Available online at www.sciencedirect.com
SCIENCE DIRECT
Automatica 42 (2006) 245–250
automatica
www.elsevier.com/locate/automatica

Stability analysis of network-based cooperative resource allocation strategies☆

Alvaro E. Gil, Kevin M. Passino*,[1]

Department of Electrical and Computer Engineering, The Ohio State University, 2015 Neil Ave., Columbus, OH 43210, USA

Received 9 February 2005; received in revised form 22 April 2005; accepted 14 September 2005
Available online 1 December 2005

Abstract

Resource allocation involves deciding how to divide a resource of limited availability among multiple demands in a way that optimizes current objectives. In this brief paper we focus on one type of distributed resource allocation problem where via an imperfect communication network multiple processors can share the load presented by multiple task types. We introduce asynchronous "cooperative" resource allocation strategies, and show that they lead to a bounded cumulative demand.

Keywords: Resource allocation; Stability; Cooperative control; Scheduling

1. Introduction

Resource allocation has played an important role in the solution of a large variety of engineering problems. Consider the scenario where we have a group of M processors, not connected over a communication network, and all process just one task type out of N task types at a time. If the group of processors does not coordinate their choices (we name such strategies "noncooperative" which can be thought of as a special type of cooperation that does not rely on direct communications), then for a variety of scheduling strategies they will choose the *same* task type to process all the time, and suppose this is possible (e.g., two machines processing parts from one buffer). Here, we focus on the problem where there is a network of processors, each of which can process different task types. To study the key challenge in this case, we focus on a total "overlap" in processing responsibilities and provide guidelines to handle the impact of other demands on the processors when partial overlapping responsibilities are present in the system. In this "cooperative scheduling problem" there are local schedulers for each of the $M > 1$ processors and a communication network that allows them to share the load presented by the $N > M$ buffers. Here, two or more processors are not allocated to service the same buffer at the same time so that the set of processors is always simultaneously processing task types from M buffers. Notice that this approach could limit the benefits of cooperation in some applications since if one buffer has a very high arrival rate then using more than one processor for it could be justified. We will further assume that each processor can only process one task type at a time and that to switch processing to another task type it incurs a random but bounded delay. The motivation of studying the above cooperative problem instead of the noncooperative one lies in the performance achieved by the former approach depending on the system load. For instance, for the lightly loaded case, noncooperative strategies process tasks from buffers faster than cooperative strategies which forces the set of processors to switch between buffers more frequently in the noncooperative case than in the cooperative one. In this case the cooperative strategy better distributes processing resources to minimize wasteful delays due to setup times. On the other

☆ This paper was not presented at any IFAC meeting. This paper was recommended for publication in revised form by Associate Editor Ioannis Paschalidis under the direction of Editor Ian Petersen.

* Corresponding author. Tel.: +1 614 292 5716; fax: +1 614 292 7596.
E-mail address: passino@ece.osu.edu (K.M. Passino).

[1] This work was supported by the AFRL/VA and AFOSR Collaborative Center of Control Science (Grant F33615-01-2-3154) and the DARPA/ANTS program.

0005-1098/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.automatica.2005.09.015

hand, for the highly loaded case, such benefits are not found as often in the cooperative case since all the processors are busy and do not switch as much. Hence, it is so important to carry out stability analysis of cooperative problems. We will consider the case where the information needed by the processors in order to make decisions about which task type to process next is either held by one processor all the time, or passed along a communication network to any processor. In this study we use a deterministic setting; it is an important research direction to consider cooperative resource allocation problems of the type we consider here in a stochastic setting (see Andradóttir, Ayhan, & Down, 2003).

The work on resource allocation and scheduling in the area of flexible manufacturing systems (FMS) that is most closely related to ours was done by Perkins and Kumar (1989). In some respects our work can be viewed as an extension of the results by Perkins and Kumar (1989). For instance, (i) we focus on M processors processing tasks from M of N buffers at a time, (ii) the class of policies that we introduce is a generalization of the ones defined by Perkins and Kumar (1989) but with the peculiarity that the decisions are made over a range of M times rather that one time, (iii) the processors in this case are heterogeneous, (iv) they operate asynchronously over a network with random but bounded delays, (v) cooperative or noncooperative strategies can be implemented in real-time based on the system load and (vi) Remark 5 considers processors processing tasks from a subset of the total available buffers in a single station. These distinctions are amplified by the fundamentally different set of ideas that are needed to prove our main result. Moreover, our overall philosophical approach is similar to the one of Perkins and Kumar (1989) and hence differs from a stochastic scheduling viewpoint of Gershwin (1994).

Other resource allocation methods include new product development process as studied by Wang, Perkins, and Khurana (2002), "pull" production control methods of Seidman and Holloway (2002), and constrained optimization problems of Ibaraki and Katoh (1988). Moreover, application of the approaches in this paper has been studied by Quijano, Gil, and Passino (2005), Gil, Passino, Ganapathy, and Sparks (2003).

2. Asynchronous cooperative resource allocation

Suppose that the number of task types N is fixed, that we number them, and denote the set of task types as $P = \{1, 2, \ldots, N\}$. Let $p_i, i \in P$ denote the arrival rate of task type i to its respective buffer i so $p_i t$ is the number of tasks of type $i \in P$ that have arrived by time $t \geqslant 0$. Let $x_i(t), i \in P, t \geqslant 0$ denote the size of the buffer level holding task type i and assume that there are sensors that provide the value of these levels whenever a processor requests it. There is a time delay that represents the amount of time that it takes for the processor to switch from processing task type i to another task type j, $j \neq i$. We will call this type of delay $\delta_{i,j} > 0$ and assume it is a random but bounded delay with bound $\overline{\delta}$.

We assume that the number of processors is constant, we number them, and denote the set of processors as $Q = \{1, 2, \ldots, M\}$. Let $1/a_{i_j}$ represent the "rate" at which processor $j \in Q$ processes task type $i \in P$. It can be shown that when processor j starts processing tasks from the ith buffer, the buffer level decreases at a rate $(1 - a_{i_j} p_i)/a_{i_j}$. Hence, it is clear that a necessary condition for stability is that $0 < a_{i_j} p_i < 1$, for all $i \in P$, $j \in Q$. Moreover, this necessary condition indicates how fast processing of buffer levels can occur depending on the set of values of a_{i_j}, p_i. The term $a_{i_j} p_i$ represents the load on processor j due to task type i. We assume in this paper that $N > M$.

We define the set $U(t) \subset P$ as the set of "unattended" task types not processed by any processor at the current time t while the set $U_j^a(t) = \{i_j^*(t)\} \cup U(t)$ is the set of task types that can be considered for processing by processor $j \in Q$. Here, $i_j^*(t)$ is the task type being processed by processor j at time t. Notice that processor j just needs to sample the buffer levels contained in the set $U(t)$ at time t whenever processor j is ready to make a new decision. Define $A(t)$ as the set of task types processed by the group of M processors at the current time t; hence $P = U(t) \cup A(t), t \geqslant 0$.

The "capacity condition" (Perkins & Kumar, 1989) for our case is

$$\rho = \sum_{i=1}^{N} \overline{a}_i p_i < M, \tag{1}$$

where $\overline{a}_i = \max_j \{a_{i_j}\}$. This capacity condition is the sum of all the individual processor's capacity conditions as defined by Lu and Kumar (1991). It is not possible to prove that all resource allocation strategies are stable given this new capacity condition. However, our goal is to obtain the least restrictive conditions by trying to get ρ as close as possible to M for a few strategies.

Now, we will explain how asynchronous decision making is accomplished in this system. Define a processor $j^u \in Q$ that holds the set $U(t)$. We assume that whenever a processor $\ell \in Q$ where $\ell \neq j^u$ (if $\ell = j^u$ there is no need for a request) finishes processing a task type at time t^f such that $x_{i_\ell^*}(t^f) = 0$ (we consider "clearing policies" as by Perkins & Kumar, 1989, but in some situations it would be better not to clear the buffers as by Kumar & Seidman, 1990), it broadcasts a request for the set $U(t)$ to all the processors. Let the amount of time it takes to broadcast the request and receive $U(t)$ be $\delta_c \geqslant 0$ which is random, but bounded by a constant $\overline{\delta}_c > 0$. In the time interval $[t^f, t^f + \delta_c]$ that processor ℓ waits for the unattended set it continues to process task type i_ℓ^* so $x_{i_\ell^*}(t') = 0, t' \in [t^f, t^f + \delta_c]$. The instant that processor ℓ gets $U(t)$ (and the "request queue" defined below), it becomes processor j^u, samples the buffer levels contained in the set $U(t)$, puts task type i_ℓ^* on $U(t)$, decides which task type to process next, and takes it off $U(t)$. So, at time $t^f + \delta_c$ the unattended set is again available for another processor to request.

Since two or more processors could request the set $U(t)$ at the same time, we use a mutual exclusion algorithm which coordinates the access of all processors to the set $U(t)$ in such a way that this set can be accessed and updated by only one processor at a time. Whenever processor j requests $U(t)$,

it will receive both $U(t)$ and the request queue. The request queue contains the number of processors that are waiting for $U(t)$. Thus, any processor that receives both $U(t)$ and the request queue updates the set $U(t)$ and passes this $U(t)$ along with the queue to the new processor at the head of the queue, and this process is repeated until the queue becomes empty.

Note that we have described the case where both the set $U(t)$ and the request queue are passed along the network and they are held by the processor that requested this information; it is clear for this case that a "tracking" mechanism is needed to know the current processor that holds this information, unless broadcast type requests are made as we assume here. However, another scenario can be studied as well, where processor j^u always holds both the set $U(t)$ and the request queue, and whenever a processor $\ell \in Q$, $\ell \neq j^u$, requests the set $U(t)$ held by the processor j^u, it modifies it with the new unattended task types, and sends it back to the processor j^u. Regardless of the strategy used to share $U(t)$, here the key point will be that it is shared over an asynchronous network with random but bounded delays.

Let $k^j, k^j \in \{0, 1, 2, \ldots\}$, denote the index of the sequence of times that processor j makes allocation decisions, $j \in Q$. Let D_{k^j} be the time when processor $j \in Q$, decides to process task type $i_j^*(k^j)$, and assume that at the initial time $D_{k^j}=0$ for $k^j=0$. Let D_{k^j+1} be the next decision time for processor j which is when it completes the processing of task type $i_j^*(k^j)$ and gets the unattended set. For each j define $D_{k^{j^c}}$ to be the closest decision time made by any other processor j^c, previous to the decision time D_{k^j+1} (so given j we can define j^c at each D_{k^j+1}). Note that if no other processor except j makes a decision between times D_{k^j} and D_{k^j+1} then $D_{k^{j^c}}$ is just equal to D_{k^j} so $j^c = j$. Since $\delta_{i,j} > 0$ and $\delta_c \geqslant 0$ we know that $D_{k^j+1} > D_{k^{j^c}}$. By the definition of $j^c \neq j$, $D_{k^j} \leqslant D_{k^{j^c}} < D_{k^j+1}$, and we know that

$$D_{k^j+1} - D_{k^{j^c}} \leqslant \frac{\overline{\delta} + a_{i_{j^c}^*(k^{j^c})} x_{i_{j^c}^*(k^{j^c})}(D_{k^{j^c}})}{1 - a_{i_{j^c}^*(k^{j^c})} p_{i_{j^c}^*(k^{j^c})}} + \overline{\delta}_c. \tag{2}$$

It is important to highlight that for this case $p_{i_{j^c}} = p_i$, for all $j^c \in Q$; this notation is meant to emphasize that buffer level i^* is being chosen by processor j^c. Notice that if $x_{i_{j^c}^*}(t) < \infty$, for all $t \geqslant 0$ and $0 < a_{i_j} p_i < 1$, for all $i \in P$, $j \in Q$ then the buffer level decreasing rate $(1-a_{i_j} p_i)/a_{i_j}$ will cause the buffer $x_{i_{j^c}^*}$ to be cleared within a finite time and thus two consecutive decisions for processor j^c will occur within a finite time, with bounds given by Eq. (2). Furthermore, note that in the time interval $t \in [D_{k^{j^c}}, D_{k^j+1}]$ the set $U(t)$ is constant. This will be useful in our proof below.

3. Cooperative resource allocation strategies

The strategy "Process M buffer levels greater or equal to the average one" is a generalization of the one for the $M = 1$ case by Perkins and Kumar (1989). At time k^j the resource allocation strategy on processor $j \in Q$, chooses to process task type $i_j^*(k^j)$ such that

$$x_{i_j^*(k^j)}(D_{k^j}) \geqslant \frac{\sum_{i_j \in U_j^a(D_{k^j})} x_{i_j}(D_{k^j})}{N - M} \quad \forall i_j \in U_j^a(D_{k^j}). \tag{3}$$

Processor j then processes it until it finishes the tasks in the buffer (which will only take a finite amount of time) at which time it sends a request for $U(t)$ keeping $x_{i_j^*}(t')=0$ for $D_{k^j+1} - \delta_c \leqslant t' \leqslant D_{k^j+1}$ until it receives $U(t)$. Note that when a processor j finishes processing a task type $i_j^*(k^j - 1)$ it chooses a new task type $i_j^*(k^j)$ from the buffer level that is greater than or equal to the average of the buffer levels contained in the set $U_j^a(D_{k^j})$ and, then replaces it with $i_j^*(k^j - 1)$ to form $U(D_{k^j})$. Ties are broken arbitrarily. Generally for $j \neq j'$, $D_{k^j} \neq D_{k^{j'}}$ and $U_j^a(D_{k^j}) \neq U_{j'}^a(D_{k^{j'}})$ so Eq. (3) represents how decisions are made over a range of M times, not just one time as it is in the $M = 1$ case. Since there can be many more decisions made by one processor than another it could be that $D_{k^j} - D_{k^{j'}} \to \infty$ as $k^j \to \infty$ and $k^{j'} \to \infty$, $j \neq j'$.Note that although the processors could complete processing of their respective task types at the same time, their *decisions* will occur at different times since the processors will make choices depending on the queue held by processor j^u so that they will pick different buffers to process due to the use of the mutual exclusion algorithm.

Other strategies can also be defined. For instance, one strategy that "process M task types with the largest buffer levels", also a generalization of one from Perkins and Kumar (1989), can be used in this framework. Similarly, the strategy "process M task types expected to be most difficult to process", chooses the buffers expected to take the longest time to process.

4. Stability analysis

In this section we analyze the stability of the implementation of the strategies defined in Eq. (3). The same proof strategy holds if the other strategies in Section 3 are used.

Theorem. *Assume* $0 < a_{i_j} p_i < 1$, *for all* i, j *and*

$$\sum_{i=1}^{N} \overline{a}_i p_i < M(1 + \underline{ap} - \overline{ap}), \tag{4}$$

where $\underline{ap} = \min_{i,j}\{a_{i_j} p_i\}$, *and* $\overline{ap} = \max_{i,j}\{a_{i_j} p_i\}$.

(i) *For the cooperative resource allocation strategies in Eq.* (3) *a specific bound on the ultimate buffer level for all* $i \in P$ *is given by*

$$\lim_{t\to\infty} x_i(t) \leqslant \frac{\overline{\delta}}{\underline{a}}\left(\overline{ap} + \frac{1}{M}\sum_{i=1}^{N} \overline{a}_i p_i - \underline{ap}\right) + (N - M)\frac{\overline{a}}{\underline{a}} \times \max_{i_j}\left\{\frac{(\overline{\delta} + \overline{\delta}_c(1 - a_{i_j} p_i))(\sum_{i=1}^{N} a_{i_j} p_i - M\underline{ap})}{a_{i_j}(1 - a_{i_j} p_i - (1/M)\sum_{i=1}^{N} a_{i_j} p_i + \underline{ap})}\right\}, \tag{5}$$

where $\underline{a} = \min_{i,j}\{a_{i_j}\}$, *and* $\overline{a} = \max_{i,j}\{a_{i_j}\}$.

(ii) *If the initial condition of the buffer levels satisfies*

$$\sum_{i=1}^{N} x_i(0) \leqslant (N-M) \max_{i_j} \left\{ (\overline{\delta} + \overline{\delta}_c(1 - a_{i_j} p_i)) \times \frac{(\sum_{i=1}^{N} a_{i_j} p_i - M\underline{ap})}{a_{i_j}(1 - a_{i_j} p_i - (1/M)\sum_{i=1}^{N} a_{i_j} p_i + \underline{ap})} \right\} \quad (6)$$

then the inequality in (5) *is satisfied for all* $i \in P, t \geqslant 0$.

Proof. Let

$$V(t) = \sum_{j=1}^{M} V_j(t) = \sum_{j=1}^{M} \left(a_{i_j^*} x_{i_j^*}(t) + \sum_{i \in U(t)} \frac{a_{i_j} x_i(t)}{M} \right). \quad (7)$$

The reader familiar with the first proof of Perkins and Kumar (1989) will notice some similarities to this proof since it is the special $M = 1$ case of this one. There are, however, significant fundamental differences since we have M heterogenous processors operating simultaneously, asynchronously, and over a network with delays.

Define the function $V_j(t)$ for processor j as

$$V_j(t) = a_{i_j^*} x_{i_j^*}(t) + \sum_{i \in U(t)} \frac{a_{i_j} x_i(t)}{M}. \quad (8)$$

The function $V_j(t)$ represents the amount of work at time t that processor j needs to do in order to process the task type that it is currently processing, plus the task types that are unattended. Notice that the unattended set $U(t)$ is being "artificially shared" among all processors and hence the term in Eq. (8) is divided by M.

Consider the sum of the values of the $V_j(t)$ at the *set* of decision times D_{k^j+1},

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) = \sum_{j=1}^{M} \left[a_{i_j^*(k^j)} x_{i_j^*(k^j)}(D_{k^j+1}) + \sum_{i \in U(D_{k^j+1})} \frac{a_{i_j} x_i(D_{k^j+1})}{M} \right]. \quad (9)$$

There is an important difference between Eqs. (7) and (9). While Eq. (7) is evaluated at a specific time t, Eq. (9) is evaluated at the M processor decision times D_{k^j+1}. Since these M decision times occur at different times, it is clear that there is in general a time misalignment among all processor's decisions. Hence, the left-hand side of Eq. (9) is *not* $V(t)$. Below, we will not use $V(t)$ as a Lyapunov-like function in our proof. Instead we use the Eq. (9) in this manner.

Next, since $x_{i_j^*(k^j)}(D_{k^j+1}) = 0$, for all $j \in Q$ ($i_j^*(k^j)$ was the task type that was just processed by processor $j \in Q$), and that by the definition of j^c, $U(D_{k^j+1}) = U(D_{k^{j^c}})$ for t such that $D_{k^{j^c}} \leqslant t \leqslant D_{k^j+1}$ then

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) \leqslant \sum_{j=1}^{M} \{V_{j^c}(D_{k^{j^c}}) - \alpha(i_{j^c}^*(k^{j^c})) \times x_{i_{j^c}^*(k^{j^c})}(D_{k^{j^c}}) + \beta(i_{j^c}^*(k^{j^c}))\}, \quad (10)$$

where

$$\alpha(i_{j^c}) = a_{i_{j^c}} \left(\frac{1 - a_{i_{j^c}} p_{i_{j^c}} - \sum_{i \in U(D_{k^{j^c}})} a_{i_j} p_i / M}{1 - a_{i_{j^c}} p_{i_{j^c}}} \right),$$

$$\beta(i_{j^c}) = \frac{(\overline{\delta} + \overline{\delta}_c(1 - a_{i_{j^c}} p_{i_{j^c}}))\sum_{i \in U(D_{k^{j^c}})} a_{i_j} p_i / M}{1 - a_{i_{j^c}} p_{i_{j^c}}}.$$

It is clear that $\beta(i_{j^c}) > 0$, and is easy to show that $\alpha(i_{j^c})$ is also greater than zero. Moreover, we use the definition of *either* resource allocation strategy for $j = j^c$ to obtain

$$\alpha(i_{j^c}^*(k^{j^c})) x_{i_{j^c}^*(k^{j^c})}(D_{k^{j^c}}) \geqslant \frac{\alpha(i_{j^c}^*(k^{j^c}))}{(N-M)\overline{a}} V_{j^c}(D_{k^{j^c}}).$$

Combine this with Eq. (10) to get

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) \leqslant \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) \left(1 - \frac{\min_{i_{j^c}} \alpha(i_{j^c})}{(N-M)\overline{a}} \right) + M \max_{i_{j^c}} \beta(i_{j^c}) \right\}. \quad (11)$$

This means that we have a contractive mapping in Eq. (11). Notice, however, that in Eq. (11) we have on the left-hand side $V_j(D_{k^j+1})$ and on the right $V_{j^c}(D_{k^{j^c}})$, so the mapping is contractive as we go from k^{j^c} to $k^j + 1$, for all $j \in Q$. The sums in Eq. (11) account for all time so that the contractive mapping is valid for all $t \geqslant 0$.

Define for $k \geqslant 0$ $\overline{V}(k) = \sum_{j=1}^{M} V_{j^c}(D_{k^{j^c}})$ and $\overline{V}(k+1) = \sum_{j=1}^{M} V_j(D_{k^j+1})$. Now, we use $\overline{V}(k)$ and $\overline{V}(k+1)$ in Eq. (11)

$$\overline{V}(k+1) \leqslant \gamma \overline{V}(k) + \zeta, \quad (12)$$

where $\gamma = (1 - \min_{i_{j^c}} \alpha(i_{j^c})/(N-M)\overline{a})$ and $\zeta = M \max_{i_{j^c}} \beta(i_{j^c})$ which are both constants. But, since $0 < \alpha(i_{j^c}) < 1$ for all $i \in P$, $j^c \in Q$ then $0 < \gamma < 1$. Eq. (12) is a difference inequality with a solution that is bounded for all k by $\overline{V}(k) \leqslant (\overline{V}(0) - \zeta/(1-\gamma))\gamma^k + \zeta/(1-\gamma)$. Notice that if $\overline{V}(0) > (\zeta/(1-\gamma))$ ($\overline{V}(0) < \zeta/(1-\gamma)$) then since $\gamma^k \to 0$ as $k \to \infty$, $\overline{V}(k)$ decreases (increases) to $\zeta/(1-\gamma)$ as $k \to \infty$. Now,

$$\frac{\zeta}{1-\gamma} = M(N-M)\overline{a} \max_{i_{j^c}} \frac{\beta(i_{j^c})}{\alpha(i_{j^c})}, \quad (13)$$

which gives us a bound on the transient and ultimate $\overline{V}(k)$ values at the decision times as $k \to \infty$. It can be shown that for all t, $D_{k^{j^c}} + \overline{\delta} \leqslant t \leqslant D_{k^j+1}$ the inequality $V_{j^c}(t) \leqslant V_{j^c}(D_{k^{j^c}} + \overline{\delta})$ holds. On the other hand, using Eq. (8) an upper bound for

$$V_{j^c}(D_{k^{j^c}} + \overline{\delta}) \leqslant V_{j^c}(D_{k^{j^c}}) + \overline{\delta} \left(\overline{ap} + \sum_{i=1}^{N} \frac{\overline{a}_i p_i}{M} - \underline{ap} \right)$$

is obtained. Next, note that for any j^c

$$x_{i^*_{j^c}}(t) + \sum_{i \in U(t)} \frac{x_{i_j}(t)}{M} \leqslant \frac{V_{j^c}(D_{k^{j^c}})}{\underline{a}} + \frac{\overline{\delta}}{\underline{a}} \left(\overline{ap} + \sum_{i=1}^{N} \frac{\overline{a}_i p_i}{M} - \underline{ap} \right)$$

for all t, $D_{k^{j^c}} \leqslant t \leqslant D_{k^j+1}$. Hence, using Eq. (13)

$$\lim_{t \to \infty} x_{i^*_{j^c}}(t) \leqslant B, \tag{14}$$

where B is the right-hand side of Eq. (5).

Now, we must show that each buffer will get chosen by some processor $j \in Q$ infinitely often so that every buffer becomes $i^*_{j^c}$ persistently so that Eq. (14) provides a bound for each buffer level $i \in P$. If that is the case, then we can replace the variable $x_{i^*_{j^c}}(t)$ in the left-hand side of Eq. (14) by $x_i(t)$. Note that we have a bound for every j^c for every buffer level so $x_{i^*_{j^c}}(t)$ is bounded, and using Eq. (2), $D_{k^j+1} - D_{k^{j^c}}$ is bounded. This results in a bound on the time that the unattended set will not be changed. Ignored buffers rise, so eventually any ignored buffer in $U(t)$ will be taken off $U(t)$ and hence become $i^*_{j^c}$.

For the condition stated in (ii), we know by definition that $\overline{V}(0) = \sum_{i=1}^{N} a_i x_i(0) \leqslant \overline{a} \sum_{i=1}^{N} x_i(0)$. Now, if Eq. (6) holds, then following the above procedure we finally obtain that for all $i \in P$, $t \geqslant 0$ $x_i(t)$ is bounded by the right-hand side of Eq. (5). □

Remark 1. The upper bound obtained is in term of system parameters. For instance, when the time delays, $\overline{\delta}$ and $\overline{\delta}_c$, increase, the bound also increases (e.g., network delays can result in ignoring buffers longer). If the number of tasks and processors are about the same, then the ultimate bound decreases.

Remark 2. If $\overline{ap} = \underline{ap}$ in Eq. (4), then $\sum_{i=1}^{N} \overline{a}_i p_i < M$, which indicates that the multiprocessor system could work at its maximum capacity. Otherwise, the capacity of the system is decreased. If the loads are not all equal then when a processor is processing a task type with a high load it will take a considerable amount of time to complete processing that buffer which slows down the overall processing rate thereby reducing processing capacity of the system.

Remark 3. If we assume that $0 < a_{i_j} p_i < M/N$, for all $i \in P$, $j \in Q$ and that Eq. (1) holds, then the same ultimate bound on the buffer level for all $i \in P$ is given by (5) when the cooperative resource allocation strategy in Eq. (3) is used.

Remark 4. The above theorem can be applied to the concatenation or arbitrary interconnections of several cooperative scheduling systems described here. We can derive a similar result to Eq. (5) by using the statements of Theorem 5 and Lemma 6 introduced by Perkins and Kumar (1989) and slightly modifying Eq. (2) to guarantee the stability of the system.

Remark 5. We can relax the necessary condition $0 < a_{i_j} p_i < 1$ in the above theorem by assuming that it only holds for some $j \in Q$, for all $i \in P$. If $a_{i_{j'}} p_i \geqslant 1$, $j' \neq j$ holds for some processor j', processor j' should never engage in processing tasks from buffer i. The proof presented above can be changed to derive an ultimate bound for partial or no overlapping responsibilities of processors for some buffers.

References

Andradóttir, S., Ayhan, H., & Down, D. G. (2003). Dynamic server allocation for queueing networks with flexible servers. *Operations Research*, *51*(6), 952–968.

Gershwin, S. B. (1994). *Manufacturing system engineering*. Englewood Cliffs, NJ: Prentice-Hall.

Gil, A. E., Passino, K. M., Ganapathy, S., & Sparks, A. (2003). Cooperative scheduling of tasks for networked uninhabited autonomous vehicles. In *Proceedings of the IEEE CDC* (pp. 522–527). Maui, HI.

Ibaraki, T., & Katoh, N. (1988). *Resource allocation problems: Algorithmic approaches*. Cambridge, MA: MIT Press.

Kumar, P. R., & Seidman, T. J. (1990). Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, *35*(3), 289–298.

Lu, S. H., & Kumar, P. R. (1991). Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control*, *36*(12), 1406–1416.

Perkins, J. R., & Kumar, P. R. (1989). Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems. *IEEE Transactions on Automatic Control*, *34*(2), 139–148.

Quijano, N., Gil, A. E., & Passino, K. M. (2005). Experiments for decentralized and networked dynamic resource allocation, scheduling, and control. *IEEE Control and Systems Magazine*, *25*(1), 63–79.

Seidman, T., & Holloway, L. (2002). Stability of pull production control methods for systems with significant setups. *IEEE Transactions on Automatic Control*, *47*(10), 1637–1647.

Wang, Y., Perkins, J., & Khurana, A. (2002). Optimal resource allocation in new product development projects: A control-theoretic approach. *IEEE Transactions on Automatic Control*, *47*(8), 1267–1276.

Alvaro E. Gil received his B.S. and M.S. degrees in electrical engineering from Instituto Universitario Politécnico, Barquisimeto, Venezuela, in 1990 and 1998, respectively, and the Ph.D. degree in electrical engineering from The Ohio State University, Columbus, in 2003. From 1990 to 1999, he held engineering positions in the Automation Department at Petróleos de Venezuela (PDVSA) in Maracaibo, Venezuela. From 2002 to 2003 he was a research associate at the Department of Electrical Engineering, The Ohio State University. He worked as a postdoctoral researcher at Ohio State University between 2003 and 2005. He is now with Xerox Corporation, Webster, NY. His current research interests include networked cooperative resource allocation strategies, cooperative scheduling, distributed mobile sensor networks, and multiobjective control.

Kevin M. Passino (S'79, M'90, SM'96, Fellow 2004) received his Ph.D. in Electrical Engineering from the University of Notre Dame in 1989. He is currently a Professor of Electrical and Computer Engineering at The Ohio State University and Director of the OSU Collaborative Center of Control Science that is funded by AFOSR and AFRL/VA. He has served as the Vice President of Technical Activities of the IEEE Control Systems Society (CSS); was an elected member of the IEEE Control Systems Society Board of Governors; was the Program Chair of the 2001 IEEE Conference on Decision and Control; and is currently a Distinguished Lecturer for the IEEE Control Systems Society. He is co-editor (with P.J. Antsaklis) of the book "An Introduction to Intelligent and Autonomous Control", Kluwer Academic

Press, 1993; co-author (with S. Yurkovich) of the book "Fuzzy Control", Addison Wesley Longman Pub., 1998; co-author (with K.L. Burgess) of the book "Stability Analysis of Discrete Event Systems", John Wiley and Sons, 1998; co-author (with V. Gazi, M.L. Moore, W. Shackleford, F. Proctor, and J.S. Albus) of the book "The RCS Handbook: Tools for Real Time Control Systems Software Development", John Wiley and Sons, NY, 2001; co-author (with J.T. Spooner, M. Maggiore, R. Ordonez) of the book "Stable Adaptive Control and Estimation for Nonlinear Systems: Neural and Fuzzy Approximator Techniques", John Wiley and Sons, NY, 2002; and author of "Biomimicry for Optimization, Control, and Automation", Springer-Verlag, London, UK, 2005. For more information, see: http://www.eleceng.ohio-state.edu/∼passino/.