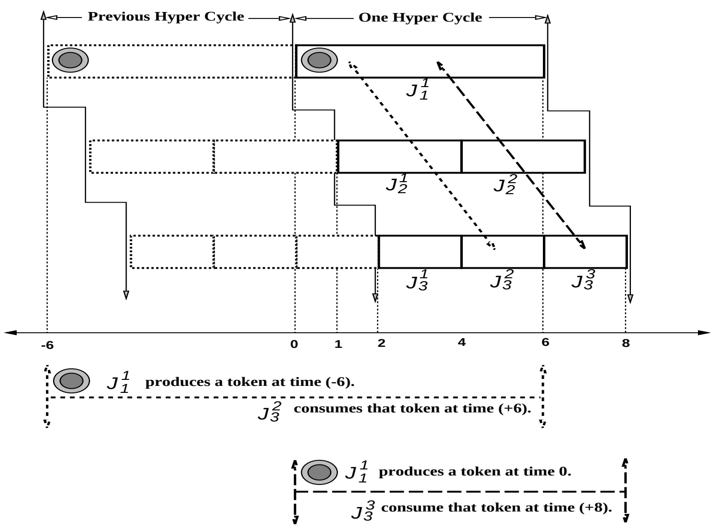

Figure 10: The Timing Diagram for the Second Example

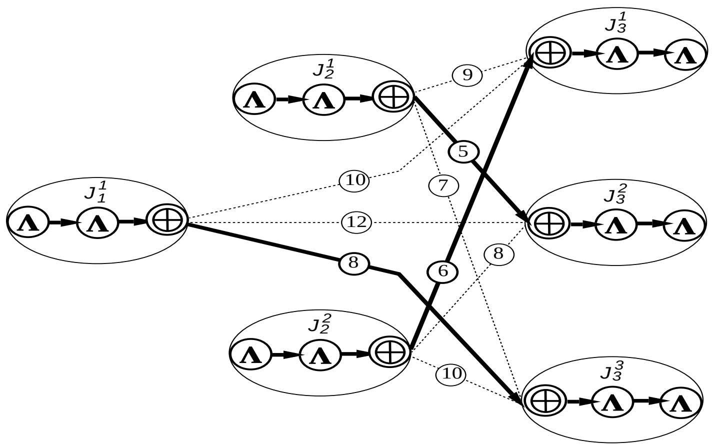

Figure 11: The Solution under OFUOS

5.2 OWCF: Optimal Worst-Case Freshness

If one unit of resource r is consumed *immediately* after its production, then r is *optimally fresh*. In reality, such freshness is hard to achieve because

- the scheduler doesn't know "exactly" when the resource will be produced and when the consumer will be ready to consume that resource (it only knows the earliest and latest possible time epochs.);
- if A depends on B and B depends on C, sometimes it is impossible to schedule these three jobs at the same time.

Alternatively, let's consider the time interval between the producer's starting and the consumer's ending. This interval represents the worst case freshness for one resource-dependency relation because

- the resource can not be produced before the starting time of the producer, and
- the resource can not be consumed after the ending time of the consumer.

Policy 2 (Optimal Worst Case Freshness, OWCF) $\forall i, j, k, l$,

$$a_{J_i^k, J_j^l} = (t_j^1 + l \times T_j) - \{t_i^1 + [k - (\mathcal{N}_{J_i^k, J_j^l} \times \frac{T_{hyper}}{T_i}) - 1] \times T_i\},$$

where

$$\mathcal{N}_{J_i^k, J_j^l} = \min\{n \mid (n \geq 0) \wedge [(t_j^1 + (l-1) \times T_j] \geq [t_i^1 + (k - (n \times \frac{T_{hyper}}{T_i}) - 1) \times T_i)]\}.$$

Please note that Policy OWCF is the same as Policy OFUOS except its definition of $\mathcal{N}_{J_i^k, J_j^l}$. The right hand side of the inequality in OWCF's $\mathcal{N}_{J_i^k, J_j^l}$ has a "-1", while the one in OFUOS does not. The reason for this difference is that OFUOS does not allow the consumer to start before its producer finishes.

Example 3 *Let's try the three tasks τ_1, τ_2, and τ_3 in Example 2. The AND/XOR graph is still the same one in figure 9.*

Let's calculate two a_{ij}'s with OWCF:

$a_{J_1^1, J_3^3}$: *In figure 10, since the starting time of J_1^1 is earlier than the starting time of J_3^2, $\mathcal{N}_{J_1^1, J_3^3} = 0$. Thus by OWCF:*

$$a_{J_1^1, J_3^3} = (2 + 3 \times 2) - (0 + (1 - 0 \times \frac{6}{6} - 1) \times 6) = 8 - 0 = 8.$$

This answer is exactly the same as in OFUOS because J_1^1 and J_3^3 don't overlap.

$a_{J_1^1, J_3^2}$: *Since J_1^1 starts earlier than J_3^2, consumes r as early as time 4, too, $\mathcal{N}_{J_1^1, J_3^2}$ is still 0. Therefore,*

$$a_{J_1^1, J_3^2} = (2 + 2 \times 2) - (0 + (1 - 0 \times \frac{6}{6} - 1) \times 6) = 6 - (0) = 6.$$

This implies that if we let J_3^2 consume the r produced by J_1^1, the worst case freshness is "6". However, the scheduler has to cooperate to achieve this bound because it has to let J_1^1 go first, then let J_3^2 consume that unit of fresh resource.

After calculating all the a_{ij}'s and running an MCF algorithm, we get one optimal solution $2+3+8 = 13$ *shown in figure 12: $J_1^1 \overset{r}{\rightsquigarrow} J_3^2$; $J_2^1 \overset{r}{\rightsquigarrow} J_3^1$; $J_2^2 \overset{r}{\rightsquigarrow} J_3^3$. In this example, the optimal solution is not unique.*

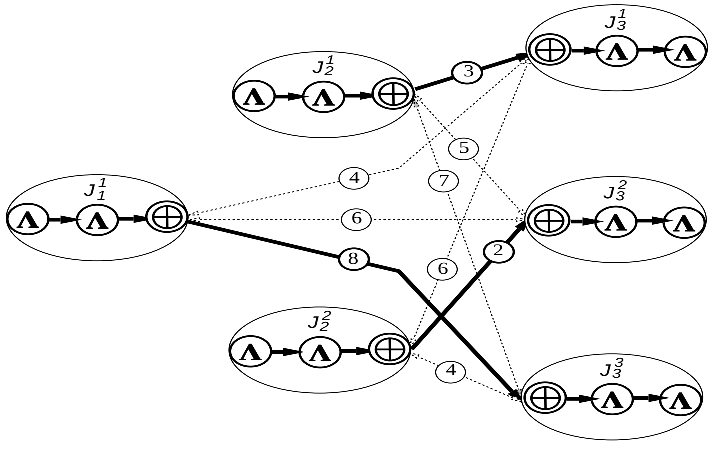

Figure 12: The Solution under OWCF

6 Extending the Priority Ceiling Protocol

In previous sections, an MCF setting is presented with two different fresh resource allocation policies. In this section, we show how to apply this MCF setting to extend the *priority ceiling protocol* to work with *general unbounded semaphores* [4]. From now on, in order to avoid confusion between binary semaphores and general unbounded semaphores, we use the word **semaphore** for binary semaphores and the word **resource** for general unbounded semaphores.

6.1 PCP/OFUOS

Since the OFUOS is independent of what scheduling policy we pick, it can work with the orginal PCP except we need to put a certain amount of resources into the system at the *initialization phase.* The reason for these initial resources is that some of the production/consumption relations selected by the MCF setting require the producer and the consumer to be in two different hyper-cycles (as shown in figure 8). This implies that the consumer in the first hyper-cycle would not get the resource because no previous hyper-cycle exists. Therefore, the system has to put some amount of initial resources to solve this problem. Let's intuitively introduce this problem in the following example:

Example 4 *Consider the three tasks τ_1, τ_2, and τ_3 in Example 2. The AND/XOR graph is still the same one as in figure 9, and using policy OFUOS, the optimal solution is depicted in figure 11. Since only one type of resource is in the system, we consider "how many units of type r resources do we need to put into the resource buffer INITIALLY?"*

Let's consider the first hyper-cycle of τ_3:

- *J_3^1 needs the resource from J_2^2 in the previous hyper-cycle. So, for J_3^1, we should definitely put "one" unit of r into the buffer initially. This "one" is equal to the $\mathcal{N}_{J_2^2,J_3^1}$ in OFUOS because $\mathcal{N}_{J_2^2,J_3^1} = 1$ means that J_3^1 has to consume the resource from J_2^2 in the previous hyper-cycle. However, the J_3^1 in the first hyper-cycle can not depend on anybody. Therefore, we need to put one unit of r initially.*
- *J_3^2 needs the resource from J_2^1 in the same hyper-cycle. Therefore, zero units of resource is needed initially for J_3^2. ($\mathcal{N}_{J_2^1,J_3^2} = 0$).*

[4] A general unbounded semaphore, S_p^∞, is a semaphore with an infinite buffer. The operation $Signal(S_p^\infty)$ can be treated as a "produce" action, while $Wait(S_p^\infty)$ is a "consume" action. In this paper, these two operations need not be paired in one task. However, we assume that all the binary semaphores are used as "critical sections."

- J_3^3 *needs the resource from* J_1^1 *also in the same hyper-cycle. Similarly, zero units of resource is needed initially for* J_3^3. *(*$\mathcal{N}_{J_1^1,J_3^3} = 0$*).*

Since, in any one hyper-cycle, J_1^1, J_2^1, J_2^2 *do not need resource* r *and* J_3^1, J_3^2, J_3^3 *need one unit of* r*, we have*

$$\begin{aligned} I^r_{OFUOS} &= 0+0+0+(1\times \mathcal{N}_{J_2^2,J_3^1}) + (1\times \mathcal{N}_{J_2^1,J_3^2}) + (1\times \mathcal{N}_{J_1^1,J_3^3}) \\ &= 0+0+0+(1\times 1)+(1\times 0)+(1\times 0) \\ &= 1. \end{aligned}$$

Totally, one unit of r *is needed initially.*

Formally, the number of initial resources for a set of tasks is computed by the following lemma:

Lemma 1 (Initial Resources, I^r_{OFUOS}**)** *The initial number of type* r *resources,*

$$I^r_{OFUOS} = \sum_{\forall J_i^k \in V} \{ \sum_{\forall J_j^l \in V} [x_{J_i^k,J_j^l} \times \mathcal{N}_{J_i^k,J_j^l} \times \pi^r(\tau_i,\tau_j)]\},$$

where V *is the AND/XOR graph constructed by Algorithm 1,* $x_{J_i^k,J_j^l}$*'s are the results from the MCF setting under OFUOS, and* $\mathcal{N}_{J_i^k,J_j^l}$*'s are defined in OFUOS.*

proof:

The equation for I^r_{OFUOS} is the summation of all the directed arcs in V. First, the term $x_{J_i^k,J_j^l}$ is to decide whether that particular arc is in the MCF optimal solution. Second, the term $\mathcal{N}_{J_i^k,J_j^l}$ is to compute the hyper-cycle difference for one pair of job occurrences (J_i^k, J_j^l). Third, the term $\pi^r(\tau_i,\tau_j)$ is to select all the arcs with resource label r because arcs without label r have $\pi^r(\tau_i,\tau_j) = 0$. □

6.2 PCP/OWCF

Two key concepts in the priority ceiling protocol are *priority inheritance* and *priority ceiling*. The purpose of *priority inheritance* among jobs is to avoid the *priority inversion* problem. And, the *priority ceiling* of binary semaphores is defined to prevent *deadlocks* as well as to reduce the blocking time due to lower priority jobs. In this subsection, both concepts are extended for $OWCF$. Furthermore, the schedulability analysis is done for this extended PCP.

6.2.1 Extended Priority Inheritance

In PCP, a job J_i^k inherits the priority P_j, where $P_j > P_i$, if and only if J_i^k is holding one binary semaphore S_p^1 and there exists an active job J_j^l, which needs S_p^1. This inheritance makes no jobs with priorities between P_i and P_j preempt J_i^k and thus indirectly preempt J_j^l.

We have also inheritance relations among jobs through general unbounded semaphores. After computing the resource dependency relations on the AND/XOR graph, we can determine that J_j^l would consume the resource from J_i^k *transitively* if $x^+_{J_i^k,J_j^l} = 1$. Therefore, in this case, J_i^k should inherit the priority from J_j^l after J_j^l starts *i.e.,* after $t_j^1 + (l-1)\times T_j$.

Definition 10 (Extended Inherited-Priority for J_i^k**)** $\forall\, i,j,k,l$

$$EIP_i^k = \max\left\{ IP_i, \max_{j\neq i}\left\{ P_j : (x^+_{J_i^k,J_j^l} = 1) \wedge [t_{current} \geq (t_j^1 + (l-1)\times T_j)]\right\}\right\},$$

where IP_i *is the inherited priority of* J_i^k *through binary semaphores.*

The Definition 10 implies that a job J_i^k inherits a higher priority through a transitive production/consumption relation from J_j^l if and only if J_i^k and J_j^l would overlap. The reason is that if these two jobs do not overlap, then J_i^k's deadline is earlier than the starting time of J_j^l. Thus, in this case (the same as OFUOS), J_i^k will not block J_j^l as long as it does not miss its deadline. However, if they overlap, we call J_i^k an *overlap transitive producer* of J_j^l. The following lemma computes the overlap time period for any two jobs:

Lemma 2 (Overlap between J_i^k and J_j^l) *Given two jobs, J_i^k and J_j^l, the overlap time period betweem them, $OL(J_i^k, J_j^l)$ is equal to*

$$\{T_j - [(t_i^1 + (k-1) \times T_i) - (t_j^1 + (l-1) \times T_j)]^+ - [(t_j^1 + l \times T_j) - (t_i^1 + k \times T_i)]^+\}^+,$$

where

$$(\varepsilon)^+ = \begin{cases} \varepsilon & \mathit{if}\ \varepsilon \geq 0, \\ 0 & \mathit{otherwise} \end{cases}$$

If $OL(J_i^k, J_j^l) = 0$, then J_i^k and J_j^l do not overlap.

6.2.2 Extended Priority Ceiling

In PCP, the *priority ceiling (PC)* attached to one binary semaphore S_p^1 is the highest priority among the set of tasks that would need S_p^1. With priority ceilings, two nice properties about PCP are retained: *deadlock free* and *one longest block only*. After introducing general unbounded semaphores, with extended priority inheritance (Definition 10) and priority ceilings, the first property is still true but the second one no longer holds. (Please note that a high priority consumer waiting for a low priority producer is not considered as blocking because there is no way to avoid this waiting time.)

Example 5 *Five tasks J_a, J_b, J_c, J_d, and J_e with $P_a > P_b > P_c > P_d > P_e$. Assume J_b and J_d share a binary semaphore S_p^1, while J_c and J_e share another binary semaphore S_q^1. Also, J_a is a consumer of one unit of r_1 and one unit of r_2, where J_d is the producer of r_1 and J_e is the producer of r_2.*

By the definition of priority ceiling in [SRL90], $PC(S_p^1) = P_b$ and $PC(S_q^1) = P_c$ because J_b and J_c are the highest priority jobs that would access S_p^1 and S_q^1 respectively. Now, J_c starts at time 0 and gets S_q^1, and at time 2, J_b gets S_p^1 and preempts J_c because $P_b > P_c = PC(S_q^1)$. Then, J_a, J_d, and J_e start at time 4, and J_d preempts J_b because, by Definition 10, $EPI(J_d) = J_a$. However, at time 5, J_d is blocked by J_b because of S_p^1. Then, J_b resumes and finishes, and J_d finishes at time 7. Now, J_e resumes and at time 8 it is blocked by J_c because of S_q^1. Finally, after both J_c and J_e finish, J_a finishes at time 13.

We can see that J_a could possibly be blocked by two lower priority jobs that are not its producers. In the worst case, the number of blockings is proportional to the number of low priority transitive producers a job has. The time diagram is shown in figure 13.

In order to reduce the blocking time due to lower priority jobs, we define an *extended priority ceiling (EPC)* for all the binary semaphores. The extended priority ceiling of S_p^1, for example, would consider not only all the tasks that would share S_p^1 but also all the tasks with lower priority transitive producers that would share S_p^1. Also, we only consider those transitive producers that would overlap with the high priority consumer. In Example 5, the EPC of S_p^1 is P_a not P_b because J_a's producer J_d would share S_p^1. Similarly, the EPC of S_q^1 is P_a, but not P_c. Now, with EPC, only J_c can indirectly block J_a because, at time 2 when J_b tries to get S_p^1, J_b can not be granted ($EPC(S_q^1) = P_a > P_b$).

Definition 11 (Extended Priority Ceiling for S_p^1) $\forall\, p$,

$$EPC(S_p^1) = max\,\{PC(S_p^1), \max_j\,\{P_j\ :\ \exists i, k, l, (x^+_{J_i^k, J_j^l} = 1) \wedge (OL(J_i^k, J_j^l) \neq 0) \wedge (S_p^1 \in \tau_i)\}\},$$

where $PC(S_p^1)$ is the priority ceiling of S_p^1, and $(S_p^1 \in \tau_i)$ means that τ_i needs S_p^1.

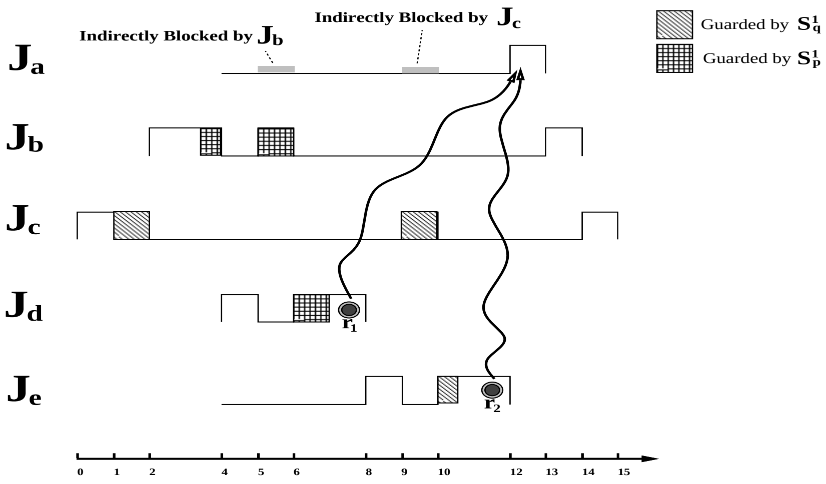

Figure 13: Sequence of Events in the example

6.2.3 The Properties of PCP/OWCF

As we mentioned before, two nice properties about PCP are *deadlock free* and *one longest block only*, *i.e.,* Theorem 10 and 12 in [SRL90]. We now show these two properties still hold for PCP/OWCF.

Lemma 3 (Transitive Blocking) *PCP/ OWCF prevents transitive blocking.*

proof:

Suppose that transitive blocking is possible. Let J_a block J_b and let J_b block J_c, and $P_a < P_b < P_c$.

J_a **blocks** J_b **first:** In this case, by Definition 11, $EPC(S_p^1) \geq P_b$. Thus, later J_b can not lock anything unless J_a releases S_p^1. We reach a contradiction.

J_b **blocks** J_c **first:** It is clear that J_a can not lock anything later because J_b has locked S_q^1 and $EPC(S_q^1) \geq P_c$. We also reach a contradiction. □

Theorem 2 (Deadlock Free) *PCP/ OWCF is deadlock free.*

proof:

We consider three cases: deadlocks among general unbounded semaphores, among binary semaphores, and between general and binary semaphores.

General unbounded semaphores: Since the OWCF policy forces the producer to start earlier than the consumer, the resource dependency relations always go forward in the time diagram. Therefore, it is impossible to form a deadlock;

Binary semaphores: First, we assume a job can not deadlock with itself. By Lemma 3, the number of jobs in a blocking cycle must be two, and we call them J_a and J_b. In order to make a deadlock, both J_a and J_b need at least two binary semaphores, S_p^1 and S_q^1, respectively. This implies both the priority ceilings are equal to or higher than P_a ($P_a > P_b$). Thus, after one of the binary semaphore, *e.g.* S_p^1, is locked, it is impossible for J_b to lock S_q^1.

Binary and general unbounded semaphores: The consumer locks S_p^1 first and the producer needs S_p^1 later to provide the resource. However, this is impossible with OWCF, which forces the producer to start earlier. Thus, the producer inherits the consumer's priority, and the consumer can not run until the producer finishes. □

Theorem 3 (Blocking in PCP/OWCF) *If a job J_i^k has m transitive producers that would overlap with J_i^k, J_i^k can be blocked for the production time of those m resources plus at most the duration of one critical section in any other jobs with priorities lower that P_i.*

proof:

It is obvious that J_i^k has to wait until the m units of resources are produced. The only thing left to prove is that J_i^k and its overlap transitive producers would together be blocked at most once by other lower priority jobs.

Assume that J_i^k would be blocked by both J_a and J_b through S_p^1 and S_q^1, respectively.

Case I: Assume that both S_p^1 and S_q^1 are needed by either J_i^k or one producer. By Definition 11, both $EPC(S_p^1)$ and $EPC(S_q^1)$ is greater than or equal to P_i. If J_a locks S_p^1 earlier than J_b locks S_q^1, then J_b can not lock S_q^1 because $P_b < P_i \leq EPC(S_p^1)$. Similarly, if J_b goes first, J_a can not lock S_p^1. Thus, we have a contradiction.

Case II: Assume that S_p^1 is needed by one transitive producer J_p of $J_i^{(k+\epsilon)}$, where $\epsilon \neq 0$. In other words, J_p produces one unit of resource for another job occurrence of τ_i transitively, but not J_i^k. Now, if S_p^1 is held by J_a, then J_b can not get S_q^1. Furthermore, even after J_a releases S_p^1, J_p can not get S_p^1 to block J_i^k because J_p can not inherit P_i at that moment by Definition 10. This is also a contradiction. □

6.2.4 Schedulability for PCP/OWCF

In this section, we construct two theorems for the schedulability for PCP/OWCF. These two theorems and the scheduability theorems in [SRL90, LSD89] are analogous. However, there is a difference in defining B_i, the duration of the worst case blocking time of a job occurrence in task τ_i. In PCP, B_i only concerns the binary semaphores needed by τ_i and all the tasks with priority higher than P_i. With PCP/OWCF, it is necessary to consider also the binary semaphores needed by all the producers that would have *inherited* priority higher than P_i. Thus, we define the binary semaphores needed by τ_i as all the binary semaphores in τ_i as well as all the overlap transtive producers.

Definition 12 (B_i with PCP/OWCF) $\forall i, B_i = \max_{\forall j, P_j \geq P_i}\{\max_{\forall k, P_j > P_k}\{TimeInterval(\beta_{jk}^*)\}\}$, *where β_{jk}^* is the longest critical section where τ_k can block either τ_j or one of its overlap transitive producers.*

Another concern in PCP/OWCF is the time spent by a consumer task waiting on its producers [5]. In OWCF, we assume that the starting time of a producer must be earlier than the starting time of its consumer. If the producer and the consumer do overlap, then, in the worst case, we have to schedule both of them after the starting time of the consumer. (Please note if they do not overlap, then there is no waiting time caused by this producer.) Thus, we can define the *producer blocking time* for a consumer as the sum of the $WCET$'s (worst case execution time) for all the producers that have lower priorities and do overlap with the consumer.

[5] In PCP/OFUOS, a task would never have to wait for the completion of its producers.

Definition 13 (Producer Blocking Time for τ_i, PT_i) $\forall i$,

$$PT_i = \max_{1 \le k \le \left(\frac{T_{hyper}}{T_i}\right)} \left\{ \sum_j \left[WCET_j \times 1(P_j < P_i) \times \sum_{l=1}^{\left(\frac{T_{hyper}}{T_j}\right)} (x^{+}_{J^l_j, J^k_i} \times 1(OL(J^l_j, J^k_i) \neq 0)) \right] \right\},$$

where

$$1(\varepsilon) = \begin{cases} 1 & \text{if } \varepsilon \text{ is } TRUE, \\ 0 & \text{otherwise} \end{cases}$$

Theorem 4 (A Sufficient Condition for Schedulability with PCP/OWCF) *Using PCP/ OWCF, a set of n periodic tasks can be scheduled by the rate monotonic algorithm [LSD89] if the following conditions are satisfied:* $\forall i, 1 \le i \le n$,

$$\frac{(WCET_1 + PT_1)}{T_1} + \frac{(WCET_2 + PT_2)}{T_2} + \ldots + \frac{(WCET_i + PT_i)}{T_i} + \frac{B_i}{T_i} \le i(2^{\frac{1}{i}} - 1),$$

where PT_i is the production time of resources for τ_i, and B_i is the worst case blocking time for τ_i.

The proof is similar to Theorem 16 in [SRL90] except we treat $WCET^*_i = (WCET_i + PT_i + B_i)$ (they treated $WCET^*_i = (WCET_i + B_i)$ and their definition of B_i is different from ours as indicated in Definition 12.)

Theorem 5 (Exact Characterization for Schedulability with PCP/OWCF) *Using PCP/ OWCF, a set of n periodic tasks can be scheduled by the rate monotonic algorithm if the following conditions are satisfied:* $\forall i, 1 \le i \le n$,

$$\min_{(k,l) \in R_i} \left[\sum_{j=1}^{i-1} U_j \frac{T_j}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil + \frac{(WCET_i + PT_i)}{lT_k} + \frac{B_i}{lT_k} \right] \le 1,$$

where $R_i = \left\{ (k,l) | 1 \le k \le i, l = 1, \ldots, \left\lfloor \frac{T_i}{T_k} \right\rfloor \right\}$, and U_j is the utilization of τ_j.

The proof is similar to Theorem 18 in [SRL90].

6.2.5 Initial Resources

Similarly, the number of initial resources for a set of tasks is computed by the following lemma:

Lemma 4 (Initial Resources, I^r_{OWCF}) *The initial number of type r resources,*

$$I^r_{OWCF} = \sum_{\forall J^k_i \in V} \{ \sum_{\forall J^l_j \in V} [x_{J^k_i, J^l_j} \times \mathcal{N}_{J^k_i, J^l_j} \times \pi^r(\tau_i, \tau_j)] \},$$

where V is the AND/ XOR graph constructed by Algorithm 1, $x_{J^k_i, J^l_j}$'s are the results from the MCF setting under OWCF, and $\mathcal{N}_{J^k_i, J^l_j}$'s are defined in OWCF.

The proof is similar to Lemma 1 except the value of $\mathcal{N}_{J^k_i, J^l_j}$ is from OWCF, not OFUOS.

6.3 OFUOS versus OWCF

In general, OFUOS achieves higher process utilization factor, while OWCF provides *fresher* resources. It is possible to define other kinds of fresh resource allocation policies by defining the linear cost function in our MCF setting differently.

Here, we consider a special case: for any given production/consumption relation, the producer has a higher priority than the consumer. This is the case in real-time monitoring where the low-level sensors have very short periods. And, if we use *rate monotonic scheduling*, then the shorter cycle time means higher priority.

Under this special case, we find out that PCP/OFUOS and PCP/OWCF have the same process utilization factor under rate-monotonic scheduling, while PCP/OWCF still gets fresher resources. This implies that PCP/OWCF is definitely better in both aspects and should be preferred first if the system fits this case.

Theorem 6 (Higher Rate Producer Theorem) $\forall i, j, k, l$, *if* $(x_{J_i^k, J_j^l} = 1) \wedge (P_i > P_j)$, *then PCP/ OWCF and PCP/ OFUOS are the same in processor utilization, but PCP/ OWCF would achieve better resource freshness.*

proof:

If $x_{J_i^k, J_j^l} = 1$ and $P_i > P_j$, then by Definition 10, $\forall j, EIP_j^l = IP_j^l$, also by Definition 11, $EPC(S_p^1) = PC(S_p^1), \forall S_p^1$. This further implies two things:

- the definition of our B_i becomes the same as defined in [SRL90];
- $\forall i, PT_i = 0$.

If we substitute these two items into Theorems 4 and 5, we get exactly the same results as in Theorem 16 and 18 in [SRL90]. Thus, OWCF and OFUOS achieve the same processor utilization factor, but, by their definitions, the resources allocated with OWCF are fresher. □

In general, the schedulability of PCP/OWCF is better if the ratio between high-priority and low-priority producers is higher. This relation is not true for PCP/OFUOS.

7 Related Work

The real-time producers/consumers problem, but without the concern of resource freshness, was pointed out as an open problem in Rajkumar's thesis [Raj91]. The same problem with *deterministic* production/consumption relations has been developed in various forms *e.g.,* [JP91, MAC87]. Here, the word "deterministic" means that the employee has to eat the donuts from the same shop(s) EVERYDAY. We would like to point out that if the production/consumption relations are deterministic, then the *resource freshness* can be purely controlled by the production and consumption rates of periodic tasks. For example, if we ask Dunkin donuts to produce donuts more frequently, then all Dunkin donut's *periodic and static* customers would get *fresher donuts*. However, in the nondeterministic version of this problem, the employee has the freedom to choose from different donut shops to optimize the donut's freshness.

In [MAC87], the consumers and producers are represented in an acyclic dataflow graph model. Since the dataflow graph is deterministic, it is transformed into a process model, which can be used to perform schedulability analysis. The goal of [MAC87] as well as [JP91] is to avoid *data loss*, *i.e.,* a customer has the responsibility to eat the previous donut before the next donut is baked. Therefore, the consumption rate has to be fast enough to match the production rate, and the freshness of each consumed donut is bounded by the producer's cycle length. However, in our problem, because we allow many donuts in the shelf simultaneously, the freshness can not be simply bounded by the production rate.

Our idea of constructing an AND/XOR graph is partially from [Orl82, Bur86, GL90]. Assuming that every task has the same cycle time, *i.e.,* $T_1 = T_2 = \ldots = T_n = K$, Orlin [Orl82] models the resource minimization problem as a *minimum cardinality chain-covers problem* in a *periodic partial ordered set (periodic POSET)*. Burkard [Bur86] analyzed the *optimal phasing* to optimize *maximal waiting time*, *minimal safety interval*, and *average waiting time* simultaneously with two periodic tasks with different cycle time. Unfortunately, no uniform results exist for when the number of tasks with different cycle length are greater than two. In [GL90], the AND/OR precedence constraint scheduling problem is discussed with the goal to minimize the completion time of all the jobs. In this paper, we use the AND/XOR graph instead of the AND/OR graph in order to get a feasible MCF setting.

8 Remarks

In this paper, we define and study a real-time producers/consumers problem, where the production/consumption relations are nondeterministic. Under a sufficient condition, *resource balance*, we map this problem into a *feasible* minimum cost flow problem on an AND/XOR graph. We further develop two *fresh resource allocation* policies by simply defining different linear cost functions. The first policy, OFUOS, assumes the scheduler is vicious, and it plays a non-cooperative game with the scheduler to optimize the freshness for all the consumed resources. The strong point about OFUOS is that it can work with any scheduler without sacrificing the schedulability. On this other hand, the second policy, OWCF, assumes that the scheduler would cooperatively schedule a set of deterministic production/consumption relations. Potentially, with OWCF, the consumed resources are fresher. But, the weak point is that we have to compromise the schedulability.

Both OFUOS and OWCF can work together with the priority ceiling protocol (with rate-monotonic scheduling) to schedule a set of tasks with both binary semaphores and general unbounded semaphores. For OFUOS, we need to do nothing but calculate the amount of initial resources. With OWCF, the definitions of *priority inheritance* and *priority ceiling* are extended. Under this extension, we prove that PCP/OWCF is deadlock free and a job can only be blocked for at most the duration of one longest critical section. Furthermore, schedulability analysis for PCP/OWCF is done. (For PCP/OFUOS, the schedulability analysis is identical to the one for the PCP in [SRL90].)

Finally, in high-speed network monitoring applications, we find that the sensor's (information producer) cycle time is usually shorter than the manager's (information consumer). With rate-monotic scheduling, this implies that producers have higher priority than consumers. Under this special case, we showed that PCP/OWCF and PCP/OFUOS are the same in schedulability but PCP/OWCF is better in freshness. This theorem strongly supports PCP/OWCF in the domain of real-time network monitoring.

Acknowledgments

Dr. Ragunathan Rajkumar had provided many valuable suggestions.

References

[Bak91] T.P. Baker. Stack-based scheduling of realtime processes. *The Journal of Real-Time Systems*, 3(1):67–99, March 1991.

[BJS90] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, 1990.

[Bur86] Rainer E. Burkard. Optimal schedules for periodically recurring events. *Discrete Applied Mathematics*, 15:167–180, 1986.

[CL90] Min-In Chen and Kwei-Jay Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. *The Journal of Real-Time Systems*, 2(4):325–346, December 1990.

[CL91] Min-Ih Chen and Kwei-Jay Lin. A priority ceiling protocol for multiple-instance resources. In *IEEE Real-Time Systems Symposium*, pages 140–149, San Antonio, TX, December 1991.

[GL90] Donald W. Gillies and Jane W.-S. Liu. Scheduling tasks with and/or precedence constraints. In *2nd IEEE Symposium on Parallel and Distributed Processing*, pages 394–401, Dallas TX, December 1990.

[JP91] Kevin Jeffay and Dan Poirier. The real-time producer/consumer paradigm: a paradigm for the construction of efficient, predictable real-time systems. Technical report, CS Department, U. of North Carolina at Chapel Hill, September 1991.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[LSD89] John P. Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica CA, December 1989.

[MAC87] Aloysius K. Mok, Prasanna Amerasinghe, and Moyer Chen. Synthesis of a real-time message processing system with data-driven timing constraints. In *IEEE Real-Time Systems Symposium*, pages 133–143, San Jose CA, December 1987.

[Orl82] James B. Orlin. Minimizing the number of vehicles to meet a fixed periodic schedule: An application of periodic posets. *Operations Research*, 30(4):760–776, July-August 1982.

[Raj91] Ragunathan Rajkumar. *Task Synchronization in Real-Time Systems*. Kluwer Academic Publishers, 1991.

[Roc84] R.T. Rockafellar. *Network Flows and Monotropic Optimization*. John Wiley & Sons, 1984.

[SRL90] 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, September 1990.

[WK90] Shyhtsun F. Wu and Gail E. Kaiser. Network management with consistently managed objects. In *IEEE Global Telecommunications Conference, Communications: Connecting the Future*, volume 1, pages 304.7.1–304.7.6, San Diego CA, December 1990.