STABILITY ANALYSIS OF NETWORK-BASED COOPERATIVE RESOURCE ALLOCATION STRATEGIES

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Alvaro E. Gil, B.S.E.E., M.S.

* * * * *

The Ohio State University

2003

Ph.D. Examination Committee:

Professor Kevin M. Passino, Adviser

Professor Ümit Özgüner

Professor Andrea Serrani

Approved by

____

Adviser

Department of Electrical Engineering

ABSTRACT

Resource allocation involves deciding how to divide a resource of limited availability across multiple demands in a way that optimizes current objectives (e.g., allocating a processor's computing resources to the demand presented by tasks in order to maximize task completion throughput). In “distributed” resource allocation there are multiple resource types each of which can be subdivided, but then each can only be allocated to a subset of the demands (e.g., in a multiprocessor system where each processor can only process certain task types). In this dissertation we focus on three types of resource allocation problems where via an imperfect communication network multiple agents can share the workload presented by multiple task types. First, we define a model for a network of processors processing task types from buffers and show that they lead to the cumulative demand being bounded by a constant. We demonstrate via simulations when they can be superior to one noncooperative strategy. Second, we model a cooperative control problem for a network of uninhabited autonomous vehicles (UAVs) where it is assumed that before the mission starts a set of tasks is given to a set of UAVs, but then after deployment the UAVs must cooperate to decide which UAV should process each task. We introduce cooperative scheduling of tasks for a set of UAVs where the cooperation must occur over a network that has random but bounded delays. We show how to view this as a cooperative scheduling (resource allocation) problem, and how to derive bounds on mission-level performance metrics for cooperative scheduling methods. Simulations will be used to compare the approach to a noncooperative strategy and to provide design guidelines for the cooperative scheduler. Finally, we introduce an inexpensive laboratory testbed for networked cooperative scheduling strategies. We describe the apparatus, highlight the challenges it presents, and we compare the performance of two scheduling strategies that seek to optimize different objectives. This experiment establishes a basis for future research in networked cooperative scheduling strategies.

To God, Elizabeth, Emmanuel, and Génesis

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Kevin M. Passino, for his guidance and support throughout this academic experience. He had tremendously impacted my way of thinking not only as a student but also as a person. I would also like to thank Dr.Ümit Özgüner and Dr. Andrea Serrani for serving in my examination committee.

I am thankful to the Venezuelan public institution, FUNDAYACUCHO, and the Venezuelan Petroleum Incorporation, PDVSA, for their financial support and for giving me opportunity to study in the United States. I would also like to acknowledge the financial support from the National Institute of Standards and Technology , NIST, and AFRL/VA, AFOSR through the Collaborative Center of Control Science, CCCS, during the last stage of my studies.

I would like to thank my wife, kids, parents, brothers, and in-laws for their support and encouragement during my studies.

I would especially like to thank Sriram Ganapathy for his absolute contribution in the Monte Carlo simulations presented in Section 3.4. I am also thankful to Nicanor Quijano for having shared his dSPACE expertise with me.

A number of other people contributed to the construction of the electromechanical experiment. First, we would like to thank William Thalgott and Emmet Evans for helping with some details in the construction of the electromechanical arcade, which was originally constructed by a group of undergraduates in an EE682P design project and later improved by Kristina Guspan and Dawn M. Kin.

Finally, I want to thank my friends Jorge Finke, Veysel Gazi, Yanfei Liu, Yixin Diao, Hazem Nounou, Yang Liu, and Manfredi Maggiore for their fruitful discussions and suggestions, and for creating a nice working environment.

VITA

February 19, 1967	Born - Barquisimeto, Venezuela
December, 1990	B.S. in Electrical Engineering Antonio José de Sucre Experimental Polytechnic National University, Barquisimeto, Venezuela
January 1991 - December 1992	Application Engineer Venezuelan Petroleum Inc. (PDVSA), Maracaibo, Venezuela
January 1993 - December 1993	Field Engineer Venezuelan Petroleum Inc. (PDVSA), Maracaibo, Venezuela
January 1994 - December 1998	Coordinator of Automation Projects Venezuelan Petroleum Inc. (PDVSA), Maracaibo, Venezuela
January 1995 - December 1998	M.S. in Electrical Engineering Antonio José de Sucre Experimental Polytechnic National University, Barquisimeto, Venezuela
January 1996 - August 1999	Technical Advisor for Special Automation Projects Venezuelan Petroleum Inc. (PDVSA), Maracaibo, Venezuela
September 2002-present	Graduate Research Associate The Ohio State University, Columbus, Ohio

PUBLICATIONS

Research Publications

Alvaro E. Gil and Kevin M. Passino, "Stability Analysis of Cooperative Resource Allocation Strategies," *Proceedings of the IEEE Conference on Decision and Control*, December 2003.

Alvaro E. Gil, Kevin M. Passino, Sriram Ganapathy, and Andy Sparks, "Cooperative Scheduling of Tasks for Networked Uninhabited Autonomous Vehicles," *Proceedings of the IEEE Conference on Decision and Control*, December 2003.

FIELDS OF STUDY

Major Field: Electrical Engineering

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Resource allocation has played a fundamental role in the solution of a large variety of engineering problems. We can use resource allocation algorithms to solve optimization problems, so that given a specific constraint on the amount of resources available, the algorithm will allocate them to several activities such that a pre-defined cost function will be optimized. On the other hand, cooperative control involves coordinating the activities of several agents to work together to complete tasks in order to achieve a common goal. The coordination can occur via a communication network and the goal could be to optimize the task completion rate. Cooperative control can be useful in a variety of applications including multiprocessor computing systems, networked flexible manufacturing systems assembling a product, multiple receivers being coordinated to find multiple emitters in an uncertain environment, and uninhabited autonomous vehicles (UAVs) to perform tasks in order to try to minimize the time it takes to complete tasks during the mission. We want to fuse the last two definitions in order to develop and to study network-based cooperative resource allocation strategies for different scenarios. For example, suppose that we have a processor that needs to process multiple task types. Tasks arrive at the processor, are stored in "buffers" and there is a scheduler (controller, resource allocator) on that processor that decides which task it should process next in order to maximize throughput and hence minimize the buffer levels. In this case the resource being allocated is the processing capability of the processor. We can also use distributed computing and control concepts to develop cooperative resource allocation strategies. For example, there may be the need for a "cooperative" scheduler for a multiprocessor system with local schedulers on each processor. In the same way, hierarchical resource allocation can be applied to optimize a pre-defined cost function in the upper level of a system given a constraint on the resources available. The result can be passed to the lower levels of the system. Each lower level will in turn allocate its resources to optimize its own local cost function. On the other hand, we can also use distributed control ideas to coordinate the activities of several agents to work together to complete tasks in order to achieve a common goal. The coordination can occur via a communication network and the goal could be to optimize the task completion rate.

In this dissertation we consider a cooperative controller to allocate a set of processors to a group of buffers in order to maximize throughput. We also study the use of cooperative scheduling strategies for coordinating uninhabited autonomous vehicles (UAVs) to perform tasks in order to try to avoid ignoring high priority tasks for too long and yet minimize UAVs travel time to perform tasks. Moreover, we illustrate the performance of cooperative scheduling strategies in an experiment called the " electromechanical arcade." Generally, we focus on the design and analysis of cooperative resource allocation strategies in the presence of significant uncertainties. In particular, we pay special attention to two types of uncertainties, those encountered in communications and in the environment.

Uncertainty in communication networks arise from having imperfect communications and hence a lack of centralized and perfect (e.g., up-to-date) information. The network may have a communication topology that is not fully connected in the sense that each agent may not be able to directly communicate with every other agent. Also, each "link" may be imperfect in that there may be delays or bandwidth constraints in sending or receiving information, mis-ordering of messages/information, or noise that corrupts the information. It may not be possible to have one central place to keep all information as if this is one special agent there is a problem with fault tolerance of the whole system (if there is a problem with the special agent the whole system may not be able to function). Moreover, it is typically not possible to keep all information at a special location (e.g., teleoperation center) as it may be out of communication range.

Imperfect communications arise from many sources. First, there is no perfect communication network in spite of recent and envisioned advances in communication network technology. Link bandwidth, delays (e.g., random but bounded ones), and noise are significant problems. Therefore, a reasonable approach to avoid confronting the above problems is to avoid the need for passing extraordinary amounts of information when there are hard real-time constraints.

It is important, however, also to recognize that communication imperfections can arise from sources other than the physical network itself. For instance, if "line-of-sight" is needed for the communication technology, then as a group of agents moves there could be occlusions among them. In some contexts, this may be modeled as a communication delay that can be of quite a significant length of time. Also, clearly this affects both the topology and specific messages over an established link. Next, note that in some cases there may be a need to periodically communicate with a central to have a type of "semi-tele-operated" agent.

Uncertainty in the environment arises from having imperfect information about the tasks that need to be processed by the agents or malfunctioning of the agent devices. For instance, once an agent is engaged in pursuing a task, it may need to travel to where the task is; however, if this agent only knows that the task to process is within a specific area (due to imperfect sensor information), then this agent may spend more time trying to search for the task location than what the agent originally estimated. Another clear example related to the environment is the "pop-up" type task. In this case the agent can be ready to process the task but the agent needs to wait for the appearance of the task in order to start the processing of it. Thus, there is a time delay related to these uncertainties. There may be cases where the agents need to have their local clocks synchronized by a global clock. Desynchronization of the local clocks, which would be viewed as a malfunctioning device, could lead the agents to incorrect decisions that would affect the performance of the whole group of agents.

In summary, we see that there can be significant uncertainties in communications and the environment. Cooperation requires shared information, either via a priori information or communication of information gathered during a mission. If that information is not perfect then we naturally expect to achieve lower levels of cooperation and hence performance. Uncertainty creates a type of passing of bad information that typically leads to poor group decision making, and if the information is quite bad it could be worse to try to cooperate than to simply take a noncooperative approach where no communications are required.

Next, we overview the literature that is relevant to our study of resource allocation strategy modeling, analysis, and design.

1.1 Literature Overview

Resource allocation has been studied in the area of flexible manufacturing systems (FMS). In [1], a resource allocation problem is considered in flexible manufacturing systems where several types of scheduling policies are studied. Several part types arrive at various machines. Strategies are designed for the machines to ensure that the cumulative production of each part type is bounded by a constant. In [2], [3] extensions to the work in [4] were given. For instance, a discrete time framework and additional policies (e.g., time based ones as in [4]) are considered. Also, there is work there on trying to explicitly use global information to obtain more efficient scheduling than when only local information is used. Although the multiple-machine cases above consider the situation of serial production where parts can move from one machine to another one, they do not take into account the case where multiple machines can work together to service a set of part types in the sense that we will consider it here (e.g., via two or more agents processing in parallel several task types). In [5], Chase and Ramadge find a new lower bound on performance by the introduction of machine "idling." In [6], Lou, Sethi, and Sorger improve on the isolated machine clear-a-fraction (CAF) buffer bound of [1] by the implementation of a CAF of work policy. In [7], Kumar and Seidman study general issues in stability of FMS and introduce a universally stabilizing supervisory mechanism (USSM) that can stabilize any scheduling policy. In [8], Lu and Kumar prove stability for a class of policies whose aims are to optimize the performance of the FMS with respect to criteria such as the mean and variance of the total delay incurred in processing a part (other related work can be found in [9]). In [10], Humes introduces the use of "regulators" (devices which control the rates of part flows throughout an FMS) to stabilize general nonacyclic FMS, and in [11], Perkins, Humes, and Kumar explain how to generate buffer bounds in a completely regulated FMS (i.e., one that has regulators directly before each machine buffer input). The bounded delay element is essentially the discrete-time version of the bounded delay in [12]. A control-theoretic approach is considered in [13] in order to allocate heterogeneous resources (people) in the new product development process (NPD). Although most of the problems are based on the minimization of buffer levels in the FMS case, NPD focuses on performance measures considered in scheduling theory of job shops and flow shops, such as the minimizations of functions which depend on due dates and completion times. In [14], Seidman and Holloway analyze the stability of pull production control methods for systems with significant setups; they show that by selecting appropriately the buffer parameters, signal Kanban policies and pattern production policies will be stable subject to a capacity constraint and a limit on the burstiness of total demand. In [14], a set of M machines are working in parallel to serve a set of N buffers with a common input stream to a specific queue. This approach is different from the analysis considered in this dissertation (see Chapter 2) from the perspective that first, the policies of pull production systems are based on response to demand whereas in our case, the policies are based on current buffer levels and the production is designed for future demands. Second, the stability analysis in [14] looks for bounds of the queue of backorders or that the queue will eventually be empty and remains in that condition while the stability analysis in this dissertation focuses on ultimate bounds of the buffer levels.

Resource allocation has been used for the detection of the appearance of a set of events. An application of this technique can be found in [15], where attentional strategies are used for simulating a predator/prey environment. The basic idea is to simulate an organism in an environment with multiple predators that is trying to attend to every movement of all predators so that it can defend itself. In addition to that, it is assumed that there are multiple preys in the same environment, and the organism pays attention to them in case it decides to kill and eat one or several. In [15], the author assumed that each predator/prey appear at a certain time period and that there is just one organism in a predator/prey environment. This approach inspired the development in Chapter 3, however, this work in [15] does not consider the case where there is a group of organisms working together to pay attention to its environment. In [16], the receiver and multiple emitter problem is modeled as a sensor resource allocation problem, where several methods are proposed for the design of optimal dynamic scan schedulers. Similarly, another approach is taken in [17] for solving the sensor allocation problem (what is called there "sensor management") in an environment where there is a single receiver and a certain number of emitters that can vary in time.

Resource allocation has also been formulated as an optimization problem [18]

$$\begin{gathered}\text{minimize } f(x_1, x_2, \ldots, x_n)\\ \text{subject to } \textstyle\sum_{i=1}^{n} x_i = N\\ x_i \geq 0, i = 1, 2, \ldots, n\end{gathered}$$

Thus, given one type of resource whose total amount is equal to N, we want to allocate it to n activities so that the cost function $f(\cdot)$, which is typically some measure of cost of the N activities using the resource, becomes as small as possible. In [19], an ordinal optimization approach is used to solve the above resource allocation problem since convergence of such algorithms is fast and there is no need for step size or scaling parameters commonly used in other classes of algorithms. The necessary and sufficient conditions for a globally optimal solution of a deterministic problem, as well as the similar stochastic case where the costs are estimated, are provided and the algorithm converges in probability to the optimal allocation. In [20], it is shown that under certain additional mild conditions the algorithm also converges almost surely. Moreover, the probability of finding the optimal allocation converges exponentially to one if the simulation length increases linearly. Another approach is formulated in [21], where basically the original discrete problem is transformed into a continuous set over which a surrogate optimization problem is solved. The intention of this is simply to take advantage of a stochastic approximation type of algorithm with the ability to obtain sensitivity estimates with respect to discrete decision variables. One application of this methodology can be found in [22], where a surrogate problem approach is used in the lot sizing problem in manufacturing systems. In [23], an algorithm for stochastic discrete resource allocation problems is proposed. Basically, the algorithm combines the nested partitions (NP) method, ordinal optimization, and an efficient simulation control technique. The NP method is promising for solving difficult combinatorial deterministic optimization problems. The NP method combines global search through global sampling of the space and local search where it should be concentrated. Moreover, the convergence rate of the ordinal comparison can be exponential as the simulation length is increased. Finally, an efficient simulation control technique, optimal computing budget allocation (OCBA), is studied. It is a technique that can be used in a simulation-optimization procedure for selecting the most appropriate design. In [24], a new algorithm for optimal allocation of buffers in production lines can be found . This problem considers tandem production lines that comprise serial processing by a group of machines that are prone to failure. The algorithm combines NP with a local search method, the tabu search (TS) method, to optimally allocate a given buffer capacity among a number of machines such that a cost function, throughput, is maximized.

There is another type of resource allocation problem different from the one studied in [18] that has been used in real-time systems. In [25], adaptive control techniques are used in a closed-loop method for adapting on-line the fraction of assigned resource to the task requirements of a system. On the other hand, in [26] it has been proposed to apply adaptive resource management techniques based on feedback control theory to resource allocation problems that arise during run-time adaptation using PID controllers. In [27], a sequential linear programming algorithm is used to optimize a complex oil production system by selecting the best combination of pressures and flow rates of the set formed by the gathering station, compression plants, and high-pressure gas manifolds.

Also, there is a significant amount of current research activity focused on cooperative control of autonomous vehicles. Receding horizon control approaches are studied in [28, 29, 30]. Work focusing on cooperative search and coordinated sequencing of tasks include [31, 32, 33, 30] and the "map-based approaches" in [34]-[41]. Using such approaches, significant mission performance benefits can be realized via cooperation in some situations, most notably when there is not a high level of uncertainty.

Here, we consider throughout this dissertation environments that are highly uncertain and where we have an imperfect communication network so that there are random but bounded delays on messages communicated to coordinate activities. Due to the presence of so much uncertainty it is generally not possible to accurately predict far into the future, and hence generally not useful to employ optimization approaches to develop long sequences of planned coordinated operations either off- or on-line (as studied in many of the above-mentioned UAV studies). It is well-known that the complexity of coordinated sequencing/planning is significant when there are, for instance, many agents and tasks; however, with significant uncertainty arising from the environment and communication network, coordinated sequencing/planning of long sequences of tasks is not useful anyway so in this case the challenge should often not even be confronted. Instead, the challenge is to overcome the effects of uncertainty so that benefits of cooperation can still be realized. Hence, when uncertainty dominates we will not be able to achieve the high level of coordination achieved in many of the above-mentioned studies; we simply seek to achieve *some* benefit from cooperation.

In this dissertation we define the concept of a cooperative resource allocation similar to the one defined in [42], and it refers to the case where there are a group of agents processing a set of tasks of different types. These agents are separated and they are communicating certain messages to each other in the presence of communication delays. In addition to that, each agent is processing its particular task but cooperating at the same time with the rest of agents in order to achieve a specific goal. Similarly, note that the strategies to be defined have some features mentioned in [42] for synchronous cooperative algorithms in the sense that every agent will choose a task type based on a previous agent selection, the processing time for every agent can be independent of the processing time of the other agents, and every agent communicates its task type selection choice at every decision time. Note that these synchronous strategies are generally not efficient for the case when some agents with a fast processing rate need to wait for the processing of agents with slow processing rate in order for the former to choose new task types to process, this is primarily the reason because we devote our attention in this dissertation to asynchronous strategies. The asynchronous cooperative resource allocation problems considered in this document are defined in such a way that every agent does not wait for the complete processing of the task types being processed by the rest of agents present in the system, so that we can potentially reduce the ultimate bound of every buffer level since there is no "idle" time for any agent. Once a agent completely processes a task type at a decision point, it chooses another task type available for processing.

1.2 Dissertation Outline

In this dissertation we consider the problems of mathematically modeling, performing stability analysis, simulating, and implementing experimentally network-based cooperative resource allocation strategies. First, in Chapter 2 we focus on one type of cooperative resource allocation problem where via an imperfect communication network multiple processors can share the workload presented by multiple task types. We introduce asynchronous "cooperative" resource allocation strategies, and show that they lead to the cumulative demand being bounded by a constant. We demonstrate via simulations when they can be superior to one noncooperative strategy.

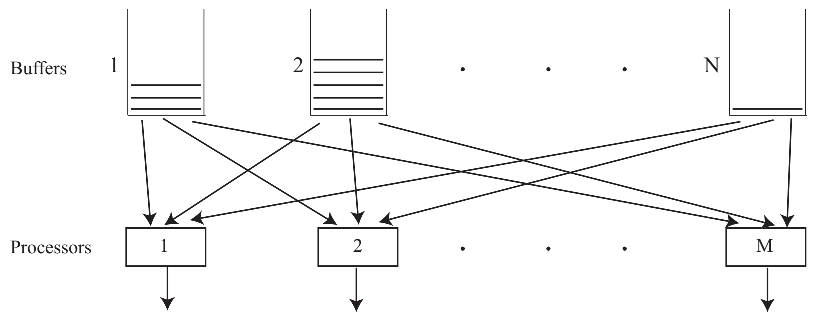
In Chapter 3 we model a cooperative control problem for a network of UAVs where it is assumed that before the mission starts a set of tasks is given to a set of UAVs, but then after deployment the UAVs must cooperate to decide which UAV should process each task. The cooperation must occur in spite of imperfect communications (e.g., messages with random but bounded delays). We show how to view this as a cooperative scheduling (resource allocation) problem, and then derive bounds on mission-level performance metrics for cooperative scheduling methods. Simulations will be used to compare the approach to a noncooperative strategy and to provide design guidelines for the cooperative scheduler.

In Chapter 4 we evaluate and compare the performance of scheduling strategies defined in [15] and in Chapter 3 in the electromechanical arcade experiment. The experiment is composed of eight "pop-up" targets that "appear" and "disappear" at frequencies that are independent of each other and guns that need to detect targets' appearance. We implement algorithms to cooperatively schedule the shooting of the guns in the presence of an imperfect communication network that allows communication between the guns.

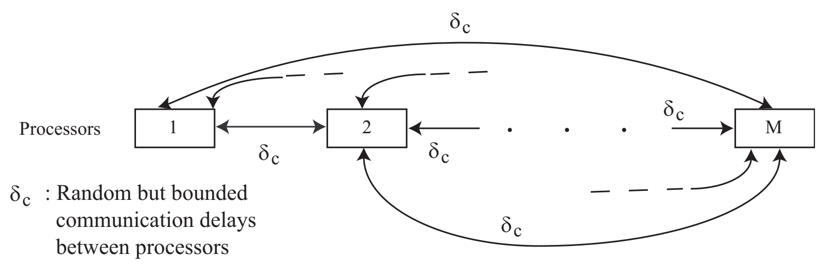
CHAPTER 2

ASYNCHRONOUS COOPERATIVE RESOURCE ALLOCATION

In this chapter we consider the problems of mathematically modeling, performing stability analysis, and simulating network-based cooperative resource allocation strategies. We focus on the case where tasks arrive at buffers, each of which is only accessible to one of many processors. To study the key challenge in this case, we focus on the "overlap" in processing responsibilities and (for now) ignore the impact of other demands on the processors. Hence, we study the problem shown in Figure 2.1(a). 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 workload presented by the $N > M$ buffers that the M processors are responsible for. The goal of the cooperative scheduler is to allocate the M processors to N buffers to maximize throughput. We will 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. We will consider the case where the information needed by the processors in order to make decisions about the task type to process next is either held by one processor all the time or passed along the network to any processor, but where the communication network can have random but bounded delays and



a. Cooperative processing.


δ_c
Processors
1
2
M
δ_c
δ_c
δ_c
δ_c
δ_c : Random but bounded communication delays between processors

b. Processors on a communication network.

Figure 2.1: Cooperative scheduling system.

can be asynchronous (see below for more discussion and Figure 2.1(b)). Notice that there is one resource that must be shared (processor's capabilities), and the resource allocation strategy must decide how it is shared (what task type to process) over an asynchronous network through time. We focus on one type of cooperative resource allocation problem where via an imperfect communication network multiple processors can share the workload presented by multiple task types. We introduce asynchronous "cooperative" resource allocation strategies, and show that they lead to the cumulative demand being bounded by a constant. We demonstrate via simulations when they can be superior to one noncooperative strategy.

2.1 Preliminaries

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. Let t denote time. Let $x_i(t), i \in P, t \geq 0$ 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 $\overline{\delta}$.

Let $1/a_i$ represent the "rate" at which processor processes task type $i \in P$. What is the effect of the a_i and p_i parameters on the buffer level once a processor is processing task types from it? It can be shown that when a processor starts processing tasks from the i^{th} buffer, the buffer level decreases at a rate $\frac{(1-a_i p_i)}{a_i}$. Hence, it is clear that a necessary condition for stability is that $0 < a_i p_i < 1, \forall i \in P$. Moreover, this necessary condition indicates how fast processing of buffer levels can occur depending on the set of values of a_i, p_i. The term $a_i p_i$ represents the load on the processors due to task type i. We assume that the number of processors is constant, we number them, and denote the set of processors as $Q = \{1, 2, \ldots, M\}$ where $N > M$. Note that we are not concerned with the case where $N \leq M$ because if $N = M$ then the number of processors and task types are the same and every processor will be processing tasks from the same buffer all the time. Moreover, if there are more processors than task types ($N < M$) then at least two or more processors will be processing simultaneously tasks from the same buffer.

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 $U_j^a(t) = \left\{ i_j^*(t) \right\} \cup U(t)$ is the set of task types that can be considered for processing by processor $j, j \in Q$. Here, $i_j^*(t)$ is the task type being processed by processor j at time t. Define $A(t)$ as the set of task types processed by the group of M processors at the current time t; hence $P = U(t) \cup A(t), t \geq 0$.

Now, what is the "capacity" [1] to do work for this case? Clearly it is necessary that

$$\rho = \sum_{i=1}^{N} a_i p_i < M \tag{2.1}$$

and this serves as a "capacity condition." How can we interpret Equation (2.1)? First, the capacity condition of our system is the sum of all the individual processor's capacity conditions in [1]. Second, the advantage of having M processors working together is reflected in the capacity condition of each of them since the work to be done in order to process all task types is reduced by the number of processors. Can we prove that all resource allocation strategies are stable given this new capacity condition? No. 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.

2.2 Asynchronous Decision Making

In this section we will explain how asynchronous decision making is accomplished. 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 $x_{i_\ell^*}(t^f) = 0$ (i.e., the instant the buffer is emptied), 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 $\delta_c > 0$ 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, it samples the buffer levels, 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 need to 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. Assume processor j^u has a "request queue." There are certain ways of creating that queue, and one possibility is to use the policy first in-first out (FIFO), and another one is simply to use a predefined order (e.g., requests made from processor 1 up to processor M). Thus, if processor j^u has already built a queue and updated the set $U(t)$, it proceeds to send both $U(t)$ and the request queue to the processor located at the head of the queue when it transmits it. The 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. These types of communications can be viewed as "token-asking algorithms," where a processor cannot update the set $U(t)$ unless it possesses the token. It is not the intention of this dissertation to focus on a detailed definition of distributed mutual exclusion algorithms since these types of algorithms can be found in [43] and elsewhere. We simply need to establish here our strategy for cooperation, which is based on asynchronous communication of the set $U(t)$ along with the request queue.

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 where random but bounded delays can be incurred.

Let $k^j, k^j = 0, 1, \ldots,$ denote the index of the sequence of times that processor j makes allocation decisions. 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 $D_{k^j} = 0$. 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 $\delta_c > 0$ we know that $D_{k^j+1} > D_{k^{j^c}}$. By the definition of j^c, $D_{k^j} \leq D_{k^{j^c}} < D_{k^j+1}$, and we know that

$$D_{k^j+1} - D_{k^{j^c}} \leq D_{k^j+1} - D_{k^j} \leq \frac{\overline{\delta} + a_{i_j^*(k^j)} x_{i_j^*(k^j)}(D_{k^j})}{1 - a_{i_j^*(k^j)} p_{i_j^*(k^j)}} + \overline{\delta}_c \tag{2.2}$$

Also, if $j^c \neq j$

$$D_{k^j+1} - D_{k^{j^c}} < D_{k^{j^c}+1} - D_{k^{j^c}} \leq \frac{\overline{\delta} + a_{i^*_{jc}(k^{j^c})} x_{i^*_{jc}(k^{j^c})}(D_{k^{j^c}})}{1 - a_{i^*_{jc}(k^{j^c})} p_{i^*_{jc}(k^{j^c})}} + \overline{\delta}_c \tag{2.3}$$

This can be seen in Figure 2.2. 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.

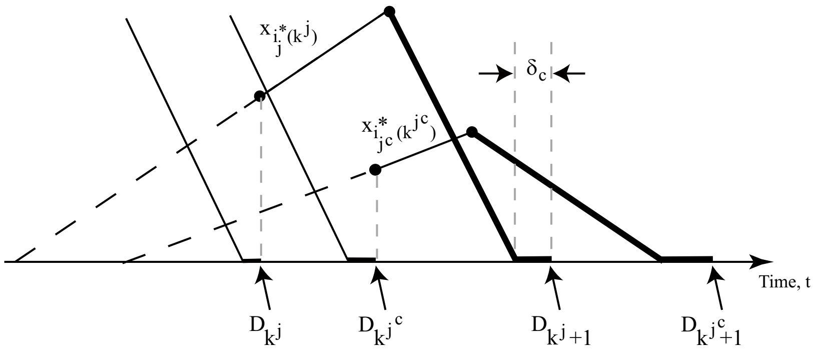


Figure 2.2: Decision times made by $M = 2$ processors.

2.3 Synchronous Decision Making

While in the remainder of the chapter we study asynchronous cooperative resource allocation strategies, we have also studied a type of synchronous cooperative resource allocation problem, which consists in a group of processors trying to make decisions at the same time. In this case, the processors start processing their chosen buffers, and then wait till each other are all done processing before they decide again. Notice that some processors could finish processing their respective task types some time before the last of the M processors finishes as it can be seen in Figure 2.3. Thus, there could be idling where some processors could have brought their buffer levels to zero (and buffer levels remain at that value since arrivals and processing continues but the

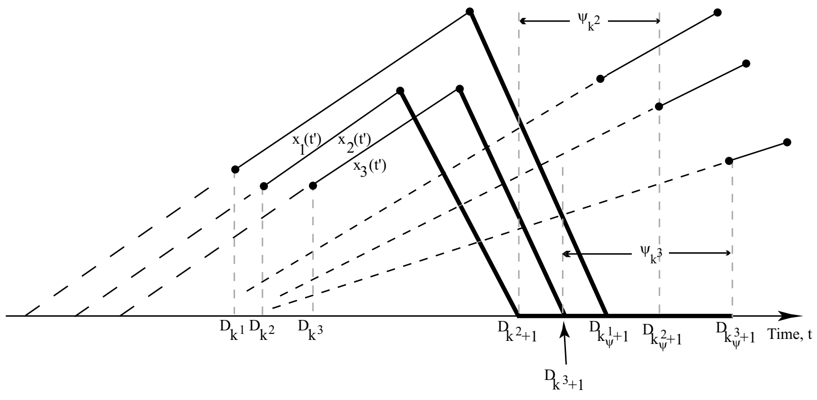


Figure 2.3: Illustration of task type processing by $M = 3$ processors working synchronously (ψ_k^j denotes "idling" by processor j before it makes a decision).

processor immediately takes care of each task), but they cannot make any decision because they need to wait until the rest of the processors are done. The stability analysis of this strategy is not included in this dissertation because we think of it as a special case of the asynchronous one; this can be easily seen by making idling, ψ_k^j, equal to δ_c in Figure 2.3. In fact, we could follow the approach detailed in Section 2.5 to obtain a single stability result for both asynchronous and synchronous cooperative resource allocation strategies. This can be done by considering idling in the analysis, which leads to the result that such strategies are still stable in the presence of such idling. However, it would be interesting to explore whether there exists a condition where idling could produce lower bounds for synchronous strategies rather than for asynchronous ones based on the previous work done in [5] for the $M = 1$ case.

2.4 Cooperative Resource Allocation Strategies

Next, we introduce the resource allocation strategies that we study in the remainder of this chapter.

Processing M Task Types with the Largest Buffer Level

This strategy is a generalization of the one for the $M = 1$ case in [1]. For it, at time k^j the resource allocation strategy on each processor j chooses to process task type $i^*_j(k^j)$, such that

$$x_{i^*_j(k^j)}(D_{k^j}) \geq x_{i_j}(D_{k^j}), \forall i_j \in U^a_j(D_{k^j}) \tag{2.4}$$

and processes it until it finishes the tasks in the buffer (which will only take a finite amount of time) when it sends a request for $U(t)$ and continues processing until it receives $U(t)$ (keeping $x_{i^*_j}(t^{'}) = 0$ for $D_{k^j+1} - \delta_c \leq t^{'} \leq D_{k^j+1}$). 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 largest buffer level contained in the set $U^a_j(D_{k^j})$ and, then replaces it with $i^*_j(k^j - 1)$ to form $U(D_{k^j})$. If there is more than one maximizer for any processor at any time, then the resource allocation strategy will simply choose one of these at random (likewise, for the strategies below). Generally for $j \neq j^{'}$, $D_{k^j} \neq D_{k^{j'}}$ and $U^a_j(D_{k^j}) \neq U^a_{j'}(D_{k^{j'}})$ so Equation (2.4) 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.

Processing M Task Types with Buffer Levels Greater or Equal to the Average One

This strategy is also a generalization of one from [1]. For it, at time k^j the resource allocation strategy on each processor j chooses to process task type $i_j^*(k^j)$ such that

$$x_{i_j^*(k^j)}(D_{k^j}) \geq \frac{1}{N-M} \sum_{i_j \in U_j^a(D_{k^j})} x_{i_j}(D_{k^j}), \forall i_j \in U_j^a(D_{k^j}) \tag{2.5}$$

and process it as from the above policy until it receives $U(t)$ so it can make another decision. Similar additional comments made for the last policy hold here also.

Processing M Task Types Expected to Be Most Difficult to Process

This strategy is taken from [15], which was inspired by the work done in [1]. Assume that at each k^j each processor j chooses to process task type $i_j^*(k^j)$ at time D_{k^j} if

$$i_j^*(k^j) = \arg\max_{i_j} \left\{ w_{i_j} \left(\frac{x_{i_j}(D_{k^j}) + \delta_{i_j,j} p_{i_j}}{\frac{(1-a_{i_j} p_{i_j})}{a_{i_j}}} \right) \right\}, \forall i_j \in U_j^a(D_{k^j}) \tag{2.6}$$

where $w_{i_j} > 0$ are weighting factors. This strategy estimates the length of time that elapses from the time where the peak occurs, until the buffer level currently processed is brought to zero for the first time. The weighting factors w_{i_j} can be chosen to force the processor to process some task types more than others. We have derived a specific bound on the ultimate buffer level for all $i \in P$ when this strategy is used for the $M = 1$ case defined in [15].

Note that all the above strategies have in common that they are all clearing policies. We also studied a pre-emptive approach for the "clear the largest buffer" (CLB) [1] strategy (i.e., $x_{i^*(k)}(D_k) \geq x_{i(k)}(D_k), \forall i \in P$) for the $M = 1$ case. For it, once the processor engages in processing task types from buffer i^*, it will continue processing task types from that buffer until the following condition is satisfied

$$x_{i^*(k)}(D_{k+1}) \leq \epsilon x_{i^*(k)}(D_k)$$

where $0 < \epsilon < 1$. Notice that the condition $\epsilon < 1$ will guarantee that the buffer level i^* at time D_{k+1} be always less than its value at time D_k, while the condition $0 < \epsilon$ will avoid that the buffer levels be brought to zero. We do not show the result obtained for this pre-emptive approach, however, the bound for the pre-emptive case is greater than the one in [1]. From that perspective, the result, in general, indicates that it is beneficial to clear the buffer once a processor starts processing it. Unfortunately, however, since these bounds are conservative we cannot conclude that one approach is superior to the other.

2.5 Stability Analysis

In this section we analyze the stability of the implementation of the strategies defined in Equations (2.4) and (2.5). The same proof strategy holds if Equation (2.6) is used.

Theorem: Assume that $0 < a_i p_i < 1, \forall i \in P$ and

$$\sum_{i=1}^{N} a_i p_i < M(1 + \underline{ap} - \overline{ap}) \tag{2.7}$$

where $\underline{ap} = \min_i\{a_ip_i\}$, and $\overline{ap} = \max_i\{a_ip_i\}$. For the cooperative resource allocation strategies in Equations (2.4) and (2.5) a specific bound on the ultimate buffer level for all $i \in P$ is given by

$$\begin{aligned}\lim_{t\to\infty} x_i(t) &\leq \frac{\overline{\delta}}{\underline{a}}\left(\overline{ap} + \frac{1}{M}\sum_{i=1}^{N} a_ip_i - \underline{ap}\right) \\ &+ (N-M)\frac{\overline{a}}{\underline{a}}\max_i\left\{\frac{(\overline{\delta}+\overline{\delta_c}(1-a_ip_i))(\sum_{i=1}^{N} a_ip_i - M\underline{ap})}{a_i(1-a_ip_i-\frac{1}{M}\sum_{i=1}^{N} a_ip_i + \underline{ap})}\right\}\end{aligned}$$

where $\underline{a} = \min_i\{a_i\}$, and $\overline{a} = \max_i\{a_i\}$.

Proof: Let

$$V(t) = \sum_{i=1}^{N} a_ix_i(t)$$

The proof to follow focuses on the strategy where the task type i_j^* is chosen by processor $j \in Q$, that has the buffer level greater or equal to the average one; however, a special case of this is when the largest buffer level is chosen at each decision point so the above bound holds for that resource allocation strategy also. The reader familiar with the first proof in [1] 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 processors operating simultaneously, asynchronously, and over a network with delays.

Now, we split the $V(t)$ function into two terms: the task types that are "unattended", $U(t)$, and the ones that are being "attended", $A(t)$. Thus,

$$V(t) = \sum_{i=1}^{N} a_ix_i(t) = \sum_{i\in U(t)} a_ix_i(t) + \sum_{i\in A(t)} a_ix_i(t)$$

Since there are M processors processing different task types at the same time, we next define a function for each processor such that when we sum the functions we obtain $V(t)$. Let

$$\sum_{i\in U(t)} a_i x_i(t) = M \sum_{i\in U(t)} \frac{a_i x_i(t)}{M} = \sum_{j=1}^{M} \left(\sum_{i\in U(t)} \frac{a_i x_i(t)}{M} \right)$$

and notice that

$$\sum_{i\in A(t)} a_i x_i(t) = \sum_{j=1}^{M} a_{i_j^*} x_{i_j^*}(t)$$

Hence,

$$V(t) = \sum_{j=1}^{M} V_j(t) = \sum_{j=1}^{M} \left(a_{i_j^*} x_{i_j^*}(t) + \sum_{i\in U(t)} \frac{a_i x_i(t)}{M} \right) \tag{2.8}$$

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

$$V_j(t) = a_{i_j^*} x_{i_j^*}(t) + \sum_{i\in U(t)} \frac{a_i x_i(t)}{M} \tag{2.9}$$

The function $V_j(t)$ represents the amount of work that every processor needs to do in order to process at time t the task type that it is currently processing plus the task types that are unattended. However, notice that the unattended set is being "artificially shared" among all processors since the set $U(t)$ is shared by the number of processors and this is why the second term of Equation (2.9) is divided by M.

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

$$\sum_{j=1}^{M} V_j(D_{k^j}) = \sum_{j=1}^{M} \left[a_{i_j^*(k^j)} x_{i_j^*(k^j)}(D_{k^j}) + \sum_{i\in U(D_{k^j})} \frac{a_i x_i(D_{k^j})}{M} \right] \tag{2.10}$$

There is an important difference between Equations (2.8) and (2.10). While Equation (2.8) is evaluated at a specific time t, Equation (2.10) is evaluated at the M processor decision times D_{k^j}. Since these M decision times can 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 Equation (2.10) is *not* $V(t)$. Below, we will not use $V(t)$ as a Lyapunov-like function in our proof. Instead we use the Equation (2.10) in this manner.

Next, since $x_{i_j^*(k^j)}(D_{k^j+1}) = 0, \forall j \in Q$ ($i_j^*(k^j)$ was the task type that was just processed by processor $j \in Q$),

$$\begin{aligned} \sum_{j=1}^{M} V_j(D_{k^j+1}) &= \sum_{j=1}^{M} \left[a_{i_j^*(k^j)} x_{i_j^*(k^j)}(D_{k^j+1}) + \sum_{i \in U(D_{k^j+1})} \frac{a_i x_i(D_{k^j+1})}{M} \right] \\ &= \sum_{j=1}^{M} \left[\sum_{i \in U(D_{k^j+1})} \frac{a_i x_i(D_{k^j+1})}{M} \right] \end{aligned} \tag{2.11}$$

Note that by the definition of j^c, $U(D_{k^j+1}) = U(D_{k^{j^c}})$ since the unattended set will not change for every processor j for t such that $D_{k^{j^c}} \leq t \leq D_{k^j+1}$. Hence, considering task arrivals in this time period,

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) = \sum_{j=1}^{M} \left\{ \sum_{i \in U(D_{k^{j^c}})} \left[\frac{a_i}{M} x_i(D_{k^{j^c}}) + \frac{a_i p_i}{M} (D_{k^j+1} - D_{k^{j^c}}) \right] \right\} \tag{2.12}$$

Hence, for processor j^c in Equation (2.9) at time $D_{k^{j^c}}$

$$\sum_{i \in U(D_{k^{j^c}})} \frac{a_i}{M} x_i(D_{k^{j^c}}) = V_{j^c}(D_{k^{j^c}}) - a_{i_{j^c}^*(k^{j^c})} x_{i_{j^c}^*(k^{j^c})}(D_{k^{j^c}}) \tag{2.13}$$

We use Equations (2.3), (2.12), and (2.13) to obtain

$$\begin{aligned} \sum_{j=1}^{M} V_j(D_{k^j+1}) \leq\ & \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) - a_{i_{j^c}^*(k^{j^c})} x_{i_{j^c}^*(k^{j^c})}(D_{k^{j^c}}) \right. \\ & + \left. \sum_{i \in U(D_{k^{j^c}})} \frac{a_i p_i}{M} \left(\frac{\overline{\delta} + a_{i_{j^c}^*(k^{j^c})} x_{i_{j^c}^*(k^{j^c})}(D_{k^{j^c}})}{1 - a_{i_{j^c}^*(k^{j^c})} p_{i_{j^c}^*(k^{j^c})}} + \overline{\delta}_c \right) \right\} \\ \leq\ & \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) - a_{i_{j^c}^*(k^{j^c})} x_{i_{j^c}^*(k^{j^c})}(D_{k^{j^c}}) \left(1 - \frac{\sum_{i \in U(D_{k^{j^c}})} \frac{a_i p_i}{M}}{1 - a_{i_{j^c}^*(k^{j^c})} p_{i_{j^c}^*(k^{j^c})}} \right) \right. \\ & + \left. \frac{(\overline{\delta} + \overline{\delta}_c (1 - a_{i_{j^c}^*(k^{j^c})} p_{i_{j^c}^*(k^{j^c})})) \sum_{i \in U(D_{k^{j^c}})} \frac{a_i p_i}{M}}{1 - a_{i_{j^c}^*(k^{j^c})} p_{i_{j^c}^*(k^{j^c})}} \right\} \end{aligned}$$

Define

$$\alpha(i_{j^c}) = a_{i_{j^c}} \left(\frac{1 - a_{i_{j^c}} p_{i_{j^c}} - \sum_{i \in U} \frac{a_i p_i}{M}}{1 - a_{i_{j^c}} p_{i_{j^c}}} \right) \tag{2.14}$$

and

$$\beta(i_{j^c}) = \frac{\left(\overline{\delta} + \overline{\delta}_c(1 - a_{i_{j^c}} p_{i_{j^c}})\right) \sum_{i \in U} \frac{a_i p_i}{M}}{1 - a_{i_{j^c}} p_{i_{j^c}}}$$

So

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) \leq \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) - \alpha(i^*_{j^c}(k^{j^c})) x_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) + \beta(i^*_{j^c}(k^{j^c})) \right\} \tag{2.15}$$

It is clear that $\beta(i_{j^c}) > 0$, and we need to find out whether $\alpha(i_{j^c})$ is also greater than zero. We know that

$$\frac{1}{M} \sum_{i=1}^{N} a_i p_i = \frac{1}{M} \sum_{i \in A} a_i p_i + \frac{1}{M} \sum_{i \in U} a_i p_i < 1 + \underline{ap} - \overline{ap} \leq 1 + \underline{ap} - a_i p_i, \forall i \in P$$

where the second to last inequality comes from the assumption in the statement of the theorem. Thus,

$$-1 - \underline{ap} + \frac{1}{M} \sum_{i \in A} a_i p_i < -\frac{1}{M} \sum_{i \in U} a_i p_i - a_i p_i < 0$$

Therefore,

$$0 = -\underline{ap} + \frac{1}{M} M \underline{ap} < -\underline{ap} + \frac{1}{M} \sum_{i \in A} a_i p_i < 1 - \frac{1}{M} \sum_{i \in U} a_i p_i - a_i p_i < 1 \tag{2.16}$$

Since the terms $a_{i_{j^c}} > 0$, $1 - a_{i_{j^c}} p_{i_{j^c}} > 0$ in Equation (2.14), and by using Equation (2.16) we know that $\alpha(i_{j^c}) > 0$.

We use the definition of *either* resource allocation strategy for $j = j^c$

$$\alpha(i^*_{j^c}(k^{j^c})) x_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) \geq \frac{\alpha(i^*_{j^c}(k^{j^c}))}{(N - M)\overline{a}} \left[\sum_{i \in U^a_j(D_{k^{j^c}})} a_i x_i(D_{k^{j^c}}) \right]$$

But notice that

$$\begin{aligned}
\alpha(i^*_{j^c}(k^{j^c}))x_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) &\geq \frac{\alpha(i^*_{j^c}(k^{j^c}))}{(N-M)\overline{a}}\left[a_{i^*_{j^c}(k^{j^c})}x_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) + \sum_{i \in U(D_{k^{j^c}})} a_i x_i(D_{k^{j^c}})\right] \\
&\geq \frac{\alpha(i^*_{j^c}(k^{j^c}))}{(N-M)\overline{a}}\left[a_{i^*_{j^c}(k^{j^c})}x_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) + \sum_{i \in U(D_{k^{j^c}})} \frac{a_i x_i(D_{k^{j^c}})}{M}\right] \\
&\geq \frac{\alpha(i^*_{j^c}(k^{j^c}))}{(N-M)\overline{a}} V_{j^c}(D_{k^{j^c}})
\end{aligned}$$

Combine this with Equation (2.15) to get

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) \leq \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) \left(1 - \frac{\alpha(i^*_{j^c}(k^{j^c}))}{(N-M)\overline{a}}\right) + \beta(i^*_{j^c}(k^{j^c}))\right\} \tag{2.17}$$

This means that we have a contractive mapping in Equation (2.17). Notice, however, that in Equation (2.17) we have on the left 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 $k^j + 1, \forall j \in Q$. Do the sums in Equation (2.17) account for all time so that the contractive mapping is valid for all $t \geq 0$? The answer is yes, and we explain why next. Recall how we defined j^c: Given a j we have some D_{k^j+1} and from that value we define j^c as the index of the processor that most recently finished its processing at time $t < D_{k^j+1}$. So given any D_{k^j+1} we can always find a j^c (at $t = 0$ we consider all processors to have just finished processing to get $x_{i_j}(0) = 0$). At any time t such that $x_{i_j}(t)$ becomes zero is labeled as D_{k^j+1} as soon as processor j has already received $U(t)$. Then the time range $[D_{k^{j^c}}, D_{k^j+1}]$ is considered in the mapping since the left hand side of Equation (2.17) is revaluated at the right side of this interval for one j and the right side of Equation (2.17) is evaluated at the left side of the interval. Now, the key is to note that we can relabel this j^c (and hence $D_{k^{j^c}}$) as $D_{k^{j'}+1}$ since it was the time that processor j^c finished so it is the new decision time for the processor. Then there is of course a $j^{'c}$ and so on so that all time intervals are considered in the contractive mapping. Note that this accounts for the fact that the sums in the contraction are based on a *set* of times that are not necessarily contiguous. Note also that in Equation (2.17) it is not possible that $V_j(D_{k^j+1}) = V_{j^c}(D_{k^{j^c}})$ for any j since $\delta_{i,j} > 0$ so that $D_{k^j+1} > D_{k^j}$ and $D_{k^{j^c}} < D_{k^j+1}$ by definition. Also it is not possible that for any j, $V_j(D_{k^j+1}) \to V_{j^c}(D_{k^{j^c}})$ as $k^j \to \infty$. Why? Because, while this may happen for some j values, it cannot happen for all such values. Note that if for $M-1$ processors $V_j(D_{k^j+1}) \to V_{j^c}(D_{k^{j^c}})$, then for the remaining processors, say $j^{'}$, $D_{k^{j'^c}} < D_{k^{j'}+1}$, $V_{j'}(D_{k^{j'}+1}) \nrightarrow V_{j'^c}(D_{k^{j'^c}})$.

Next, notice that

$$\begin{aligned}\sum_{j=1}^{M} V_j(D_{k^j+1}) &\leq \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) \left(1 - \frac{\min_{i_{j^c}} \alpha(i_{j^c})}{(N-M)\overline{a}}\right) + \max_{i_{j^c}} \beta(i_{j^c}) \right\} \\ &\leq \left(1 - \frac{\min_i \alpha(i)}{(N-M)\overline{a}}\right) \sum_{j=1}^{M} \{V_{j^c}(D_{k^{j^c}})\} + M \max_i \beta(i) \qquad (2.18)\end{aligned}$$

Define for $k \geq 0$

$$\overline{V}(k) = \sum_{j=1}^{M} V_{j^c}(D_{k^{j^c}})$$

and

$$\overline{V}(k+1) = \sum_{j=1}^{M} V_j(D_{k^j+1})$$

so that $\overline{V}(k+1)$ is the sum at the set of times when all processors $j \in Q$ have already finished processing their respective task types and have received $U(D_{k^{j^c}})$ in order to choose which task types they will process next.

Now, we use $\overline{V}(k)$ and $\overline{V}(k+1)$ in Equation (2.18)

$$\overline{V}(k+1) \leq \gamma \overline{V}(k) + \zeta \qquad (2.19)$$

where $\gamma = \left(1 - \frac{\min_i \alpha(i)}{(N-M)\overline{a}}\right)$ and $\zeta = M \max_i \beta(i)$ which are both constants. But, notice that

$$\left[1 - \frac{\alpha(i^*_{j^c}(k^{j^c}))}{(N-M)\overline{a}}\right] = 1 - \frac{\left\{a_{i_{j^c}} \left(\frac{1 - a_{i_{j^c}} p_{i_{j^c}} - \sum_{i \in U(D_{k^{j^c}})} \frac{a_i p_i}{M}}{1 - a_{i_{j^c}} p_{i_{j^c}}}\right)\right\}}{(N-M)\overline{a}}$$

By using the bounds of Equation (2.16), we know that

$$0 < \frac{1 - a_{i_{j^c}} p_{i_{j^c}} - \sum_{i \in U} \frac{a_i p_i}{M}}{1 - a_{i_{j^c}} p_{i_{j^c}}} < 1, \forall i_j \in P$$

Furthermore,

$$0 < \frac{a_{i_{j^c}}}{(N-M)\overline{a}} < 1$$

hence

$$0 < 1 - \frac{\alpha(i^*_{j^c}(k^{j^c}))}{(N-M)\overline{a}} < 1 \tag{2.20}$$

so that $0 < \gamma < 1$.

Equation (2.19) is a difference inequality with a solution that is bounded for all k by

$$\overline{V}(k) \leq \left(\overline{V}(0) - \frac{\zeta}{1-\gamma}\right)\gamma^k + \frac{\zeta}{1-\gamma} \tag{2.21}$$

Notice that if $\overline{V}(0) > \frac{\zeta}{1-\gamma}$ ($\overline{V}(0) < \frac{\zeta}{1-\gamma}$) then since $\gamma^k \to 0$ as $k \to \infty$ $\overline{V}(k)$ decreases (increases) to $\frac{\zeta}{1-\gamma}$ as $k \to \infty$.

Now

$$\begin{aligned} \frac{\zeta}{1-\gamma} &= \frac{M(N-M)\overline{a}\max_i \beta(i)}{\min_i \alpha(i)} \\ &= M(N-M)\overline{a}\max_i \frac{\beta(i)}{\alpha(i)} \end{aligned} \tag{2.22}$$

This gives us a bound on the transient and ultimate $\overline{V}(k)$ values as $k \to \infty$, at the decision times.

We need to consider the times in between the decision times. To do this, note that for all j^c, k^{j^c}, and k

$$V_{j^c}(D_{k^{j^c}}) \le \overline{V}(k)$$

Next, we show that for $D_{k^{j^c}} \le t \le D_{k^j+1}$

$$V_{j^c}(t) \le V_{j^c}(D_{k^{j^c}} + \overline{\delta}) \tag{2.23}$$

From Equation (2.9) and for any t, $D_{k^{j^c}} \le t \le D_{k^{j^c}} + \overline{\delta}$

$$\begin{aligned}
V_{j^c}(D_{k^{j^c}} + \overline{\delta}) &= a_{i^*_{j^c}(k^{j^c})} x_{i^*_{j^c}(k^{j^c})}(t + D_{k^{j^c}} + \overline{\delta} - t) \\
&+ \sum_{i \in U(t + D_{k^{j^c}} + \overline{\delta} - t)} \frac{a_i x_i(t + D_{k^{j^c}} + \overline{\delta} - t)}{M} \\
&= a_{i^*_{j^c}(k^{j^c})} x_{i^*_{j^c}(k^{j^c})}(t) + (D_{k^{j^c}} + \overline{\delta} - t) a_{i^*_{j^c}(k^{j^c})} p_{i^*_{j^c}(k^{j^c})} + \sum_{i \in U(t)} \frac{a_i x_i(t)}{M} \\
&+ (D_{k^{j^c}} + \overline{\delta} - t) \sum_{i \in U(D_{k^{j^c}} + \overline{\delta} - t)} \frac{a_i p_i}{M} \\
&= V_{j^c}(t) + (D_{k^{j^c}} + \overline{\delta} - t) \left(a_{i^*_{j^c}(k^{j^c})} p_{i^*_{j^c}(k^{j^c})} + \sum_{i \in U(D_{k^{j^c}} + \overline{\delta} - t)} \frac{a_i p_i}{M} \right)
\end{aligned}$$

since the second term of the right hand side of the above equation is nonnegative

$$V_{j^c}(t) \le V_{j^c}(D_{k^{j^c}} + \overline{\delta}) \tag{2.24}$$

for all t, $D_{k^{j^c}} \le t \le D_{k^{j^c}} + \overline{\delta}$.

Next, we consider t, $D_{k^{j^c}} + \overline{\delta} \le t \le D_{k^j+1}$, and use Equation (2.9)

$$\begin{aligned}
V_{j^c}(D_{k^j+1}) &= a_{i^*_{j^c}(k^{j^c})} x_{i^*_{j^c}(k^{j^c})}(t + D_{k^j+1} - t) + \sum_{i \in U(t + D_{k^j+1} - t)} \frac{a_i x_i(t + D_{k^j+1} - t)}{M} \\
&= a_{i^*_{j^c}(k^{j^c})} x_{i^*_{j^c}(k^{j^c})}(t) - (D_{k^j+1} - t) \left(\frac{1 - a_{i^*_{j^c}(k^{j^c})} p_{i^*_{j^c}(k^{j^c})}}{a_{i^*_{j^c}(k^{j^c})}} \right) a_{i^*_{j^c}(k^{j^c})} \\
&+ \sum_{i \in U(t)} \frac{a_i x_i(t)}{M} + (D_{k^j+1} - t) \sum_{i \in U(D_{k^j+1} - t)} \frac{a_i p_i}{M}
\end{aligned}$$

$$= \quad V_{j^c}(t) + (D_{k^j+1} - t)\left(-1 + a_{i^*_{jc}(k^{j^c})} p_{i^*_{jc}(k^{j^c})} + \sum_{i \in U(D_{k^j+1} - t)} \frac{a_i p_i}{M}\right)$$

since the second term of the right hand side of the last equation is nonpositive

$$V_{j^c}(t) \le V_{j^c}(D_{k^{j^c}} + \overline{\delta}) \tag{2.25}$$

for all t, $D_{k^{j^c}} + \overline{\delta} \le t \le D_{k^j+1}$.

On the other hand,

$$\begin{aligned} V_{j^c}(D_{k^{j^c}} + \overline{\delta}) &= a_{i^*_{jc}(k^{j^c})} x_{i^*_{jc}(k^{j^c})}(D_{k^{j^c}}) + \overline{\delta} a_{i^*_{jc}(k^{j^c})} p_{i^*_{jc}(k^{j^c})} + \sum_{i \in U(D_{k^{j^c}})} \frac{a_i x_i(D_{k^{j^c}})}{M} \\ &+ \sum_{i \in U(D_{k^{j^c}})} \frac{a_i p_i \overline{\delta}}{M} \\ &= V_{j^c}(D_{k^{j^c}}) + \overline{\delta}\left(a_{i^*_{jc}(k^{j^c})} p_{i^*_{jc}(k^{j^c})} + \sum_{i \in U(D_{k^{j^c}})} \frac{a_i p_i}{M}\right) \\ &\le V_{j^c}(D_{k^{j^c}}) + \overline{\delta}\left(\overline{ap} + \sum_{i=1}^{N} \frac{a_i p_i}{M} - \underline{ap}\right) \end{aligned} \tag{2.26}$$

where the second term of the right hand side of the equation was derived by using

$$\sum_{i \in U(D_{k^{j^c}})} \frac{a_i p_i}{M} = \sum_{i=1}^{N} \frac{a_i p_i}{M} - \sum_{i \in A(D_{k^{j^c}})} \frac{a_i p_i}{M} \le \sum_{i=1}^{N} \frac{a_i p_i}{M} - M\frac{\underline{ap}}{M} \tag{2.27}$$

Notice here that we can also obtain a tighter bound on the right side of Equation (2.27) by using the M minimum values of $a_i p_i$ rather than $M\underline{ap}$.

Next, note that for any j^c

$$\underline{a}\left(x_{i^*_{jc}}(t) + \sum_{i \in U(t)} \frac{x_i(t)}{M}\right) \le a_{i^*_{jc}} x_{i^*_{jc}}(t) + \sum_{i \in U(t)} \frac{a_i x_i(t)}{M} = V_{j^c}(t)$$

so that

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

Hence,

$$\begin{aligned}
\lim_{t\to\infty} x_{i^*_{jc}}(t) &\leq \lim_{t\to\infty}\left[x_{i^*_{jc}}(t) + \sum_{i\in U(t)} \frac{x_i(t)}{M}\right] \\
&\leq \frac{\overline{\delta}}{\underline{a}}\left(\overline{ap} + \frac{1}{M}\sum_{i=1}^{N} a_i p_i - \underline{ap}\right) + M(N-M)\frac{\overline{a}}{\underline{a}}\max_i \frac{\beta(i)}{\alpha(i)} \\
&\leq \frac{\overline{\delta}}{\underline{a}}\left(\overline{ap} + \frac{1}{M}\sum_{i=1}^{N} a_i p_i - \underline{ap}\right) \\
&+ M(N-M)\frac{\overline{a}}{\underline{a}}\max_i\left\{\frac{(\overline{\delta}+\overline{\delta}_c(1-a_ip_i))(\sum_{i=1}^{N}\frac{a_ip_i}{M} - \underline{ap})}{a_i(1-a_ip_i - \frac{1}{M}\sum_{i=1}^{N} a_ip_i + \underline{ap})}\right\} \\
&\leq \frac{\overline{\delta}}{\underline{a}}\left(\overline{ap} + \frac{1}{M}\sum_{i=1}^{N} a_i p_i - \underline{ap}\right) \\
&+ (N-M)\frac{\overline{a}}{\underline{a}}\max_i\left\{\frac{(\overline{\delta}+\overline{\delta}_c(1-a_ip_i))(\sum_{i=1}^{N} a_ip_i - M\underline{ap})}{a_i(1-a_ip_i - \frac{1}{M}\sum_{i=1}^{N} a_ip_i + \underline{ap})}\right\}
\end{aligned} \tag{2.28}$$

Now, we must show that each buffer will get chosen by some processor $j \in Q$ infinitely often so that every buffer becomes i^*_{jc} persistently so that Equation (2.28) provides a bound for each buffer level $i \in P$. Note that we have a bound for every k^{j^c} for every buffer level so $x_{i^*_{jc}(k^{j^c})}(D_{k^{j^c}})$ is bounded, and using Equation (2.3), $D_{k^j+1} - D_{k^{j^c}}$ 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^*_{jc}. ■

Next, we make a few remarks about the ultimate bound obtained. First of all, note that 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 load values assigned to tasks increase then the ultimate bound does also since then we may spend more time processing the higher task loads and hence an increased amount of time ignoring the lower task loads. However, if the load of all tasks is *very high*, then the delay incurred by the processor communications does not have too much influence on the bound since processors are so busy processing tasks from buffers and they do not communicate to each other so frequently. If the number of tasks and processors are about the same, then the ultimate bound decreases. On the other hand, if $\overline{ap} \neq \underline{ap}$ then in a sense, the capacity of the system is decreased. Why? Note that 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. Therefore, this processor will not be able to process other task types for a while (it is as if it were not helping) and the rest of processors have to process the remaining buffers. Notice that this slows down the overall processing rate and the processing capacity of the system is reduced.

One important point to highlight here is the fact that the analysis could take into account the behavior of heterogeneous processors (i.e., processors with different processing capabilities for the same task). Therefore, we could define a rate at which processor j processes task type i, $\frac{1}{a_{i_j}}, \forall i \in P, \forall j \in Q$, and with a slight modification of the proof in Equation (2.18) the stability analysis would hold for this type of processor.

2.6 Comparative Analysis: Noncooperative Vs. Cooperative Strategies

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 at a time. If the group of processors does not coordinate their choices, 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).

Due to the lack of communications and hence coordinated decision making we name such strategies "noncooperative." In this section we want to make a comparison between these noncooperative strategies and the cooperative strategies discussed in this chapter to clarify when a cooperative strategy might be superior.

2.6.1 Comparison between the ultimate bounds

Here, for our noncooperative strategies we simply use the "clear a fraction" policy from [1] but with M processors so they will all simultaneously process one of N buffers. Here, we start our comparative analysis by comparing the bounds for the cooperative and noncooperative cases, in *one* special case. Suppose that $\frac{1}{a_i^j}$ is the processing rate for task type i by processor j. Note that the ultimate bound obtained by Perkins and Kumar in [1], but for this noncooperative case, is

$$\begin{aligned}\lim_{t\to\infty}\sup x_i(t) \leq \lim_{t\to\infty}\sup \sum_i x_i(t) \ &\leq \ \frac{\overline{\delta}\sum_{i=1}^N a_i^t p_i}{\underline{a^t}} \\ &+ \ \frac{\overline{a^t}(N-1)}{\underline{a^t}}\max_i\left\{\frac{\overline{\delta}\left(\sum_{i=1}^N a_i^t p_i - a_i^t p_i\right)}{a_i^t(1-\sum_{i=1}^N a_i^t p_i)}\right\}\end{aligned} \tag{2.29}$$

where the "total processing rate" $\frac{1}{a_i^t}$ is

$$\frac{1}{a_i^t} = \frac{1}{a_i^1} + \ldots + \frac{1}{a_i^j} + \ldots + \frac{1}{a_i^M}, \forall i \in P$$

If we assume that $a_i^j = a_i^\ell, \forall j, \ell \in Q$, then $a_i^t = \frac{a_i}{M}$. Moreover, assume that $a_i = a_j$ and $p_i = p_j$, for all i and j so that the ultimate bound in Equation (2.29) is

$$\lim_{t\to\infty}\sup x_i(t) \leq \frac{\overline{\delta}\sum_{i=1}^N a_i p_i}{\underline{a}} + \frac{\overline{a}(N-1)}{\underline{a}}\max_i\left\{\frac{\overline{\delta}\left(\sum_{i=1}^N a_i p_i - a_i p_i\right)}{a_i(1-\frac{1}{M}\sum_{i=1}^N a_i p_i)}\right\} \tag{2.30}$$

Next, consider the cooperative case. Since we know that $a_i p_i = a_j p_j$, for all i and j, via Equation (2.28) we obtain an ultimate bound

$$\begin{aligned}\lim_{t\to\infty} x_i(t) &\leq \frac{\overline{\delta}\sum_{i=1}^{N} a_i p_i}{\underline{a}M} \\ &+ \frac{\overline{a}(N-M)}{\underline{a}} \max_i \left\{ \frac{(\overline{\delta}+\overline{\delta}_c(1-a_i p_i))(\sum_{i=1}^{N} a_i p_i - M\underline{ap})}{a_i(1-\frac{1}{M}\sum_{i=1}^{N} a_i p_i)} \right\} \quad (2.31)\end{aligned}$$

where it can be seen that when the arrival of tasks and processing rates for every processor have all the same value, the ultimate bound for the cooperative case, Equation (2.31), is *always* smaller than the ultimate bound for the noncooperative case, Equation (2.30) if $\overline{\delta}_c \to 0$. Unfortunately, however, since these bounds are conservative we cannot, even in this special case, conclude that one strategy is superior to the other.

2.6.2 Simulations

Here, we seek to determine if the cooperative strategies are superior to noncooperative ones. In particular, are cooperative strategies superior to noncooperative ones for all processing and arrival rates (i.e., for all a_i and p_i)? In addition to that, we would like to answer the following questions: Is it possible, and if so in what cases, that the cooperative strategies perform better than the noncooperative ones? Can the noncooperative strategy perform better than the cooperative one? When?

In order to compare both cooperative and noncooperative strategies we must have the conditions that were defined for stability satisfied, i.e., $0 < a_i p_i < 1$, $\sum_{i=1}^{N} a_i p_i < M(1 - \overline{ap} + \underline{ap}) \leq M$, and $\sum_{i=1}^{N} a_i p_i < M$ respectively. Note that if $a_i p_i \neq a_j p_j$, the stability constraint of the cooperative case is always smaller than the noncooperative case; hence all our cases studied below are for cases where the processors are lightly loaded *relative to noncooperative strategies.* In order to answer all the above questions we run a Monte Carlo simulation with the following values: the sampling time is $T_s = 0.0001$ sec., the length of the simulation is 5 sec., the number of simulations to run are 300 (we verified that this is a sufficient number), the values of $0 < a_i p_i < 1$ are randomly generated for every simulation, the values of the arrival rate ($0 < p_i < 10$) are randomly generated also, and for every simulation the equation $\sum_{i=1}^{N} a_i p_i < M(1 - \overline{ap} + \underline{ap})$ is forced to be satisfied (randomly generated cases where they are not are rejected). Furthermore, there are $N = 4$ task types, $M = 2$ processors, the switching times for processors 1 and 2 are $\delta^1_{i,j} = \delta^2_{i,j} = 0.01$ sec., and we do not consider network delays so, $\delta_c \to 0$. The range of workload generated randomly for every processor, $0 < a_i p_i < 0.6$, was relatively large; hence we consider this set of cases to be a test for performance for a highly loaded processor case.

Table 2.1 shows a summary of the performance measures of the Monte Carlo simulation for this case where we want to point out that those values converge after the 50^{th} simulation run. Notice that the averages of the performance measures obtained in the cooperative case are better than the noncooperative one. However, we want to comment that the performance measures of the noncooperative case were *better* than the cooperative one in 37 of the 300 cases. We observe that there is a common factor in those cases, and this is that the stability condition values, $\sum_{i=1}^{N} a_i p_i$, are relatively big as it is shown in Figure 2.4. In other words, the processors are highly loaded and that motivates us to simulate the case where the processors are lightly loaded so that the values of $0 < a_i p_i < 0.2$ are randomly generated for every simulation.

The results obtained are shown in Table 2.2. Note that for this case the averages of the performance measures of the cooperative case are again better than the noncooperative ones. In fact, the cooperative case performed better in 299 out of 300 cases.

	Noncooperative	Cooperative
Average of average values of buffer levels	$\boxed{0.0951}$	$\boxed{0.0775}$
Variance of average values of buffer levels	0.0009	0.0012
Maximum of average values of buffer levels	$\boxed{0.1378}$	$\boxed{0.1345}$
Variance of maximum of average values of buffer levels	0.0016	0.0043

Table 2.1: Average of the performance measures from the Monte Carlo simulations (highly loaded case).

	Noncooperative	Cooperative
Average of average values of buffer levels	$\boxed{0.0928}$	$\boxed{0.0625}$
Variance of average values of buffer levels	0.0009	0.0004
Maximum of average values of buffer levels	$\boxed{0.1340}$	$\boxed{0.1023}$
Variance of maximum of average values of buffer levels	0.0015	0.0009

Table 2.2: Average of the performance measures from the Monte Carlo simulations (lightly loaded case).

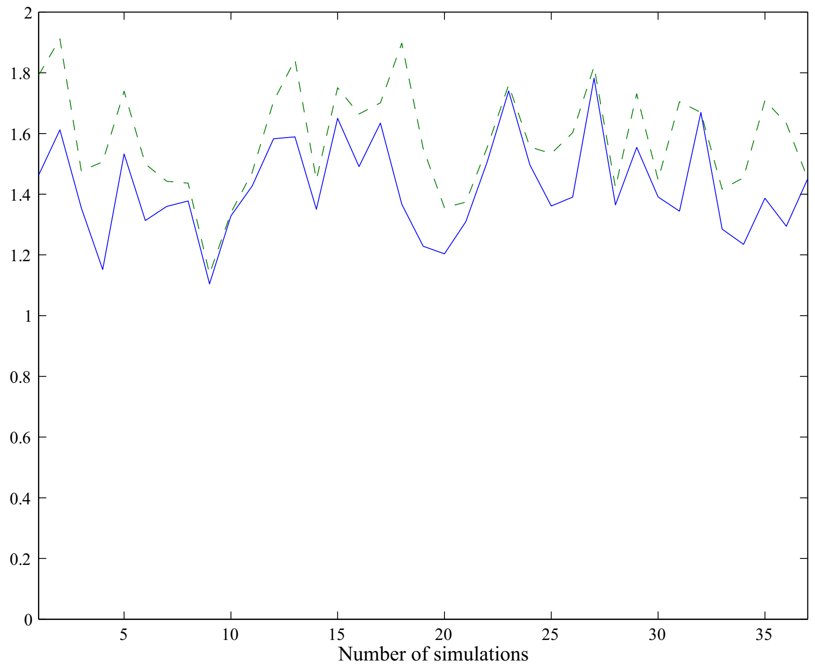


Figure 2.4: Stability condition values for the cooperative case (solid line) and noncooperative one (dashed line).

These simulations confirm that in the lightly loaded case the advantages realized by the noncooperative strategy over the cooperative one can be lost. This is because in this case the cooperative strategy better distributes processing resources to minimize wasteful delays due to set-up times. In the highly loaded case such benefits are not found as often since all the processors are busy.

2.7 Discussion: Ideas for Cooperative Strategy Design

We have the intuition that there may be cases where a *combination* of the cooperative and the noncooperative resource allocation strategies could maximize the throughput and minimize wait-times and it would be interesting to derive stability properties for this scenario. The idea behind this combination of strategies is that every time that a processor needs to choose a new buffer level to process, it needs to compute how long it will take to clear the buffer to be chosen in such a way that it may end up either helping to process a buffer level that is already being processed by another processor, or choosing a new buffer contained in the set $U(t)$. This choice could be based on selecting the longest time to process a buffer no matter whether or not this buffer is already being processed by any other processor. A combined strategy could provide the benefits of the cooperative strategy we study here with potential benefits of simultaneous processing of a buffer (which can indeed be thought of a special type of cooperation that does not rely on direct communications).

CHAPTER 3

COOPERATIVE SCHEDULING OF TASKS FOR NETWORKED UNINHABITED AUTONOMOUS VEHICLES

We model, perform mathematical analysis, and simulate in this chapter a cooperative control problem for a network of UAVs where it is assumed that before the mission starts a set of tasks is given to a set of UAVs, but then after deployment the UAVs must cooperate to decide which UAV should process each task. The cooperation must occur in spite of imperfect communications. We show how to view this as a cooperative scheduling (resource allocation) problem, and how to derive bounds on mission-level performance metrics for cooperative scheduling methods. Simulations will be used to compare the approach to a noncooperative strategy and to provide design guidelines for the cooperative scheduler.

3.1 Problem Statement

Suppose that there are several UAVs that need to process a set of tasks, where to process a task a UAV has to go to a certain location in a region of finite size. Here, we assume that a set of UAVs is given a priori a number of tasks, their respective locations, and characteristics (e.g., the priority of each task). It is assumed that the tasks must be repeatedly visited and processed (e.g., for repeated surveillance of points spread across a large region where the points of interest are more numerous than the number of UAVs). UAVs must work together autonomously to allocate their capabilities to process all these tasks in order to maximize the rate of task processing for the highest priority tasks. In this chapter we introduce the model for the cooperative scheduling of tasks problem; however, there are analogies to the multiprocessor problem that we have already defined in Chapter 2 and this will allow us to shorten our description somewhat.

3.1.1 Tasks, Prioritized Time, and Processing Time

Suppose that the number of tasks is fixed and that we number and denote them as $P = \{1, 2, \ldots, N\}$. Assume that the number of UAVs is constant and we number them and denote the set of UAVs as $Q = \{1, 2, \ldots, M\}$ where $N > M$ (if $N \leq M$, then you could dedicate at least one UAV to each task). Let $p_i, i \in P$ denote the priority of processing task i. Let t denote time. Let $T_i(t), i \in P, t \geq 0$ denote the "prioritized time" since last processing of task i and t_i denote the time since last processing task i (e.g., if task i was last processed at time zero then $T_i(t) = p_i t_i$ is its "prioritized time"). We will assume that there is a cooperative scheduling strategy that decides which task a UAV should process next. Assume that there is a global clock (e.g., via GPS) that keeps all the UAV's clocks synchronized. Let $x^i = [x_1^i, x_2^i, \theta^i]^\top$ denote the coordinates in the (x_1^i, x_2^i) plane and orientation θ^i for the i^{th} task. Let $d(x_v^j(t), x^i)$ be the distance that UAV j must travel from its current location and orientation $x_v^j(t) = [x_{v_1}^j(t), x_{v_2}^j(t), \theta_v^j(t)]^\top$ to process task i at x^i. If communication delays are finite and tasks are in a finite region, then we can find a $\overline{d} \geq d(\cdot)$ for all i, j and $t \geq 0$. Given a certain scenario that contains a fixed number of tasks located in a finite region, we could determine $\overline{d}$ by choosing the longest travel distance that any UAV must travel between two tasks considering their respective orientation angles. Suppose that all UAVs travel at a constant velocity v and either move on a constant minimum turn radius or on a straight line. Let τ_{i_j}, $0 \leq \underline{\tau} \leq \tau_{i_j} \leq \overline{\tau}, \forall i \in P, \forall j \in Q$ denote the "processing" time that takes to process task i by UAV j with $\underline{\tau}$ ($\overline{\tau}$) the minimum (maximum) time it takes to process task i after a UAV first arrives at it. Let $\delta_{i,j}$, $0 \leq \delta_{i,j} \leq \overline{\delta}$ be the random but bounded time delay that represents the amount of time that it takes for any UAV to switch from processing task i to task j, $j \neq i$. Let $\delta_c > 0$ be the random but bounded time delay incurred when UAV j has finished processing any task but it needs to wait for information held by another UAV in order to make a decision (see below for more details). Assume that δ_c is random but bounded by $\overline{\delta}_c > 0$.

Consider the case where there is only one task ($N = 1$), named "task 1" and one ($M = 1$) UAV. Suppose that at some time t', the value of prioritized time since last processing task 1 is $T_1(t') > 0$ as shown in Figure 3.1 and the last task processed by UAV 1 is task i. At time $t' + \delta_{i,1}$ UAV 1 is heading to task 1. Then, at time $t' + \delta_{i,1} + \frac{d(x_v^1, x^1)}{v}$ UAV 1 is at the location where task 1 is and it initiates the processing of task 1, and the amount of time that it takes to do so is dictated by the τ_1 parameter (here processing could, e.g., include multiple passes over an object at different angles in order to detect or identify/classify it). When processing for task 1 is completed the UAV sends a request to the other UAVs to coordinate decision making and thereby experiences the delay δ_c. Finally, at time $t' + \delta_{i,1} + \frac{d(x_v^1, x^1)}{v} + \tau_1 + \delta_c$ UAV 1 chooses the next task to perform. Moreover, we assume that $T_1(t)$ could have in Figure 3.1 any shape for the time when UAV 1 is processing task 1 to represent

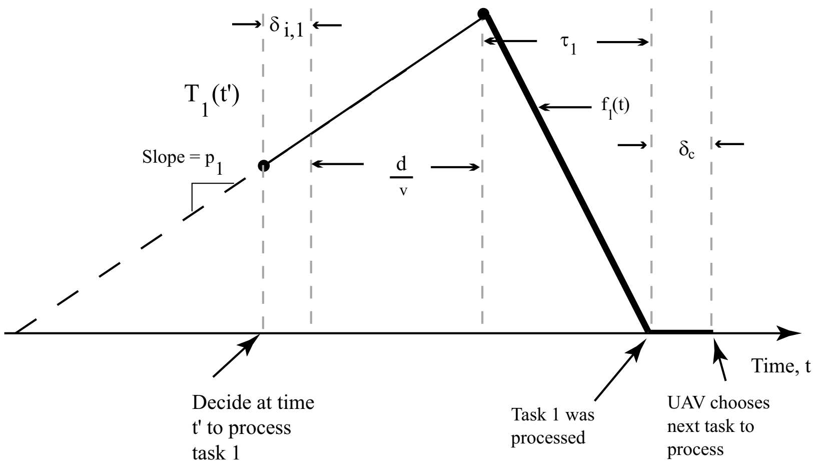


Figure 3.1: Illustration of timing of UAV decision-making and size of prioritized time since last processing.

how the processing of the task is completed as the UAV processes it. Figure 3.1 shows that $T_i(t) = f(t), \forall t, t' + \delta_{i,1} + \frac{d(x_v^1, x^1)}{v} \leq t \leq t' + \delta_{i,1} + \frac{d(x_v^1, x^1)}{v} + \tau_1$, and we assume that function $f(\cdot)$ must additionally satisfy $f(t) \leq f_\ell(t)$, where $f_\ell(t)$ is the bold line shown in Figure 3.1. Notice that the processing time, τ_1, does not depend on the value of $T_1(t' + \delta_{i,1} + \frac{d(x_v^1, x^1)}{v})$ since no matter what the current value of the prioritized time since last processing is, the UAV will always take τ_1 time units to process task 1. One consequence of this is that the slope of $f_\ell(t)$ will not be the same each time a UAV is processing a task.

3.1.2 Asynchronous Decision Making

Asynchronous decision making is accomplished in a way similar to the one described in Section 2.2. The only difference in this case is that UAV $j^u \in Q$ will hold the sets $U(t)$ and $S(t)$, which are defined next.

We define the set $U(t) \subset P$ as the set of ("unattended") tasks not processed or being pursued by any UAV at the current time t, while the set $U_j^a(t) = \left\{i_j^*(t)\right\} \cup U(t)$ is the set of tasks that can be considered for processing by UAV $j, j \in Q$. Here, $i_j^*(t)$ is the task being processed or pursued by UAV j at time t. Define $A(t)$ as the set of tasks processed or pursued by the group of M UAVs at the current time t; hence $P = U(t) \cup A(t), t \geq 0$.

Let

$$S(t) = \{\{T_i(t) : i \in U(t)\}, t_s\}$$

denote the information available for UAV j at time t when it makes a scheduling decision, where t_s is a time stamp that indicates the last time that the $T_i(t), i \in U(t)$ were updated.

We assume as we did in Section 2.2 that whenever a UAV $\ell \in Q$ where $\ell \neq j^u$ finishes processing a task at time t^f such that $T_{i_\ell^*}(t^f) = 0$ (i.e., the instant the task is processed), it broadcasts a request for the sets $U(t)$ and $S(t)$ to all the UAVs. Thus, during the time interval $[t^f, t^f + \delta_c]$ that UAV ℓ waits for the unattended set and S, $T_{i_\ell^*}(t') = 0, t' \in [t^f, t^f + \delta_c]$, which means that UAV ℓ keeps processing task i^* until it receives both sets U and S. This is consistent with our definition of the prioritized time since last processing since task i^* has already been completely processed at time t^f and it is not being ignored during the interval $[t^f, t^f + \delta_c]$ anymore. The instant that UAV ℓ gets $U(t)$ and $S(t)$, it becomes UAV j^u, it compares the t_s value to its local clock and it proceeds to update all $T_i(t), \forall i \in U(t)$ values if there exists any mismatch (the time stamp t_s indicates to UAV ℓ when was the last time that UAV j^u updated all $T_i(t), \forall i \in U(t)$). By doing so, UAV ℓ makes a decision on what task to process next with up-to-date information; however, note that even if the UAV's local clocks are not synchronized the decision maker will be able to select a new task to process.

Here, we also use a mutual exclusion algorithm which coordinates the access of all UAVs to the set $U(t)$ in such a way that this set can be accessed and updated by only one UAV at a time. We also assume UAV j^u has a "request queue", which could be built in the same way it was defined in Section 2.2. Moreover, we use the same definitions of Section 2.2 for the index of the sequence of times that UAV j makes allocation decisions (k^j), D_{k^j}, D_{k^j+1}, and $D_{k^{j^c}}$. Since $\delta_{i,j} \geq 0$ and $\delta_c > 0$ we know that $D_{k^j+1} > D_{k^{j^c}}$. By the definition of j^c, $D_{k^j} \leq D_{k^{j^c}} < D_{k^j+1}$, and we know that

$$D_{k^j+1} - D_{k^{j^c}} < D_{k^{j^c}+1} - D_{k^{j^c}} \leq \overline{\delta} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} + \overline{\tau} + \overline{\delta}_c \tag{3.1}$$

This can be seen in Figure 3.2. Note that in the time interval $t \in [D_{k^{j^c}}, D_{k^j+1}]$ the set $U(t)$ is constant.

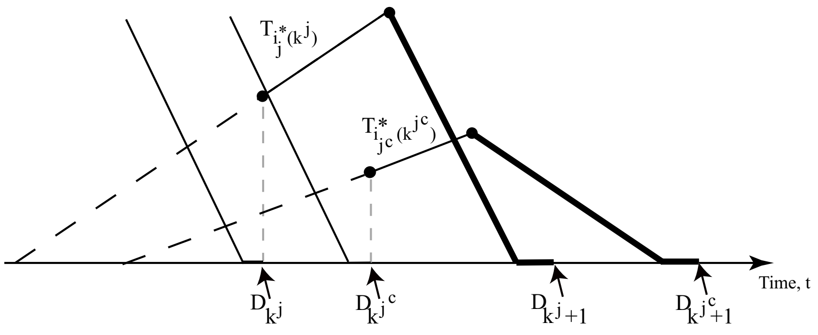


Figure 3.2: Example: decision times for $M = 2$ UAVs.

3.2 Stable Cooperative Scheduling

For the scenario in Section 3.1 tasks are spread across a limited region so that each UAV can process just one task at a time. Notice that there are M resources that must be shared (UAVs capabilities), and the scheduling strategy must decide how they are shared (what task to process). Note that if no UAV was actively engaged in processing to its environment, then clearly $T_i(t) \to \infty, i \in P, t \to \infty$ since no tasks are processed. We consider the scenario studied in this document to be "unstable" if $T_i(t) \to \infty, t \to \infty$ for any $i \in P$. Hence, the goal of the scheduling allocation strategy is to try to avoid $T_i(t) \to \infty$ for any $i \in P$ and indeed it will try to keep the $T_i(t)$ values *as small as possible* since this represents that the set of UAVs has recently processed each task. We consider a scenario to be "stable" when $T_i(t) < B, 0 < B < \infty, \forall i \in P$.

Next, we introduce a cooperative scheduling strategy that seeks to pay attention to two variables: high priority tasks and gas expenditure. Furthermore, we are interested in deriving bounds for the ignored time since last processing in terms of known parameters.

3.2.1 Cooperative Scheduling Strategies

Next, we introduce a particular cooperative scheduling strategy that we study in the remainder of the paper. Other strategies that can be applied to this particular problem can be found in Section 2.4 which are extensions of those in [1]. For it, at time k^j the cooperative scheduling strategy on each UAV j chooses to process task $i_j^*(k^j)$, such that

$$T_{i_j^*(k^j)}(D_{k^j}) - \frac{d(x_v^j(D_{k^j}), x^{i_j^*(k^j)})}{v} \geq$$

$$\frac{1}{N-M+1}\sum_{i_j\in U_j^a(D_{k^j})}\left[T_{i_j}(D_{k^j})-\frac{d(x_v^j(D_{k^j}),x^{i_j(k^j)})}{v}\right], \forall i_j \in U_j^a(D_{k^j}) \quad (3.2)$$

and makes no other decision until it has finished processing task $i_j^*(k^j)$ and received $U(t)$ and $S(t)$ (note that the quantities of both sides of Equation (3.2) can be positive or negative and if they are both zero then any task can be chosen for processing). Ties are broken with an arbitrary choice. Similar additional comments made for the strategy defined in Section 2.4 hold here also.

Equation (3.2) produces a set of admissible choices for what UAV j can process. The strategy "Process M Closest Highest Priority Tasks" where you choose $i_j^*(k^j)$ to process where

$$i_j^*(k^j) = \arg max_{i_j}\left\{T_{i_j}(D_{k^j})-\frac{d(x_v^j(D_{k^j}),x^{i_j(k^j)})}{v}\right\}, \forall i_j \in U_j^a(D_{k^j}) \quad (3.3)$$

is a special case of Equation (3.2) in the sense that it represents one possible choice for Equation (3.2); hence when in Section 3.3 we do stability analysis for Equation (3.2) it also applies if we use Equation (3.3) for our strategy.

Notice that Equation (3.2) could be viewed as a cost function and the goal of the strategy is to locally optimize the difference between the prioritized time since last processing and the travel time from current UAV location to the chosen task location. Another way to view this cost function is by thinking of the chosen task $i_j^*(k^j)$ by UAV j as the one whose prioritized "ignore time" combined with the UAVs travel time is greater or equal to the average value of those variables for each task contained in the set $U_j^a(D_{k^j})$. Thus, if all the tasks contained in the set $U_j^a(t)$ have the same prioritized time since last processing but they are located at different distances from a UAV, then this UAV chooses to process the task that is closest to it. In this sense the strategy is "myopic" (but consider the discussion in Section 1.1 on why "look-ahead" can be detrimental for the type of cooperative control problem we consider).

In some cases the p_i values are set a priori by constraints of the problem. In other cases it is possible to view them as controller design parameters that can be tuned to improve performance. Recall that the p_i parameters are embedded in the T_i variables, so by changing these parameters any UAV can put more or less emphasis in the time since last processing, that is, if the p_i values are all too small, then the UAV's will tend to choose the closest task at any decision time, whereas if all p_i values are too big, then the UAV's will tend to choose task to process based on the ignoring time neglecting the travel distance of the tasks.

3.3 Stability Analysis

In this section we present our main result, which is based on the stability (boundedness) of the t_i variables when the strategy defined in Equation (3.2) is used.

Theorem: Assume that $N > M$. For the cooperative scheduling strategy in Equation (3.2) a specific bound on the ultimate longest time that any UAV will ignore task $i \in P$ is given by

$$\lim_{t\to\infty} t_i(t) \le \max\{B^1, B^2\}$$

where

$$\begin{aligned} B^1 &= \frac{\left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c\right)}{\underline{p}} \left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) \left(NM - M^2 + M + 1\right) \\ &+ \frac{\overline{d}}{\underline{p}v} \left(M(N-M+1) + \overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) + \frac{\overline{\delta}}{\underline{p}}\overline{p} \end{aligned}$$

$$B^2 = \frac{\left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c + \frac{\overline{d}}{v}\right)}{\underline{p}} \left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) (NM - M^2 + M + 1) + \frac{\overline{p}}{\underline{p}} \left(\overline{\delta} + \frac{\overline{d}}{v}\right)$$

$\underline{p} = \min_i\{p_i\}$, $\overline{p} = \max_i\{p_i\}$, and $\overline{d} = \max\{d(x_v^j, x^i)\}$

Proof: Let

$$V(t) = \sum_{i=1}^{N} T_i(t)$$

The proof to follow focuses on the strategy where the task i_j^* is chosen by UAV $j \in Q$. This proof proceeds by extending the one in Section 2.5. There are, however, fundamental differences since the scheduling strategy is quite different (it represents a desire to achieve more than one objective), and the slope of $T_i(t)$ could be different every time that any UAV is processing a task as pointed out in Section 3.1 whereas for the problem in Section 2.5 that slope is always constant. One implication of this last point will be that we do not need a "capacity condition" as we do in Section 2.5.

First, note that

$$V(t) = \sum_{j=1}^{M} \left(T_{i_j^*}(t) + \sum_{i \in U(t)} \frac{T_i(t)}{M} \right) \tag{3.4}$$

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

$$V_j(t) = T_{i_j^*}(t) + \sum_{i \in U(t)} \frac{T_i(t)}{M} \tag{3.5}$$

Consider the values at the *set* of decision times D_{k^j},

$$\sum_{j=1}^{M} V_j(D_{k^j}) \quad = \sum_{j=1}^{M} \left[T_{i_j^*(k^j)}(D_{k^j}) + \sum_{i \in U(D_{k^j})} \frac{T_i(D_{k^j})}{M} \right] \tag{3.6}$$

Since $T_{i_j^*(k^j)}(D_{k^j+1}) = 0, \forall j \in Q$ ($i_j^*(k^j)$ was the task that was just processed by UAV $j \in Q$),

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) = \sum_{j=1}^{M} \left[\sum_{i \in U(D_{k^j+1})} \frac{T_i(D_{k^j+1})}{M} \right] \tag{3.7}$$

Note that by definition $U(D_{k^j+1}) = U(D_{k^{j^c}})$ since the unattended set will not change for every UAV j for t such that, $D_{k^{j^c}} \leq t \leq D_{k^j+1}$. Hence, considering how long the tasks have been ignored during this time period,

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) = \sum_{j=1}^{M} \left\{ \sum_{i \in U(D_{k^{j^c}})} \left[\frac{T_i(D_{k^{j^c}})}{M} + \frac{p_i}{M}(D_{k^j+1} - D_{k^{j^c}}) \right] \right\} \tag{3.8}$$

From Equation (3.5) for any UAV j^c at time $D_{k^{j^c}}$

$$\sum_{i \in U(D_{k^{j^c}})} \frac{T_i(D_{k^{j^c}})}{M} = V_{j^c}(D_{k^{j^c}}) - T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) \tag{3.9}$$

We use Equations (3.1), (3.8), and (3.9) to obtain

$$\begin{aligned}
\sum_{j=1}^{M} V_j(D_{k^j+1}) &\leq \sum_{j=1}^{M} \Bigg\{ V_{j^c}(D_{k^{j^c}}) - T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) \\
&+ \sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} \left(\overline{\delta} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} + \overline{\tau} + \overline{\delta}_c \right) \Bigg\} \\
&= \sum_{j=1}^{M} \Bigg\{ V_{j^c}(D_{k^{j^c}}) - \Bigg(T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) \\
&- \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} \sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} \Bigg) \\
&+ (\overline{\delta} + \overline{\tau} + \overline{\delta}_c) \sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} \Bigg\}
\end{aligned} \tag{3.10}$$

Focus now on the first $\sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M}$ term in Equation (3.10) and notice that the proof can be divided in two cases as follows: a) When $\sum_{i \in U(D_{k^{j^c}})} p_i \leq M$, $\forall k^{j^c}$, and b) when $\sum_{i \in U(D_{k^{j^c}})} p_i > M$, $\forall k^{j^c}$.

Case a): To start, we seek to remove the first $\sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M}$ term in Equation (3.10) in order to make the term in parenthesis the same as our strategy in Equation (3.2). Note that

$$\begin{aligned}
\sum_{j=1}^{M} V_j(D_{k^j+1}) &\leq \sum_{j=1}^{M} \Bigg\{ V_{j^c}(D_{k^{j^c}}) - \left(T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) - \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} \right) \\
&+ (\overline{\delta} + \overline{\tau} + \overline{\delta}_c) \sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} \Bigg\}
\end{aligned} \tag{3.11}$$

We use the definition of the cooperative scheduling strategy for $j = j^c$ in Equation (3.11)

$$T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) - \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} \geq$$
$$\frac{1}{N-M+1} \sum_{i_j \in U_j^a(D_{k^{j^c}})} \left[T_{i_j}(D_{k^{j^c}}) - \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i_j(k^{j^c})})}{v} \right], \forall i_j \in U_j^a(D_{k^j})$$

But notice that

$$T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) - \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} \geq$$
$$\frac{V_{j^c}(D_{k^{j^c}})}{(N-M+1)} - \frac{1}{N-M+1} \sum_{i_j \in U_j^a(D_{k^{j^c}})} \left[\frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i_j(k^{j^c})})}{v} \right]$$

Combine this with Equation (3.11) to get

$$\begin{aligned}\sum_{j=1}^{M} V_j(D_{k^j+1}) \quad &\leq \quad \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) \left(1 - \frac{1}{N-M+1} \right) \right. \\ &+ \quad \frac{1}{N-M+1} \sum_{i_j \in U_j^a(D_{k^{j^c}})} \left[\frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i_j(k^{j^c})})}{v} \right] \\ &+ \quad \left. (\overline{\delta} + \overline{\tau} + \overline{\delta}_c) \sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} \right\} \qquad (3.12)\end{aligned}$$

Also

$$\begin{aligned}\sum_{j=1}^{M} V_j(D_{k^j+1}) \quad &\leq \quad \sum_{j=1}^{M} \left\{ V_{j^c}(D_{k^{j^c}}) \left(1 - \frac{1}{N-M+1} \right) + \frac{\overline{d}}{v} \right. \\ &+ \quad \left. (\overline{\delta} + \overline{\tau} + \overline{\delta}_c) \left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p} \right) \right\} \qquad (3.13)\end{aligned}$$

since

$$\sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} = \sum_{i=1}^{N} \frac{p_i}{M} - \sum_{i \in A(D_{k^{j^c}})} \frac{p_i}{M} \leq \sum_{i=1}^{N} \frac{p_i}{M} - M \frac{\underline{p}}{M} \qquad (3.14)$$

Notice here that we can also obtain a tighter bound on the right side of Equation (3.14) by using the M minimum values of p_i rather than $M\underline{p}$.

Define

$$\beta = \frac{\overline{d}}{v} + (\overline{\delta} + \overline{\tau} + \overline{\delta}_c)\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)$$

Notice that $\beta > 0$ and that

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) \leq \sum_{j=1}^{M} \left\{V_{j^c}(D_{k^{j^c}})\left(1 - \frac{1}{N-M+1}\right) + \beta\right\}$$

But, notice that

$$\sum_{j=1}^{M} V_j(D_{k^j+1}) \leq \left(1 - \frac{1}{N-M+1}\right)\sum_{j=1}^{M} \{V_{j^c}(D_{k^{j^c}})\} + M\beta \tag{3.15}$$

This means that we have a contractive mapping in Equation (3.15). Notice that by the same arguments explained in Section 2.5, the contractive mapping is also valid in this case for all $t \geq 0$.

Define for $k \geq 0$

$$\overline{V}(k) = \sum_{j=1}^{M} V_{j^c}(D_{k^{j^c}})$$

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 Equation (3.15) to get

$$\overline{V}(k+1) \leq \gamma\overline{V}(k) + \zeta \tag{3.16}$$

where $0 < \gamma = \left(1 - \frac{1}{N-M+1}\right) < 1$ and $\zeta = M\beta$ which are both constants. Equation (3.16) is a difference inequality with a solution that is bounded for all k by Equation (2.21). The same comments made in Section 2.5 for the transient and ultimate bound on $\overline{V}(k)$ at the decision times hold for this case also. Now,

$$\frac{\zeta}{1-\gamma} = M(N-M+1)\beta \tag{3.17}$$

Next, we need to consider the times in between the decision times. To do this, note that for all j^c, k^{j^c}, and k

$$V_{j^c}(D_{k^{j^c}}) \leq \overline{V}(k) \tag{3.18}$$

Next, consider the case where $V_{j^c}\left(D_{k^{j^c}} + \delta_{i,j^c(k^{j^c})} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v}\right)$ occurs at any t, $D_{k^{j^c}} < t < D_{k^j+1}$. If this is the case, then for any $T_i(t) \leq f_\ell(t)$ and any t, $D_{k^{j^c}} \leq t \leq D_{k^{j^c}} + \delta_{i,j^c(k^{j^c})} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v}$ we have

$$V_{j^c}(t) \leq V_{j^c}\left(D_{k^{j^c}} + \delta_{i,j^c(k^{j^c})} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v}\right) \tag{3.19}$$

On the other hand, for any t, $D_{k^{j^c}} + \delta_{i,j^c(k^{j^c})} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} \leq t \leq D_{k^j+1}$ if we let $\gamma = \delta_{i,j^c(k^{j^c})} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v}$ we have

$$\begin{aligned}
V_{j^c}(D_{k^j+1}) &= T_{i^*_{j^c}(k^{j^c})}(D_{k^j+1}) + \sum_{i \in U(D_{k^j+1})} \frac{T_i(D_{k^j+1})}{M} \\
&= T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}} + \gamma) - \frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}} + \gamma)(D_{k^j+1} - D_{k^{j^c}} - \gamma)}{\tau_{i^*_{j^c}(k^{j^c})}} \\
&+ \sum_{i \in U(D_{k^{j^c}}+\gamma)} \frac{T_i(D_{k^{j^c}} + \gamma)}{M} + \sum_{i \in U(D_{k^{j^c}}+\gamma)} \frac{p_i(D_{k^j+1} - D_{k^{j^c}} - \gamma)}{M} \\
&= V_{j^c}(D_{k^{j^c}} + \gamma) + \underbrace{(D_{k^j+1} - D_{k^{j^c}} - \gamma)}_{>0}\left(\sum_{i \in U(D_{k^{j^c}}+\gamma)} \frac{p_i}{M}\right. \\
&- \left.\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}} + \gamma)}{\tau_{i^*_{j^c}(k^{j^c})}}\right)
\end{aligned} \tag{3.20}$$

Here, we have two possibilities:

Case a.1): If the slope of the task currently being processed is greater or equal to the sum of the slopes of the unattended tasks over M, i.e.,

$$\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}} + \gamma)}{\tau_{i^*_{j^c}(k^{j^c})}} \geq \sum_{i \in U(D_{k^{j^c}}+\gamma)} \frac{p_i}{M},\ \forall k^{j^c}$$

then $V_{j^c}(D_{k^j+1}) \leq V_{j^c}(D_{k^{j^c}}+\gamma)$. Thus, we obtain that $V_{j^c}(t) \leq V_{j^c}(D_{k^{j^c}}+\gamma)$ for any t, $D_{k^{j^c}} \leq t \leq D_{k^j+1}$.

For this case then

$$\begin{aligned}
V_{j^c}(D_{k^{j^c}}+\gamma) &= T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}+\gamma) + \sum_{i\in U(D_{k^{j^c}}+\gamma)} \frac{T_i(D_{k^{j^c}}+\gamma)}{M} \\
&= T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) + \overline{\delta} p_{i^*_{j^c}(k^{j^c})} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})}) p_{i^*_{j^c}(k^{j^c})}}{v} \\
&\quad + \sum_{i\in U(D_{k^{j^c}})} \frac{T_i(D_{k^{j^c}})}{M} + \sum_{i\in U(D_{k^{j^c}})} \frac{p_i\overline{\delta}}{M} + \sum_{i\in U(D_{k^{j^c}})} \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})}) p_i}{vM} \\
&= V_{j^c}(D_{k^{j^c}}) + \overline{\delta}\left(p_{i^*_{j^c}(k^{j^c})} + \sum_{i\in U(D_{k^{j^c}})} \frac{p_i}{M}\right) \\
&\quad + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v}\left(p_{i^*_{j^c}(k^{j^c})} + \sum_{i\in U(D_{k^{j^c}})} \frac{p_i}{M}\right) \\
&\leq V_{j^c}(D_{k^{j^c}}) + \overline{\delta}\left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) + \frac{\overline{d}}{v}\left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) \qquad (3.21)
\end{aligned}$$

where both the second and last terms of the right hand side of the equation was derived by using Equation (3.14).

Next, note that for all j^c and $t \geq 0$

$$T_{i^*_{j^c}}(t) + \sum_{i\in U(t)} \frac{T_i(t)}{M} = V_{j^c}(t)$$

so that using Equations (3.19) and (3.21)

$$T_{i^*_{j^c}}(t) + \sum_{i\in U(t)} \frac{T_i(t)}{M} \leq V_{j^c}(D_{k^{j^c}}) + \overline{\delta}\left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) + \frac{\overline{d}}{v}\left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)$$

Hence, using Equation (3.17)

$$\lim_{t\to\infty} t_{i^*_{j^c}}(t) \leq \lim_{t\to\infty}\left[t_{i^*_{j^c}}(t) + \sum_{i\in U(t)} \frac{t_i(t)}{M}\right] \leq \frac{M}{\underline{p}}(N-M+1)\beta$$

$$\begin{aligned}
&+ \quad \frac{\overline{\delta}}{\underline{p}}\left(\overline{p}+\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)+\frac{\overline{d}}{\underline{p}v}\left(\overline{p}+\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right) \\
&\leq \quad \frac{M}{\underline{p}}(N-M+1)\left(\frac{\overline{d}}{v}+(\overline{\delta}+\overline{\tau}+\overline{\delta}_c)\left(\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)\right) \\
&+ \quad \frac{\overline{\delta}}{\underline{p}}\left(\overline{p}+\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)+\frac{\overline{d}}{\underline{p}v}\left(\overline{p}+\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right) \\
&\leq \quad \frac{\left(\overline{\delta}+\overline{\tau}+\overline{\delta}_c\right)}{\underline{p}}\left(\sum_{i=1}^{N}p_i-M\underline{p}\right)(N-M+1) \\
&+ \quad \frac{\overline{d}}{\underline{p}v}\left(M(N-M+1)+\overline{p}+\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)+\frac{\overline{\delta}}{\underline{p}}\left(\overline{p}+\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)
\end{aligned} \tag{3.22}$$

Now, we must show that each task will get chosen by some UAV $j \in Q$ infinitely often so that every task becomes $i^*_{j^c}$ persistently so that Equation (3.22) provides a bound for *each* $t_i(t)$ $i \in P$. Note that we have a bound for every k^{j^c} for every $t_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}})$ via Equation (3.22), and using Equation (3.1), $D_{k^j+1} - D_{k^{j^c}}$ is bounded. This results in a maximum bound on the time that the unattended set will not be changed. "Ignored time" for tasks rises so eventually any ignored task in $U(t)$ will be taken off $U(t)$ and hence become $i^*_{j^c}$.

Case a.2): Now, we study the case when $\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}+\gamma)}{\tau_{i^*_{j^c}(k^{j^c})}} < \sum_{i\in U(D_{k^{j^c}+\gamma})}\frac{p_i}{M}$, for *some* k^{j^c}. If this is the case, then $V_{j^c}(D_{k^j+1}) > V_{j^c}(D_{k^{j^c}}+\gamma)$. Thus, we obtain that $V_{j^c}(t) \leq V_{j^c}(D_{k^j+1})$ for any t, $D_{k^{j^c}} \leq t \leq D_{k^j+1}$. However, this cannot be *always* the case because if $V_{j^c}(D_{k^j+1}) > V_{j^c}(D_{k^{j^c}}+\gamma)$, $\forall k^{j^c}$, then the function V_{j^c} is increasing all the time, which means that all T_i's in the unattended set grow much more faster than the value of $T_{i^*_{j^c}}$ being currently processed. But if this is the case then sooner or later the condition $\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}+\gamma)}{\tau_{i^*_{j^c}(k^{j^c})}} \geq \sum_{i\in U(D_{k^{j^c}+\gamma})}\frac{p_i}{M}$ will be satisfied since any ignored task will later become $i^*_{j^c}$ and this is the reason why we say that $\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}+\gamma)}{\tau_{i^*_{j^c(k^{j^c})}}} < \sum_{i\in U(D_{k^{j^c}+\gamma})} \frac{p_i}{M}$ holds for some k^{j^c}.

Now, for this case

$$\begin{aligned}
V_{j^c}(D_{k^j+1}) &= T_{i^*_{j^c}(k^{j^c})}(D_{k^j+1}) + \sum_{i\in U(D_{k^j}+1)} \frac{T_i(D_{k^j+1})}{M} \\
&= T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) + \gamma p_{i^*_{j^c}(k^{j^c})} \\
&- (D_{k^j+1} - D_{k^{j^c}} - \gamma)\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^j+1} - D_{k^{j^c}} - \gamma)}{\tau_{i^*_{j^c}(k^{j^c})}} \\
&+ \sum_{i\in U(D_{k^{j^c}})} \frac{T_i(D_{k^{j^c}})}{M} + (D_{k^j+1} - D_{k^{j^c}}) \sum_{i\in U(D_{k^{j^c}})} \frac{p_i}{M} \\
&\le V_{j^c}(D_{k^{j^c}}) + \gamma p_{i^*_{j^c}(k^{j^c})} + (D_{k^j+1} - D_{k^{j^c}}) \sum_{i\in U(D_{k^{j^c}})} \frac{p_i}{M} \\
&\le V_{j^c}(D_{k^{j^c}}) + \overline{p}\left(\overline{\delta} + \frac{\overline{d}}{v}\right) + \left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c + \frac{\overline{d}}{v}\right)\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)
\end{aligned} \tag{3.23}$$

Next, we apply the same steps taken in Equation (3.22) to obtain that

$$\begin{aligned}
\lim_{t\to\infty} t_{i^*_{j^c}}(t) &\le \frac{M}{\underline{p}}(N-M+1)\beta + \frac{\overline{p}}{\underline{p}}\left(\overline{\delta} + \frac{\overline{d}}{v}\right) + \frac{\left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c + \frac{\overline{d}}{v}\right)}{\underline{p}}\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) \\
&\le \frac{M}{\underline{p}}(N-M+1)\left(\frac{\overline{d}}{v} + (\overline{\delta} + \overline{\tau} + \overline{\delta}_c)\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)\right) \\
&+ \frac{\overline{p}}{\underline{p}}\left(\overline{\delta} + \frac{\overline{d}}{v}\right) + \frac{\left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c + \frac{\overline{d}}{v}\right)}{\underline{p}}\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) \\
&\le \frac{(\overline{\delta} + \overline{\tau} + \overline{\delta}_c)}{\underline{p}}\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)\left(NM - M^2 + M + 1\right) \\
&+ \frac{\overline{d}}{\underline{p}v}\left(M(N-M+1) + \overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) + \frac{\overline{\delta}}{\underline{p}}\overline{p}
\end{aligned} \tag{3.24}$$

where it can be seen that the bound obtained in Equation (3.24) is greater than the one shown in Equation (3.22) due to the first term in Equation (3.24).

Case b): Consider $\sum_{i \in U(D_{k^{j^c}})} p_i > M$, $\forall k^{j^c}$. Note that for this case we have

$$\begin{aligned}
T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}) - \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v} &> T_{i^*_j(k^j)}(D_{k^j}) \\
&- \frac{d(x_v^j(D_{k^j}), x^{i^*_j(k^j)})}{v} \sum_{i \in U(D_{k^j})} \frac{p_i}{M} \\
&\geq \frac{1}{N-M+1} \sum_{i_j \in U_j^a(D_{k^{j^c}})} \Bigg[T_{i_j}(D_{k^{j^c}}) \\
&- \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i_j(k^{j^c})})}{v} \sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} \Bigg]
\end{aligned} \tag{3.25}$$

It can be easily seen that if we use the strategy shown in Equation (3.25) in Equation (3.10) then we are solving the problem for the strategy defined in Equation (3.2). The final result will be, of course, more conservative. As in the last case, we have here two possibilities and we study them next.

Case b.1): If $\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}+\gamma)}{\tau_{i^*_{j^c}(k^{j^c})}} \geq \sum_{i \in U(D_{k^{j^c}}+\gamma)} \frac{p_i}{M}$, $\forall k^{j^c}$, then $V_{j^c}(D_{k^j+1}) \leq V_{j^c}(D_{k^{j^c}} + \gamma)$. Thus, we obtain that $V_{j^c}(t) \leq V_{j^c}(D_{k^{j^c}} + \gamma)$ for any t, $D_{k^{j^c}} \leq t \leq D_{k^j+1}$.

Note that for this case Equations (3.17) and (3.18) still hold except that β is different from the one obtained in part a). Furthermore, we have already derived in Equation (3.21) that

$$\begin{aligned}
V_{j^c}\left(D_{k^{j^c}} + \delta_{i,j^c(k^{j^c})} + \frac{d(x_v^{j^c}(D_{k^{j^c}}), x^{i^*_{j^c}(k^{j^c})})}{v}\right) &\leq V_{j^c}(D_{k^{j^c}}) + \overline{\delta}\left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) \\
&+ \frac{\overline{d}}{v}\left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)
\end{aligned} \tag{3.26}$$

Now, the procedure to obtain a bound for this case is the same as the one followed in case a.1) except that for this particular case we have

$$\beta = \frac{\overline{d}}{v}\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) + (\overline{\delta} + \overline{\tau} + \overline{\delta}_c)\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)$$

where the term that multiplies to $\frac{\overline{d}}{v}$ in the first term of the right side of the above equation is derived from Equation (3.14). Notice that this term was not present in *case a)* since $\sum_{i \in U(D_{k^{j^c}})} \frac{p_i}{M} < 1$.

Hence,

$$\begin{aligned}
\lim_{t\to\infty} t_{i^*_{j^c}}(t) &\leq \frac{(N-M+1)}{\underline{p}} \left(\sum_{i=1}^{N} p_i - M\underline{p}\right)\left(\frac{\overline{d}}{v} + \overline{\delta} + \overline{\tau} + \overline{\delta}_c\right) \\
&+ \frac{\left(\frac{\overline{d}}{v} + \overline{\delta}\right)}{\underline{p}} \left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) \\
&\leq \frac{(\overline{\delta} + \overline{\tau} + \overline{\delta}_c)}{\underline{p}} \left(\sum_{i=1}^{N} p_i - M\underline{p}\right)(N-M+1) \\
&+ \frac{\overline{d}}{\underline{p}v} \left(\sum_{i=1}^{N} p_i - M\underline{p}\right)\left(N-M+1+\frac{1}{M}\right) \\
&+ \frac{\overline{\delta}}{\underline{p}} \left(\overline{p} + \sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) + \frac{\overline{p}\overline{d}}{\underline{p}v}
\end{aligned} \tag{3.27}$$

Case b.2): If $\frac{T_{i^*_{j^c}(k^{j^c})}(D_{k^{j^c}}+\gamma)}{\tau_{i^*_{j^c}(k^{j^c})}} < \sum_{i \in U(D_{k^{j^c}+\gamma})} \frac{p_i}{M}$, for some k^{j^c}, then $V_{j^c}(D_{k^j+1}) > V_{j^c}(D_{k^{j^c}} + \gamma)$. Thus, we obtain that $V_{j^c}(t) \leq V_{j^c}(D_{k^j+1})$ for any t, $D_{k^{j^c}} \leq t \leq D_{k^j+1}$. Therefore, by using both the same arguments and Equation (3.23) as in case a.2) we obtain that

$$V_{j^c}(D_{k^j} + 1) \leq V_{j^c}(D_{k^{j^c}}) + \overline{p}\left(\overline{\delta} + \frac{\overline{d}}{v}\right) + \left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c + \frac{\overline{d}}{v}\right)\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)$$

Hence,

$$\begin{aligned}
&\lim_{t\to\infty} t_{i^*_{j^c}}(t) \leq \frac{M}{\underline{p}}(N-M+1)\beta + \frac{\overline{p}}{\underline{p}}\left(\overline{\delta} + \frac{\overline{d}}{v}\right) + \frac{\left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c + \frac{\overline{d}}{v}\right)}{\underline{p}}\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) \\
&\leq \frac{M}{\underline{p}}(N-M+1)\left[\frac{\overline{d}}{v}\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right) + (\overline{\delta} + \overline{\tau} + \overline{\delta}_c)\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)\right] \\
&+ \frac{\overline{p}}{\underline{p}}\left(\overline{\delta} + \frac{\overline{d}}{v}\right) + \frac{\left(\overline{\delta} + \overline{\tau} + \overline{\delta}_c + \frac{\overline{d}}{v}\right)}{\underline{p}}\left(\sum_{i=1}^{N} \frac{p_i}{M} - \underline{p}\right)
\end{aligned}$$

$$
\begin{aligned}
&\le \frac{(N-M+1)}{\underline{p}}\left(\sum_{i=1}^{N} p_i - M\underline{p}\right)\frac{\overline{d}}{v} + \frac{(N-M+1)}{\underline{p}}\left(\sum_{i=1}^{N} p_i - M\underline{p}\right)\left(\overline{\delta}+\overline{\tau}+\overline{\delta}_c\right)\\
&+\frac{\overline{p}}{\underline{p}}\left(\overline{\delta}+\frac{\overline{d}}{v}\right) + \frac{\left(\overline{\delta}+\overline{\tau}+\overline{\delta}_c+\frac{\overline{d}}{v}\right)}{\underline{p}}\left(\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)\\
&\le \frac{\left(\overline{\delta}+\overline{\tau}+\overline{\delta}_c\right)}{\underline{p}}\left(\sum_{i=1}^{N} p_i - M\underline{p}\right)(N-M+1)\\
&+\frac{\overline{d}}{\underline{p}v}\left(\sum_{i=1}^{N} p_i - M\underline{p}\right)\left(N-M+1+\frac{1}{M}\right)\\
&+\frac{\overline{\delta}}{\underline{p}}\left(\overline{p}+\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)+\frac{\overline{p}\overline{d}}{\underline{p}v}+\frac{\left(\overline{\tau}+\overline{\delta}_c\right)}{\underline{p}}\left(\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)\\
&\le \frac{\left(\overline{\delta}+\overline{\tau}+\overline{\delta}_c+\frac{\overline{d}}{v}\right)}{\underline{p}}\left(\sum_{i=1}^{N}\frac{p_i}{M}-\underline{p}\right)(NM-M^2+M+1)+\frac{\overline{p}}{\underline{p}}\left(\overline{\delta}+\frac{\overline{d}}{v}\right) \qquad (3.28)
\end{aligned}
$$

where we can see that the bound obtained in Equation (3.28) is greater than the one obtained in Equation (3.27).

On the other hand, it can be easily seen that if $V_{j^c}(D_{k^{j^c}}+\gamma)$ does not occur at any t, $D_{k^{j^c}} \le t \le D_{k^j+1}$, then this can not happen all the time since there exists a time interval different from $D_{k^{j^c}} \le t \le D_{k^j+1}$ where this event will take place. Note that we do not need to study this particular case since this is a special case of the ones studied in parts a.2) and b.2). This concludes the proof of the theorem. ■

Next, we make a few remarks about the ultimate bound obtained. First of all, note that when the time delays, $\overline{\delta}$ and $\overline{\delta}_c$, increase, the bound also increases (e.g., network delays can result in ignoring tasks longer). The ultimate bound decreases if the velocity of a UAV increases since the UAVs can move faster to process tasks. If the priority values assigned to tasks increase then the ultimate bound does also since then we may spend more time processing the higher priority tasks and hence an increased amount of time ignoring the lower priority tasks. If all tasks are spread out more (i.e., $\overline{d}$ increases), then the ultimate bound increases since it takes longer to travel to process tasks, and this provides a clear idea how task density could affect the ultimate bound. The ultimate bound increases if the processing time of every task is increased since the UAVs are busier processing and hence ignore other tasks longer. If the number of tasks and UAVs are about the same, then the ultimate bound decreases.

Finally, one important point to highlight here is the fact that the analysis can take into account the behavior of heterogeneous UAVs (i.e., UAVs with different processing capabilities for the same task). It also inherently takes into account trade-offs between task priorities and spatial separation between tasks.

3.4 Simulations

Here, we seek to determine how cooperative strategies perform when we implement the strategy proposed in Section 3.2.1. In particular, we will show how the cooperative scheduler will ignore lower priority tasks longer and how ignored time for tasks is affected by communication delays. On the other hand, we introduce a noncooperative framework and we are interested in deriving design guidelines for both noncooperative and cooperative scheduling systems.

3.4.1 Influences of Priorities and Communication Delays on Cooperative Scheduling Strategies

We run two simulations with the following values: the sampling time is $T_s = 0.1$ sec., the length of the simulation is 16.67 min., there are $N = 6$ tasks, $M = 4$ UAVs, the switching times for UAVs 1 and 2 are $\delta^1_{i,j} = \delta^2_{i,j} = \delta^3_{i,j} = \delta^4_{i,j} = 0$ sec., and we consider a fixed communication delay, $\delta_c = 60$ seconds.

The first simulation considers $p_i = p_j = 50, i, j \in P$. Figure 3.3 shows in the top 6 plots the time since last processing of any UAV for all tasks, and the unattended tasks in the bottom plot. Next, we let $p_1 = 10$ and we do not change the rest of

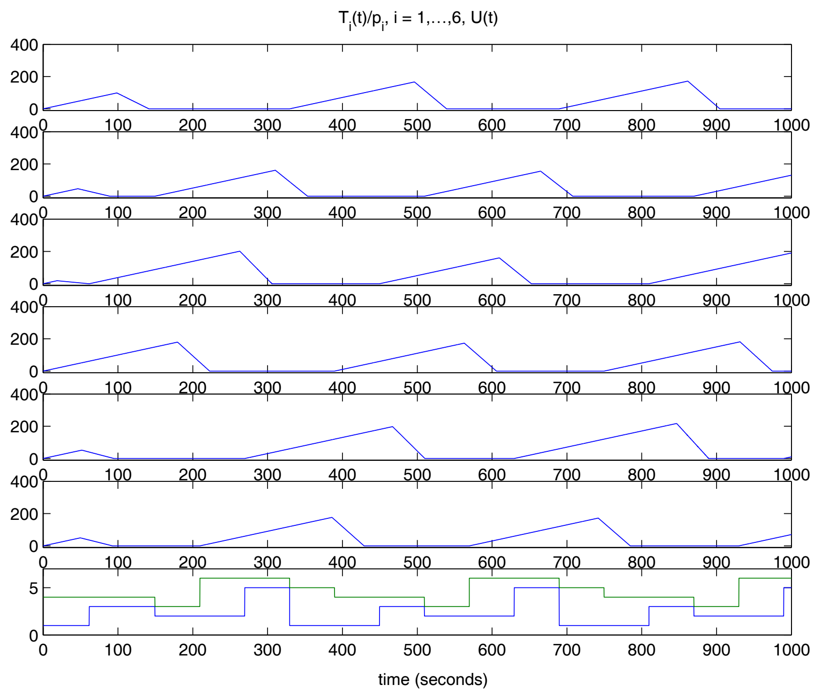


Figure 3.3: Time since last processing (seconds) of every task and the unattended set.

the values. The result of the simulation can be seen in Figure 3.4. Notice that since the priority of task 1 has been decreased, this task is ignored more than in the first simulation. This shows that UAVs can be forced to ignored tasks by assigning low priority values to tasks. Furthermore, it can be seen that there are intervals where the waiting time for the unattended set $U(t)$ is greater than $\delta_c = 60$ seconds due to the amount of previous requests that have been made by other UAVs, i.e., there are more than one UAV waiting at a specific time for the set $U(t)$.

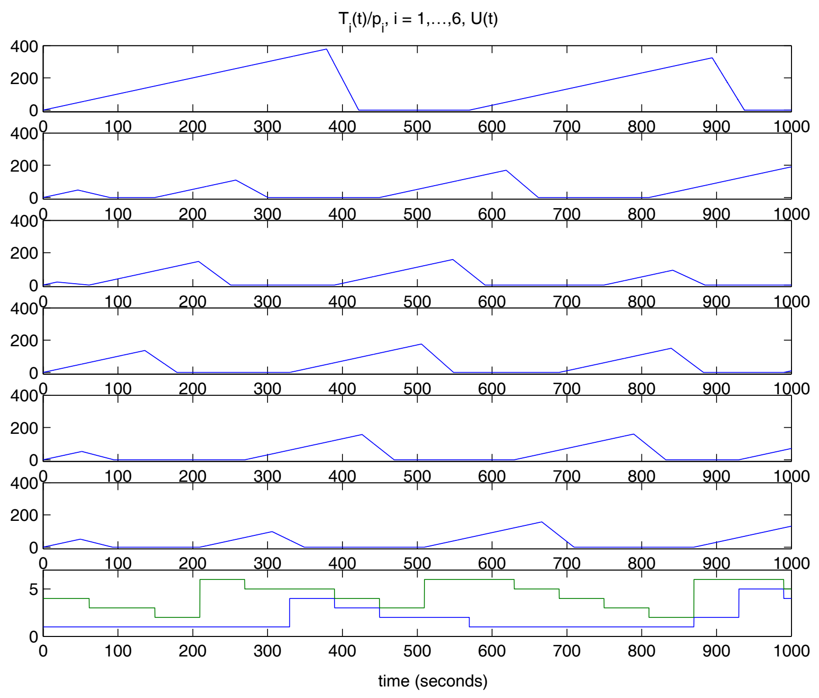


Figure 3.4: Time since last processing (seconds) of every task and the unattended set when $p_1 = 10$.

Now, we let all the priorities have the same values ($p_i = p_j = 50, i, j \in P$) and we study two cases: small communication delays, $\delta_c \to 0$, as shown in Figure 3.5 and random but bounded delays, $\delta_c \leq 180$ seconds, as shown in Figure 3.6. It is seen from the figures that the delay decreases the rate at which tasks are processed and also increases the ignored time of each task.

3.4.2 Comparative Analysis: Noncooperative Vs. Cooperative Strategies

Here, we consider a group of M UAVs not connected over a communication network. Due to the lack of communication and hence lack of coordinated decision making we call such a strategy "noncooperative." Each UAV has its own

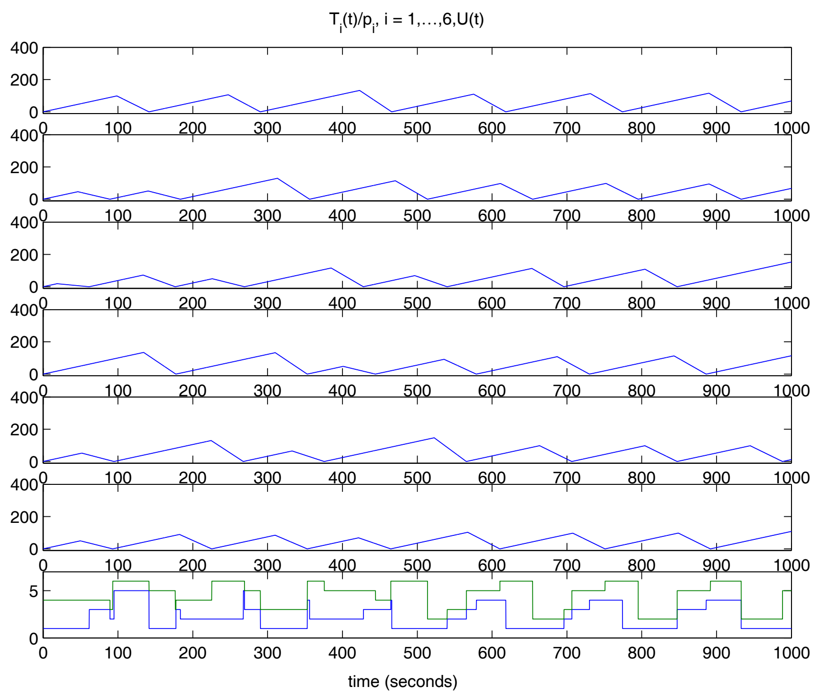


Figure 3.5: Time since last processing (seconds) of every task and the unattended set for small communication delays.

$T_{i_j}, \forall i \in P, \forall j \in Q$ and the prioritized time since last processing for each task is given by $T_i(t) = min_j\{T_{i_j}(t)\}, \forall j \in Q$. Notice that from the task point of view, this represents how long a task has been ignored by any UAV. For this case, UAVs make scheduling decision based on Equation (3.2) but considering all tasks at each decision time, hence there could be cases where one or more UAVs are processing the same task during a certain time interval.

We seek to determine if cooperative strategies are always superior to noncooperative ones. We would like to answer the following questions: Is it always beneficial to cooperate? If not, what are the conditions under it is not good to cooperate? In order to answer these questions we run a Monte Carlo simulation with

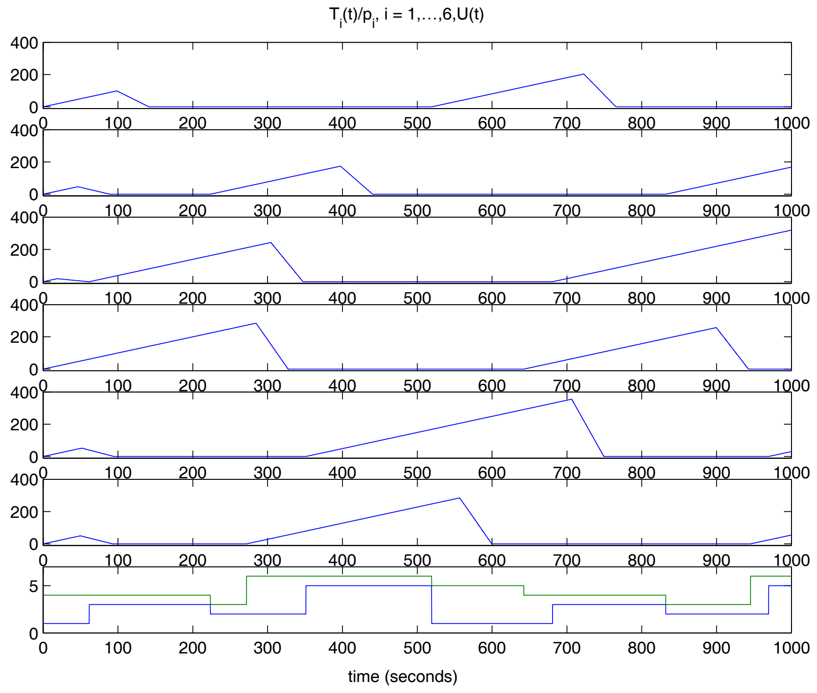


Figure 3.6: Time since last processing (seconds) of every task and the unattended set for random but bounded communication delays.

the following values: the sampling time $T_s = 0.1$ seconds, fixed delays $\delta_c(s) = \{1, 10, 50, 100, 150, 200, 300, 400\}$, a set of densities with standard deviation $\sigma(m) = \{500, 1000, 2500, 4000, 5000\}$ (we use a 2D Gaussian distribution for tasks with mean $(\overline{x}, \overline{y}) = [5000, 5000]$ meters), no switching delays, and the simulation length of 2500 seconds. Each case consists of 100 simulations. The number of simulations for each delay-density combination was chosen to be 100 because the standard deviation of the performance measures (introduced further) did not change significantly (i.e., settled to a constant value) beyond 100 simulations as it can be seen in the error-bar plots shown in Figure 3.7.

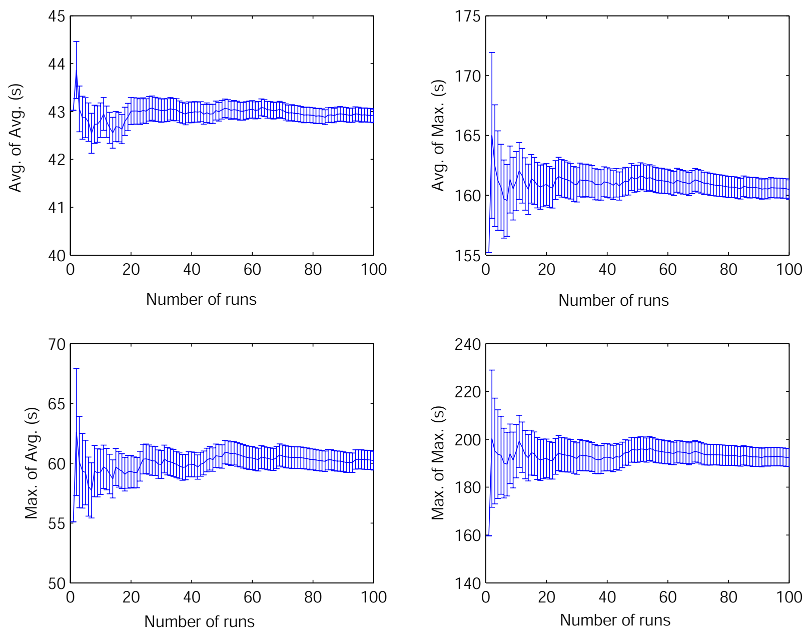


Figure 3.7: Error-bar plot of performance measure and standard deviation for $\delta_c =$ 100 sec. and $\sigma =$500 m.

Now, how can we really compare the performance of noncooperative and cooperative strategies? In order to do that, we need to introduce a way to evaluate the performance of the UAVs. There are several ways to measure performance of the scheduling strategies [15]. Here, we will compute the average of the ignored time since any task has been processed $\frac{1}{N}\sum_{i=1}^{N} T_i(k)$ at each step k. We will also compute the time average of this quantity (i.e., the time average of the average values, "average-of-average") and the maximum average value,"max-of-average", achieved over the entire simulation run. We will compute the maximum time that any task has been ignored at each time step k, $\max_i\{T_i(k)\}$. We will also compute the time average of this quantity (i.e., the time average of the maximum values, "average-of-max") and the maximum of the maximum values, "max-of-max", achieved over the entire simulation run.

Figure 3.8 shows that different performance measures give different bounds on the delay for which the cooperative case degrades to the performance of noncooperative case. For all the cases, the performance measure increases very slowly for lower values of delays (less or equal to 1 minute), when compared to the higher values of delays, where there is a steady increase in the performance measure. The effect of decreasing density only results in a relatively small increase in the different performance measures; this increase is due to the targets being spread further apart. As expected, the average-of-average and max-of-average result in a higher value of the delay "cross-point," where the cooperative case degrades to the non-cooperative case, than the max-of-max and average-of-max since the former considers performance over average values while the latter considers performance over maximum values (e.g., the max-of-max quantifies worst-case performance). Notice that there are flat regions or valleys for small delay values for some of the performance measures. This arises due to the fact small delays can result in multiple UAVs waiting for the $U(t)$ set at the same time and while doing this they all hold their corresponding $T_i(t)$ values at zero. In our case with 4 UAVs and 6 targets this leads to a performance improvement up to a certain magnitude of delay. If, however, there are many more tasks than UAVs, then generally, increasing the delay will always lead to performance degradation. In summary, communication delays have a prominent influence in the degradation of the cooperative case to the non-cooperative one. The above analysis could be used as design guidelines to decide when it is beneficial to cooperate and when not to do it.

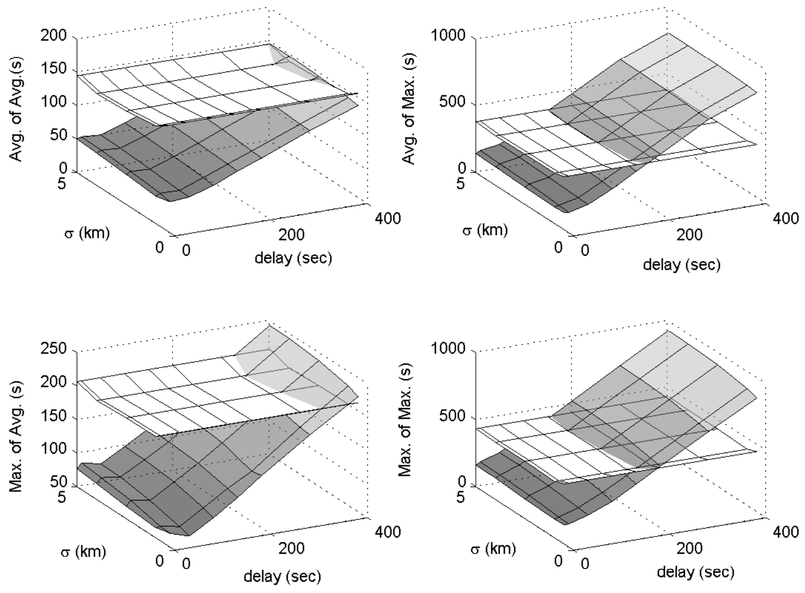


Figure 3.8: Performance measures of Monte Carlo simulation.

CHAPTER 4

ELECTROMECHANICAL ARCADE

We evaluate in this chapter the experimental performance of a set of network-based cooperative scheduling strategies. This experiment was designed to be an inexpensive testbed for networked cooperative scheduling strategies. Below, we describe the apparatus, highlight the challenges it presents, and we introduce and implement two scheduling strategies.

4.1 Experimental Apparatus and Challenges

This experiment is composed of two main devices: guns and targets as shown in Figure 4.1. Each of them is provided with a laser (Radio Shack # 277-1101) and a photodetector (Radio Shack # 276-1657). There are in total eight "pop-up" targets that "appear" (indicated by its laser being on) and "disappear" (laser off) at frequencies that are independent of each other (driven by simple timing circuits that can be adjusted by hand to provide different frequencies). The guns can detect a target appearance if its is pointed directly at it (it has no "peripheral vision") via a photodetector mounted on the gun. The two guns are each mounted on the shaft of a motor (Shinano Kenshi # LA052-040E4N02). Each of these motors has a quadrature encoder that provides the current angular position of the motor. We use a PID controller to point any gun to the target that needs to be shot and these PID loops are well tuned so their performance will not impact our strategies below. The photodetectors of all targets (8 digital inputs), the laser and photodetector of each gun (2 digital inputs and 2 digital outputs), the motor encoder, the output of each PID controller (1 analog output each) are connected to one DS1104 dSPACE card. In fact, analog outputs are connected to an Advanced Motion Control # BE12A6 amplifier that drives each motor.

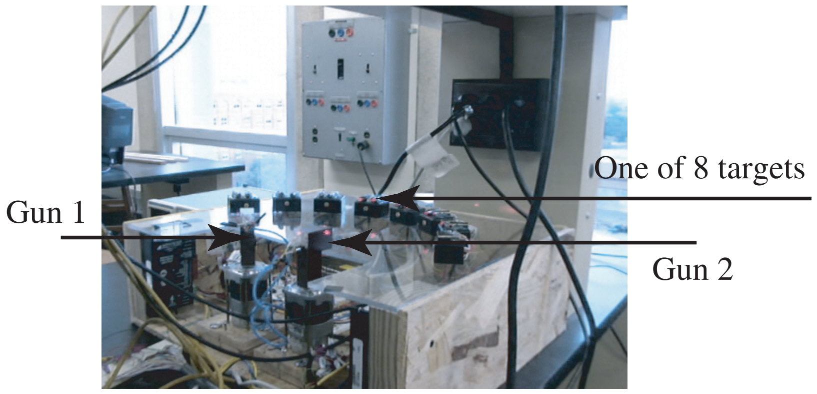


Figure 4.1: Illustration of the electromechanical arcade experiment.

All the lasers located at the targets point to the gun photodetector and if one gun is pointing to a target when it appears, this gun can shoot (turn its laser on) at that target, which triggers the corresponding photodetector of the shot target. When the photodetector of a target is triggered, the gun considers that specific target as "hit" (it gets a point for hitting it) and then the gun will look for the appearance of another target (which one depends on the scheduling strategies introduced below). The analogy with arcade games should be clear.

We assume that the guns do not know any information about the rate of appearance of all targets (but one could invent strategies for estimating appearance sequences); however, the guns do know a priori the position of all targets, and the guns can communicate to each other their decisions about the targets that they are currently processing or pursuing. The challenges for this experiment are as follows:

1. To schedule in real-time a sequence of firings so as to maximize the number of points the team gets. Since target detecting and shooting requires movement of the guns a good schedule will typically minimize the motion of the guns in maximizing point gain. Feedback is required to overcome, for instance, uncertainty about when targets appear (i.e., open-loop precomputed schedules will not be effective).

2. To cooperatively schedule the shooting of the guns in the presence of an imperfect communication network that allows communication between the two guns. While the network could be the internet and a computer could be dedicated to each gun it can also be simulated within one computer. Communication imperfections such as random but bounded delays, bandwidth constraints or message misordering could be considered.

We consider then an "environment" that is highly uncertain (e.g., uncertain target appearance times) *and* where we have imperfect communications that make it difficult for the two guns to coordinate their actions. Due to the presence of so much uncertainty it is generally not possible to accurately predict far into the future, and hence generally not useful to employ optimization approaches to develop long sequences of planned operations either off- or on-line. Finally, note that this cooperative scheduling problem can be thought of as a type of resource allocation strategy.

4.2 Scheduling Strategies and Results

We view this experiment as a 1-D version of the model introduced in Section 3.1.1 since the guns (UAVs) need to process in a cooperative manner the targets (tasks) present in the environment. The sampling time used to run the experiment in real-time is $T_s = 0.001$ seconds. Notice that the targets are aligned in a line, and hence the guns travel across that line in order to cooperatively process the pop-up targets. Using the similarities of this problem with the one defined in Section 3.1.1, we then know that the number of targets is constant and denote the set of targets as $P = \{1, 2, \ldots, 8\}$. The number of guns is constant and we number them and denote the set of guns as $Q = \{1, 2\}$. Each target i is associated with a *prioritized time* at which target i was last shot, $T_i(t), i \in P, t \geq 0$ and we initially let $T_i(0) = 0, i \in P$ so that we act as though initially we had simultaneously shot all the targets, which is clearly physically impossible. Note, however, that this is a good initialization considering the fact that below our scheduling strategies will make decisions about which event to process next based on the sizes of the $T_i(t)$, $i \in P$ (i.e., based on how long they have been ignored).

Note that if no gun was used, then clearly $T_i(t) \to \infty, i \in P, t \to \infty$ since it will never shoot a target. The goal of the scheduling strategy is to try to avoid $T_i(t) \to \infty$ for any $i \in P$ and indeed it will try to keep the $T_i(t)$ values *as small as possible* since this represents that the guns have recently shot each target. Let $p_i, i \in P$ denote the priority of processing target i ("processing" means moving to and shooting at the target when it appears). Asynchronous decision making is accomplished in a way similar to the one described in Section 2.2. In this particular case, the gun $j^u \in Q$ will hold the set $U(t)$. We assume that whenever the gun $\ell \in Q$ where $\ell \neq j^u$ (if $\ell = j^u$ there is no need for a request) finishes shooting a target at time t^f such that $T_{i^*_\ell}(t^f) = 0$ (i.e., the instant the target is processed), it "broadcasts" (simulated here with a delay on a communication link) a request for the set $U(t)$ to the gun j^u. Let the amount of time it takes to broadcast the request and receive $U(t)$ be random, but bounded by a constant $\overline{\delta}_c > 0$. In the time interval $[t^f, t^f + \delta_c]$ that gun ℓ waits for the unattended set, $T_{i^*_\ell}(t^{'}) = 0, t^{'} \in [t^f, t^f + \delta_c]$, which means that the gun ℓ keeps shooting target i^* (i.e., its laser remains on during this time) until it receives the set $U(t)$. This is consistent with our definition of the prioritized time since last shooting since target i^* has already been completely processed at time t^f and it is not being ignored during the interval $[t^f, t^f + \delta_c]$ anymore. The instant that gun ℓ gets $U(t)$, it becomes gun j^u, and makes a decision on what target to process next.

Since the two guns could modify the set $U(t)$ at the same time, we need to use a mutual exclusion algorithm which coordinates the access of both guns to the set $U(t)$ in such a way that this set can be accessed and updated by only one gun at a time. Thus, if gun j^u updated the set $U(t)$, it proceeds to send $U(t)$ to the other gun if it requests it. The gun that receives $U(t)$ updates the set $U(t)$, and passes this set to the other gun if a new request has been made.

Table 4.1 shows the priorities (from left to right viewed from the place where the guns are), and the angular positions (θ) where gun 1 (G1)/gun 2 (G2) will find targets. The p_i values were chosen by observing the illuminating of all targets and assigning higher priorities to the targets that appear more frequently.

	Priority	$\theta(°)$ G1/G2
Target 1	0.9	120/120.5
Target 2	0.85	105/98.25
Target 3	0.8	90.25/79.5
Target 4	0.75	74.6/62
Target 5	0.4	58.8/45.45
Target 6	0.4	41.5/30.25
Target 7	0.5	22.3/15
Target 8	0.2	0/-0.05

Table 4.1: Electromechanical arcade: priorities and angular positions of targets.

The processing time to process target i by gun j for this experiment is equal to $\tau_{i_j} = 1$ msec., which means that it takes for *any* gun one sampling time to detect and shoot any target once the gun is pointing at it. The switching delay, $\delta_{i,j}$, defined in Section 3.1.1 is replaced by what we call the "shooting delay," $\delta^i(t)$, which represents the delay from the target appearances. For a known target i there is some bound for $\delta^i(t)$ on the amount of time that it would take for any gun to wait for the appearance of any target. As a matter of fact, we can obtain that bound ($\overline{\delta}$) from all the appearance periods and duty cycle. Let $T^i, i \in P$ be the target appearance period and D^i denote the percentage of duty cycle of all targets. Hence, $\delta^i(t)$ is bounded by $\overline{\delta} = max_i\{T^i(1 - \frac{D^i}{100})\}, i \in P$.

4.2.1 Networked Cooperative Scheduling: Focus on Target Ignored the Longest

Next, we use a particular scheduling strategy that was defined in [15], which was inspired by the one defined in [1], and for which stability properties have been investigated in Sections 2.5 and 3.3. First, a scheduling strategy that processes the targets that were ignored more than the average one makes choices of which targets to process based on Equation (2.5), where we need to replace the variable x_i by T_i. We will use the "process the target ignored the longest" strategy, Equation (2.4), for this particular case. We have implemented (in the same computer) a routine that simulates a communication network between the two guns where random but bounded communication delays can be generated every time the two guns need to communicate to each other.

Figure 4.2 shows the results when we implement the scheduling strategy introduced in Equation (2.4) and we use a *fixed* communication delay of 10 seconds (we used a fixed delay to facilitate comparison to an approach below). Figure 4.2 shows the $T_i(t), \forall i \in P$ values as well as the targets selected by the two guns during the time the experiment is running. Notice that before the experiment starts both guns are pointing at target 1 and once the experiment starts gun 1 chooses a new target to process next, while gun 2 has to wait 10 seconds for the arrival of the set $U(t)$ coming from gun 1. During this 10 seconds, once gun 2 detects target 1, it keeps processing the same target and this is the reason why $T_1 = 0$ in this interval (recall that any gun keeps processing the target already shot while it is waiting for the unattended set). We can draw some important conclusions from the experimental data. First of all, notice the effect of the priorities values, p_i, on the variable T_i. Second, we can see how a communication delay can affect the performance of the strategy; in particular, observe how the T_i values contained in the set $U(t)$ are ignored during this time. Third, we can see how the guns allocate their processing capabilities in order to shoot (gain more points) as many targets as possible in a cooperative manner (to see this study the sequence of choices).

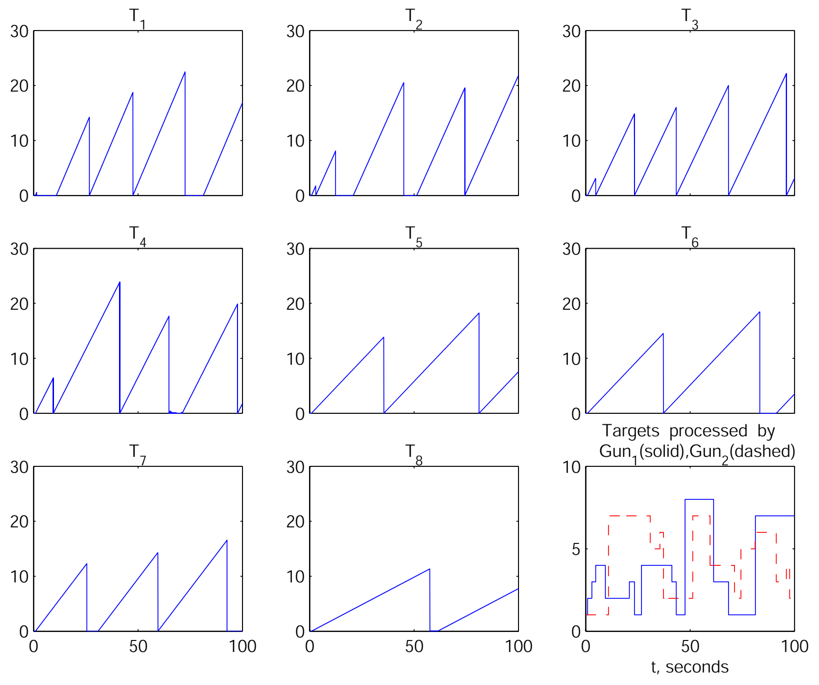


Figure 4.2: Performance of the ignored the longest strategy.

4.2.2 Networked Cooperative Scheduling: Focus on Closest Highest Priority Targets

Here, we use the strategy already defined in Section 3.2.1 that seeks to schedule the next target a gun should process to avoid ignoring high priority targets for too long and yet to minimize travel time to process targets. We view the electromechanical arcade experiment as a type of cooperative scheduling of targets for autonomous vehicles. For this, it is important to pay attention to minimize the travel time between targets and this motivated us to use the strategy defined in Equation (3.3).

Figure 4.3 shows the performance when the process the closest highest priority targets strategy is used. As the previous case, the communication delay is fixed and equal to 10 seconds and the priority values are also the same. All $T_i(t)$ values and the targets selected by the guns can be seen in Figure 4.3. Notice that the number of points in this case (or the times that each target is shot) is at least equal for each T_i to the number of points obtained in Figure 4.2. In addition to this, we can see in Figure 4.3 that the value of the peak for each T_i is less than the peaks seen in Figure 4.2. By trying to minimize travel time we are improving our score. On the other hand, we can see in Figure 4.3 that the closest targets chosen by the guns occur more frequently than those shown in Figure 4.2. In fact, compare the sequence of targets chosen by the guns when $t \in [0, 25]$ in both figures and this will be easily noticed.

Now, how can we better quantify all the facts highlighted in the above paragraph? In order to do that, we will use the performance measures defined in Section 3.4: average-of-average, max-of-average, average-of-max, and max-of-max. One additional performance measure is the number of points (number of targets shot) obtained by the two guns during the experiment. Table 4.2 shows the results obtained from the experiments for the two cases above. The process the closest highest priority targets strategy performs better for all performance measures, including the total number of points obtained. This shows that seeking to minimize travel time allows significant point improvement. It is interesting to note that the strategy did not, however, partition the targets so that each gun always was responsible for the adjacent targets. What emerges (see bottom right plot in Figure 4.3) is a type of balancing of the objectives quantified in Equation (2.5).

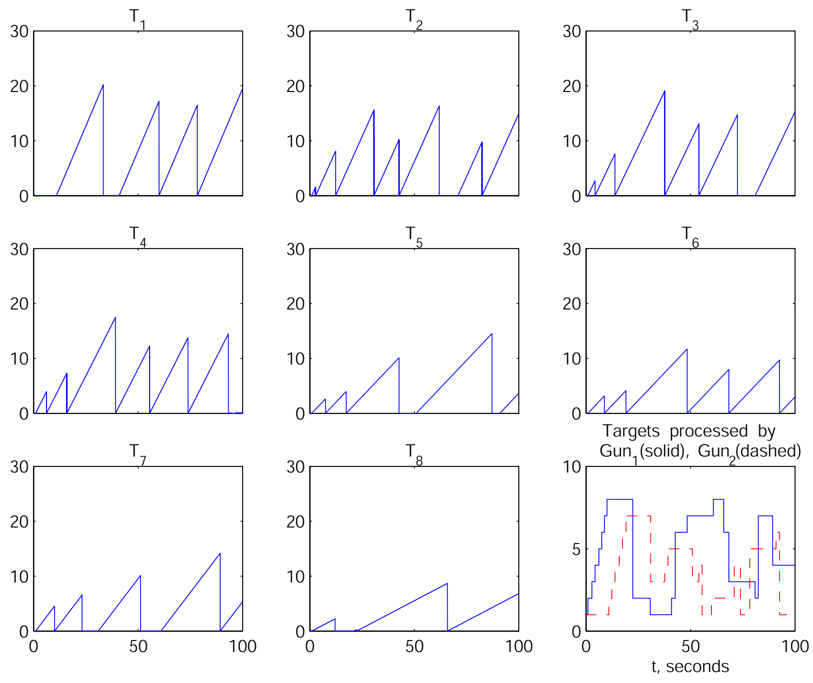


Figure 4.3: Performance of the closest highest priority targets strategy.

	Ignored the Longest	Closest Highest Priority
Average of average	7.2790	5.2229
Maximum of average	11.1548	8.6899
Average of maximum	14.4616	11.5595
Max of max	23.8718	20.2068
Number of points	25	35

Table 4.2: Performance measures.

CHAPTER 5

CONCLUSIONS

In this chapter we will summarize the work done throughout this dissertation and will also introduce some potential future research directions that could contribute to *enhance* the design of network-based cooperative resource allocators.

5.1 Summary and Contributions

In this dissertation we studied the design, modeling, mathematical analysis, simulation, and experimental performance of network-based cooperative resource allocation strategies.

In Chapter 2 we saw how the implementation of several cooperative resource allocation strategies can guarantee the stability of the multiprocessor problem. We focused on one type of cooperative resource allocation problem where via an imperfect communication network with multiple processors can share the workload presented by multiple task types. We defined three asynchronous "cooperative" resource allocation strategies, and showed that they led to the cumulative demand being bounded by a constant, which is in terms of the known parameters. Furthermore, we have obtained some design guidelines for cooperative schedulers via Monte Carlo simulations. First, a group of highly loaded processors that work in a noncooperative way can in some cases produce better performance measures than processors that work cooperatively. Second, when processors are lightly loaded, cooperative strategies can often produce better performance measures than noncooperative ones. Notice that in this dissertation cooperative and noncooperative strategies refer to agents (processors or UAVs) being connected or not connected to a communication network. The work done in [1, 15] provided the inspiration for this result. Note, however, that although the multiple-machine cases in [2], [7], [8], [9], [10], [11] and [3] consider a type of either serial or parallel production where parts can move from one machine to another one, they do not take into account the case where multiple machines can work together to service a set of part types at the same point in the system in the sense that we considered it here (e.g., via two or more processors processing in parallel task types from different buffers). To the best of our knowledge, the closest approach to the one studied in this dissertation can be found in pull production systems [14], where a set of M machines are working in parallel to serve a set of N buffers with a common input stream to a specific queue. This approach is different from the analysis considered in this dissertation from the perspective that first, the policies of pull production systems are based on response to demand whereas in our case, the policies are based on current buffer levels and the production is designed for future demands. Second, the stability analysis in [14] looks for bounds on the queue of backorders or that the queue will eventually be empty and remains in that condition while the stability analysis in this dissertation focuses on ultimate bounds on the buffer levels. Last, the work done in [14] does not consider an imperfect communication network so that there are not random but bounded delays on messages communicated to coordinate activities among the processors. One important fact that stands out for the communication network of our model is that the amount of information passed along the network is relatively small.

In Chapter 3 we have also studied stability (boundedness) for network-based cooperative scheduling strategies that seek to optimize a multiobjective cost function when all UAVs know a priori detailed information about all tasks in a limited area. For it, we have assumed that a UAV needs to process a task several times. We have also shown in Monte Carlo simulations design guidelines for cooperative and noncooperative strategies based on task density and poor performance in the communication channels. The cooperative scheduling strategies proposed in this chapter are extensions of those in [1] and try to tackle some research challenges encountered in cooperative control of UAVs. Hence, to our knowledge, this is the first time that scheduling methods from manufacturing systems have been adapted for use for autonomous vehicles. Note that we considered an environment that is highly uncertain (e.g., desynchronization of UAV's local clock and pop-up targets) *and* where we have imperfect communications so that there are random but bounded delays on messages communicated to coordinate activities among the UAVs. Due to the presence of so much uncertainty it is generally not possible to accurately predict far into the future, and hence generally not useful to employ optimization approaches to develop long sequences of planned coordinated operations either off- or on-line (e.g., like the methods in [28, 29, 30]). Also, note that the scheduling strategies provide a sub-optimal solution that seeks to pay attention to high priority tasks minimizing at the same time the inter-task travel times. However, a "global solution" is not likely to be feasible even if we had full communication capability. This can be due to the resultant complexity of conducting an optimization of the distributed decision-making a priori and on-line. Or, it could be due to the inability to compute and execute globally optimal solutions on-line due to limited available on-board UAV computing power. Moreover, the cooperative scheduling strategies introduced in this dissertation are "cheap" from a computational point of view since the distributed decisions are based on the computation of either a maximum or an average value of both the ignored time of processing tasks and inter-task travel times. Finally, these cooperative scheduling strategies can easily be utilized independent of the number of UAVs. In fact, there is *always* only one computation involved in the decision-making process (e.g., a maximum or average value) and the number of the elements considered in the computation decreases (increases) when the number of the UAVs (tasks, respectively) increases; hence, the real-time implementation of these cooperative scheduling strategies do not expand *exponentially* in the number of necessary computations when the number of UAVs or tasks increases. As in the multiprocessor problem, we paid attention to the significant problem related to link bandwidth found in imperfect communications; hence, coordinated decisions among UAVs avoid the need for passing extraordinary amounts of information.

Now we want to pose the following question: is there any connection between Chapters 2 and 3? If so, how are they connected? We think that there is an important connection between the two and that connection could be useful for manufacturing systems. For instance, suppose that there is a group of robots that need to work in a cooperative manner in order to assemble/disassemble a set of part-types arriving at a set of buffers. If the buffers are spread over an entire area and the robots need to travel to the buffers to start processing parts from them, then it is clear that we could use the cooperative scheduling strategies introduced in Chapter 3 to maximize throughput and minimize robots' travel time. Thus, it would be interesting to explore these ideas in the collective robotics area.

In Chapter 4 we evaluated and compared the performance of scheduling strategies defined in [15] and in Chapter 3 in the electromechanical arcade experiment. We implemented algorithms to cooperatively schedule in real-time the shooting of the guns in the presence of an imperfect communication network that allows communication between the guns. It was shown how in one experiment multi-objective scheduling strategies achieved a better performance compared to scheduling strategies that seek to optimize just one objective. We corroborated that the model defined in Chapter 3 possesses similarities with the electromechanical arcade experiment. This experiment establishes a basis for future research in networked cooperative scheduling strategies for graduate studies. To our knowledge, it is the first time such an experiment has been implemented and taking an electromechanical arcade viewpoint. It is clear that there are many possible extensions of this experiment.

5.2 Future Research Directions

There are another research topics that we think could be considered in the future for expanding the modeling and analysis of stability properties of cooperative resource allocation strategies.

First, we mention the research opportunities that could be explored in the multi-processor problem.

1. **Combination of cooperative and noncooperative strategies:** It would be interesting to study the possibility of obtaining lower bounds on the buffer

levels when both resource allocation strategies, cooperative and noncooperative, are combined as it was mentioned at the end of Section 2.7.

2. **Synchronous resource allocation strategies:** As we pointed out in Section 2.3, it would be interesting to explore whether there exists a condition where idling could produce lower bounds for synchronous strategies rather than for asynchronous ones based on the previous work done in [5] for the $M = 1$ case.

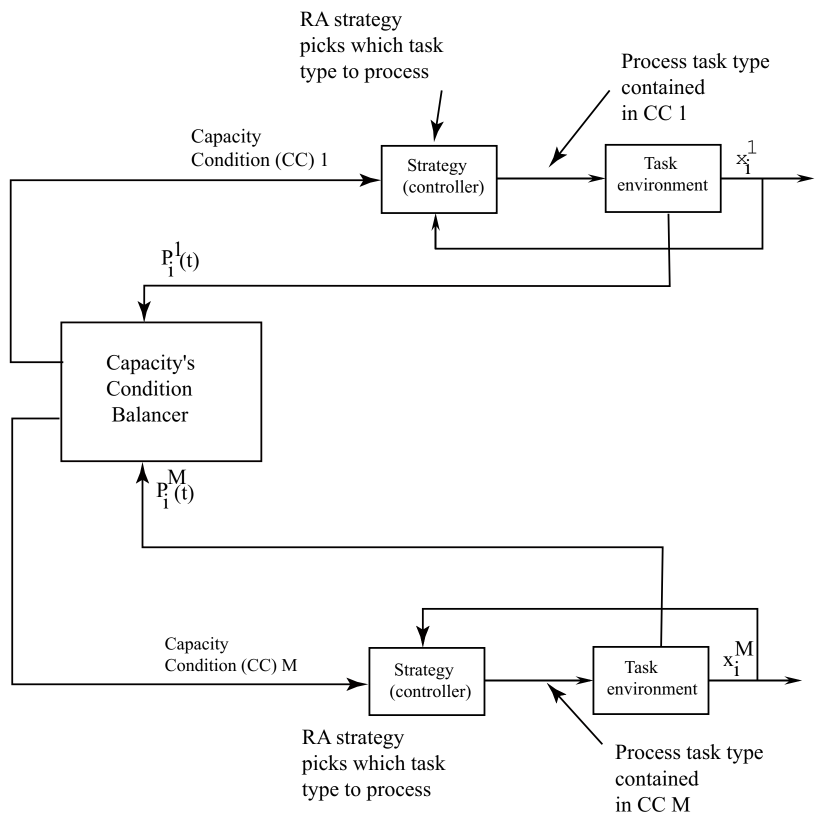
3. **Implication of the loss of a processor on the stability of the cooperative resource allocation strategy:** In Chapter 2, we have provided conditions for the stability of cooperative resource allocation strategies for the scenario where there are M processors coping with a set of N task types. Both variables are assumed to be constant when we studied the stability properties of the strategy. A more realistic problem is to deal with the possibility that M and N can be time-varying. Clearly, according to Equation (2.1), one problem that could arise is that the capacity condition would not be satisfied anymore as we discuss next.

4. **Violation of the capacity condition equation:** If we consider the scenario where both the number of task types (N) and the processors (M) are time-varying, then there exists the possibility that the capacity condition is not satisfied (see Equation (2.1)). Since the parameters a_i are fixed, the only available option that we have is to modify the arrival rates, p_i, assigned to each task type in order to force the capacity condition to be satisfied. The goal would be to design an off-line algorithm that will define the new arrival rates values

without deteriorating the past performance measures. This idea of avoiding a violation in the capacity condition is taken from [17] but using a different approach. Also, ideas from [11] could be considered as well.

5. **Hierarchical distributed resource allocation strategy:** Consider the case where the number of task types is time-varying. For it, there is an upper level controller trying to balance the workload among the processors. We are interested in studying the scenario where M processors are processing several task types in a given environment. The environment has task types that can arrive dynamically at certain time, i.e., task type i can arrive at a rate p_i up to certain time t_1 and after that time that task type is not available to the processors anymore. Furthermore, the arrival of the task types can be sensed through a device available in every processor. Thus, processors will receive a time-varying number of task types $N(t)$. Note that $N(t)$ and $N(t+t_1), t_1 > 0$ can have the same value either if processors do not receive any new task type or if the number of new processed task types is equal to the number of old unprocessed task types between times t and $t+t_1$. Moreover, if $N(t) < N(t+t_1)$ then processors found new task types, while if $N(t) > N(t+t_1)$ then one or more old task types become out of the sensing range of any processor between times t and $t+t_1$.

   We envision the design of a hierarchical cooperative resource allocation strategy in order to deal with this kind of problem. We show in Figure 5.1 a two-level controller as a possible solution. The higher level controller will be in charge of balancing the workload to be given to each processor while the lower level controller will be applying the resource allocation strategies. Suppose that each



RA strategy picks which task type to process
Process task type contained in CC 1
Capacity Condition (CC) 1
Strategy (controller)
Task environment
x_i^1
$P_i^1(t)$
Capacity's Condition Balancer
$P_i^M(t)$
Capacity Condition (CC) M
Strategy (controller)
Task environment
x_i^M
RA strategy picks which task type to process
Process task type contained in CC M

Figure 5.1: Hierarchical cooperative resource allocation (RA) viewed as a controller.

processor's sensor is provided with a limited sensing range, hence the lower level strategy is not necessary the one defined in Section 2. This is due to the fact that is possible that $P_1(t) \cap P_2(t) \cap \ldots \cap P_M(t) = \emptyset$, where $P_i(t)$ is the set composed by the task types that are in the sensing range of processor i. Moreover, we could face the case where $|P_1(t) \cap P_2(t)| = |N_1(t)| = |N_2(t)|$ (the number of task types that can be sensed by processor 1 and 2 are the same), in the presence of this case both processors can process all task types since their respective capacity conditions could be balanced. Another case that could arise is when $|P_1(t) \cap P_2(t)| = D < |N_1(t)|$ and $|P_1(t) \cap P_2(t)| = D < |N_2(t)|$ (the number of task types that are in the common sensing range for processors 1 and 2 is less than the total number that each can sense separately), then depending on the result of the balancing of the capacity condition, one processor could process the set $P_1(t)$ while the other one could process the set $P_2(t) - P_1(t) \cap P_2(t)$ or vice versa. Another possibility is that one processor could process the set $P_1(t)$ and the other one the set $P_2(t)$. In other words, both processors share the task types contained in the set $P_1(t) \cap P_2(t)$ but again this is going to depend on the commands from the higher level controller.

6. **Pre-assignment of task types to processors in order to minimize the bounds:** We have mentioned above how hierarchical cooperative resource allocation strategies could be used to balance the workload, the tasks' processing in this case, among all processors and that we are interested in studying the stability of such strategies. However, we could use the same hierarchical scheme, but the role of the upper level controller would be this time to pre-assign groups of task types to the processors in such a way that the ultimate bound be minimized. Note that in the previous workload balancing case, the upper level controller tries to give the same amount of work to each processor while in this case, each processor could have a different amount of work.

7. **Cooperative resource allocation strategies for the foraging problem:** We have studied in Chapter 2 the stability properties of a general framework where all processors have a constant processing rate. However, there are cases where this is not the case and foraging theory is one of them. Consider the case shown in Figure 5.2, where we can see that once the processor (organism) starts

processing a task type the buffer level decreases exponentially. This behavior has been proven to be the case for organisms searching for nutrient sources and it would be interesting to explore these ideas and to study the stability for this case.

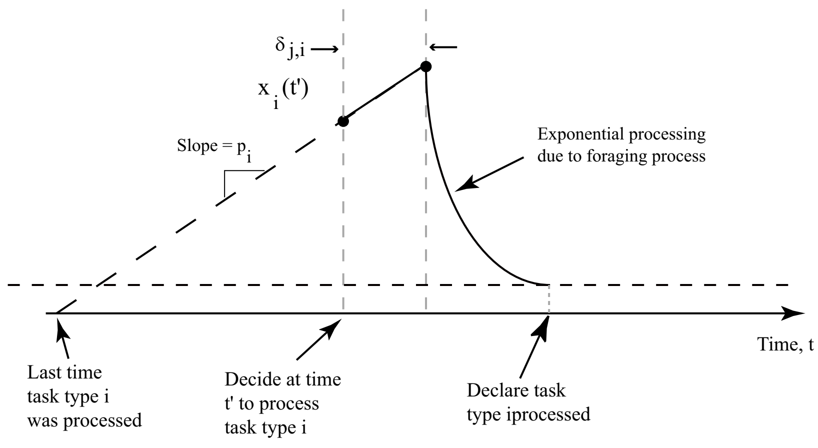


Figure 5.2: Illustration of the task type processing caused by the foraging problem.

Next, we list some research opportunities for the UAV problem.

1. **Maximizing UAVs throughput:** Figure 5.3 shows a great opportunity to improve the work done in Chapter 3 by deriving stability properties for the case where the number of tasks is time-variant and when UAVs assign different priorities to the processing of tasks (i.e., $p_i(t), \forall i \in P$) during the mission time in a limited area. The idea behind these new considerations is to quantify *how much* work the group of UAVs can execute together. In other words, we would like to quantify UAVs task throughput and derive a capacity condition as in [1] for that scenario. This approach will add flexibility to the problem

in the sense that the group of UAVs can receive tasks to process from several sources: home base (e.g., humans demanding the processing of specific tasks), global sensor, and UAVs on-board sensors. The motivation for having to assign different priority values for the same task is driven by the fact that we would like to consider an even more realistic UAV problem. For instance, consider the scenario where UAVs need to perform search, classify, attack, and verify for the same task. Initially, one UAV could have assigned a high priority value to a certain task that is considered of extreme importance for the mission, however, when this or other UAV performs the classification, it realizes that the task is not what it was initially thought and that UAV then needs to assign a new priority value to that task. Thus, we would need to define new variables in our model as the priority of searching, $p_i^s(t)$, the priority of classifying, $p_i^c(t)$, the priority of attacking, $p_i^a(t)$, and the priority of verifying, $p_i^v(t)$, for task $i, \forall i \in P$. Note, however, that if the number of tasks is fixed as it is in Chapter 3, the analysis performed there can still hold for bounded $p_i(t)$ values. For instance, we would have to compute $p_i = max_{t \geq 0}\{p_i^s(t), p_i^c(t), p_i^a(t), p_i^v(t)\}, \forall i \in P$ and these p_i values would be the ones that we could use for the bound derived in Section 3.3.

2. **Consideration of target/threat characteristics:** Recall that in our model we considered tasks in the environment, which is a general terminology tied up with the fact that our analysis considers a fixed number of tasks during the mission. Since in a more realistic problem there are targets that are not threats and vice versa, there are some that are both targets and threats, and there are even false targets, it would not be a problem to add these characteristics to

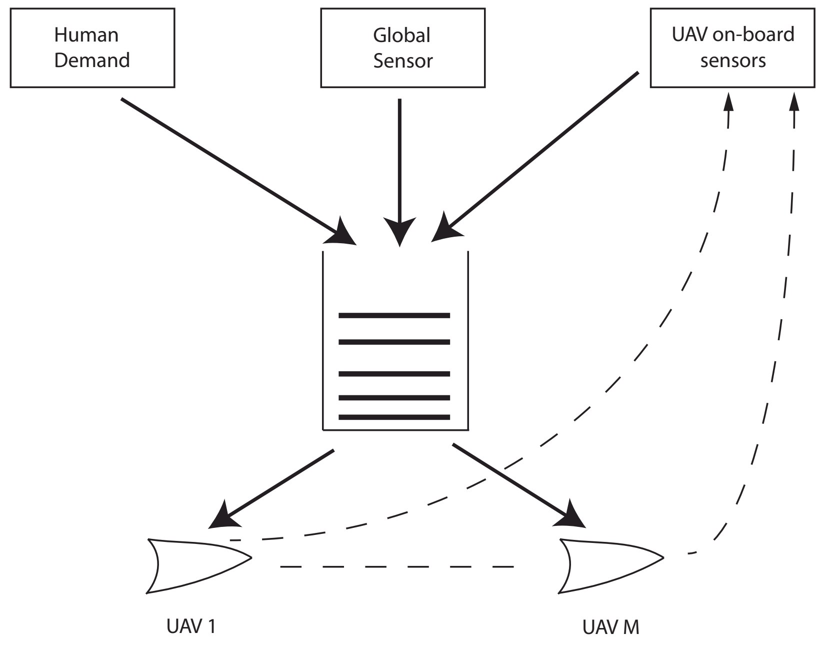


Figure 5.3: Time-variant number of tasks processed by M UAVs.

our model provided that the above problem is solved. Moreover, it would be interesting to study scenarios that have mobile targets.

3. **Controller design trade-offs:** In Section 3.2.1 we mentioned how it is possible to view the p_i values as controller design parameters that can be tuned to improve performance. Since Equation (3.2) seeks to optimize two objectives, putting more emphasis in one variable may result in degradation of the performance metrics; hence, it would be useful to study optimization methods (off-line or on-line ones) that tune the priorities in order to achieve a better performance during the mission.

4. **Exploitation of information:** Here, the idea is to use the priority values to take advantage of the a priori information that may be available before the

UAVs are deployed. For instance, a priori information could be used to establish search-maps across the entire region of interest by assigning priority values to tasks based on the likelihood of location of tasks.

5. **Combination of cooperative and noncooperative strategies:** As in the multiprocessor case, it could be worthy to study the possibility of obtaining lower bounds on the ignored time of tasks when both resource allocation strategies, cooperative and noncooperative, are combined.

6. **Multiobjective resource allocation strategies:** There are some applications where there is the need to define a total cost function that takes into account the sum of several cost functions. Consider for example the UAV problem and suppose that there is a way to compute and sum up the cost of the energy that a UAV must expend to move to a certain place, to avoid the danger/threat of some tasks, the time since last processing of each task. Then, the resources (UAVs) are going to be dynamically allocated to the processing of the tasks depending on the computation of the total cost function. Chapter 3 seeks to optimize the processing of high priority tasks and to minimize gas expenditure. However, we think that the cost of avoiding dangers/threats could implicitly be represented in the term $\frac{d(x_v^j(t),x^i)}{v}$ since this is the time that takes to UAV j to travel from its current location and orientation to process task i at x^i. Since this travel time needs to be computed by UAV j for any task $i \in U_j^a(t)$, if there is a danger/threat between UAV current position and the course that this UAV needs to follow to get where task $\ell \in U_j^a(t)$ is, then this will cause the UAV to modify its course to avoid passing close to the danger/threat, which

could increase the travel time to that task ℓ. Therefore, the cost of that task ℓ will decrease according to Equation (3.2) which will decrease the possibility of this task being chosen by UAV j at time t.

Finally, most of the research directions proposed above could be implemented in the experimental testbed designed for network cooperative scheduling strategies. Such implementations could provide more insights for a better understanding of theoretical developments. They may also provide new research challenges.

BIBLIOGRAPHY

[1] J. Perkins and P. Kumar, "Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems," *IEEE Transaction on Automatic Control*, vol. 34, pp. 139–148, February 1989.

[2] K. Passino and K. Burgess, *Stability Analysis of Discrete Event Systems.* John Wiley and Sons, Inc., NY, 1998.

[3] K. Burgess and K. Passino, "Path-clearing policies for flexible manufacturing systems," *IEEE Transactions on Automatic Control*, vol. 44, pp. 573–578, March 1999.

[4] K. Burgess and K. Passino, "Stable scheduling policies for flexible manufacturing systems," *IEEE Transactions on Automatic Control*, vol. 42, pp. 420–425, March 1997.

[5] C. Chase and P. J. Ramadge, "On real-time scheduling policies for flexible manufacturing systems," *IEEE Transactions on Automatic Control*, vol. 37, pp. 491–496, April 1992.

[6] S. Lou, S. Sethi, and G. Sorger, "Analysis of a class of real-time multiproduct lot scheduling policies," *IEEE Transactions on Automatic Control*, vol. 36, pp. 243–248, February 1991.

[7] P. Kumar and T. J. Seidman, "Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems," *IEEE Transactions on Automatic Control*, vol. 35, pp. 289–298, March 1990.

[8] P. Kumar and S. Lu, "Distributed scheduling based on due dates and buffer priorities," *IEEE Transactions on Automatic Control*, vol. 36, pp. 1406–1416, December 1991.

[9] Z. Tang and L. Shi, "Note on "Distributed scheduling based on due dates and buffer priorities" by S.H. Lu and P.R. Kumar," *IEEE Transactions on Information Theory*, vol. 37, pp. 1661–1662, October 1992.

[10] C. Humes, Jr., "A regulator stabilization technique: Kumar-Seidman revisited," *IEEE Transactions on Automatic Control*, vol. 39, pp. 191–196, January 1994.

[11] J. R. Perkins, C. Humes, Jr., and P. Kumar, "Distributed scheduling of flexible manufacturing systems: Stability and performance," *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 133–141, April 1994.

[12] R. L. Cruz, "A calculus for network delay, Part I: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, pp. 114–131, January 1991.

[13] Y. Wang, J. Perkins, and A. Khurana, "Optimal resource allocation in new product development projects: A control-theoretic approach," *IEEE Transaction on Automatic Control*, vol. 47, pp. 1267–1276, August 2002.

[14] T. Seidman and L. Holloway, "Stability of pull production control methods for systems with significant setups," *IEEE Transactions on Automatic Control*, vol. 47, pp. 1637–1647, October 2002.

[15] K. Passino, *Biomimicry for Optimization, Control, and Automation.* Springer-Verlag, London, 2004.

[16] K. Passino, "Design of stable dynamic scan scheduling policies," tech. rep., DARPA Autonomous Negotiating Teams Project Report, 2001.

[17] Y. Xun and M. Kokar, "Control based sensor management for a detection scenario," tech. rep., DARPA Autonomous Negotiating Teams Project Report, 2000.

[18] T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches.* MIT Press, 1988.

[19] C. Cassandras, L. Dai, and C. Panayiotou, "Ordinal optimization for deterministic and stochastic discrete resource allocation," in *Proceedings of the IEEE Conf. on Decision and Control*, pp. 662–667, December 1997.

[20] L. Dai, C. Cassandras, and C. Panayiotou, "On the convergence rate of ordinal optimization for a class of stochastic discrete resource allocation," *IEEE Transactions on Automatic Control*, vol. 45, pp. 588–591, March 2000.

[21] K. Gokbayrak and C. Cassandras, "Stochastic discrete optimization using a surrogate problem methodology," in *Proceedings of the IEEE Conf. on Decision and Control*, pp. 1779–1784, December 1999.

[22] C. Cassandras and R. Yu, "A surrogate problem approach for lot size optimization in manufacturing systems," in *Proceedings of the American Control Conference*, pp. 3279–3283, June 2000.

[23] L. Shi and C. Chen, "A new algorithm for stochastic discrete resource allocation optimization," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 10, pp. 271–294, 2000.

[24] L. Shi and S. Men, "Optimal buffer allocation in production lines," tech. rep., University of Wisconsin-Madison.

[25] L. Abeni, L. Palopoli, and G. Buttazzo, "On adaptive control techniques in real-time resource allocation," *IEEE Transactions on Real-Time Systems*, pp. 129–136, June 2000.

[26] B. Ravindran, P. Kachroo, and T. Hegazy, "Adaptive resource management in asynchronous real-time distributed systems using feedback control functions," *5th International Symposium on Autonomous Decentralized Systems*, pp. 39–46, March 2001.

[27] C. McKie, E. Rojas, N. Quintero, J. Chacon, and N. Perozo, "Economic benefits from automated optimization of high pressure gas usage in an oil production system," *Society of Petroleum Engineers, SPE67187*, March 2001.

[28] C. G. Cassandras and W. Li, "A receding horizon approach for solving some cooperative control problems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, (Las Vegas, Nevada), pp. 3760–3765, Dec. 2002.

[29] D. A. Castanon and C. G. Cassandras, "Cooperative mission control for unmanned air vehicles," in *Proceedings of the AFOSR Workshop on Dynamic Systems and Control*, (Pasadena, California), pp. 57–60, August 2002.

[30] J. Bellingham, A. Richards, and J. How, "Receding horizon control of autonomous aerial vehicles," in *Proceedings of the American Control Conference*, (Anchorage, Alaska), pp. 3741–3746, May 2002.

[31] P. R. Chandler, M. Pachter, and S. Rasmussen, "UAV cooperative control," in *Proceedings of the American Control Conference*, (Arlington, Virginia), pp. 50–55, June 2001.

[32] R. Beard, T. McLain, and M. Goodrich, "Coordinated target assignment and intercept for unmanned air vehicles," in *Proceedings of IEEE International Conference on Robotics and Automation*, (Washington, DC), pp. 2581–2586, May 2002.

[33] P. Chandler *et al.*, "Complexity in UAV cooperative control," in *Proceedings of the American Control Conference*, (Anchorage, Alaska), pp. 1831–1836, May 2002.

[34] K. M. Passino, M. Polycarpou, *et al.*, "Cooperative control for autonomous air vehicles," in *Cooperative Control and Optimization* (R. Murphey and P. M. Pardalos, eds.), vol. 66, pp. 233–271, Kluwer Academic Publishers, 2002.

[35] M. Baum and K. Passino, "A search-theoretic approach to cooperative control for uninhabited air vehicles," in *AIAA GNC Conference, Monterrey, CA*, Aug. 2001.

[36] J. Hespanha and H. Kizilocak, "Efficient computation of dynamic probabilistic maps," in *Proceedings of the 10th Mediterranean Conference on Control and Automation*, (Lisbon, Portugal), July 2002.

[37] J. Hespanha, H. Kizilocak, and Y. Ateskan, "Probabilistic map building for aircraft-tracking radars," in *Proceedings of the American Control Conference*, (Arlington, Virginia), pp. 4381–4386, June 2001.

[38] A. Mahajan, J. Ko, and R. Sengupta, "Distributed probabilistic map service," in *Proceedings of the 41st IEEE Conference on Decision and Control*, (Las Vegas, Nevada), pp. 130–135, Dec. 2002.

[39] S. Ganapathy and K. Passino, "Agreement strategies for cooperative control of uninhabited autonomous vehicles," (Denver, Colorado), pp. 1026–1031, June 2003. Proceedings of the 2003 American Control Conference.

[40] M. Jun, A. I. Chaudhly, and R. D'Andrea, "The navigation of autonomous vehicles in uncertain dynamic environments: A case study," in *Proceedings of the 41st IEEE Conference on Decision and Control*, (Las Vegas, Nevada), pp. 3770–3775, Dec. 2002.

[41] S. Ganapathy and K. Passino, "Distributed agreement strategies for cooperative control: Modeling and scalability analysis," in *Proceedings of the Conference on cooperative control and optimization*, (Gainsville, FL), pp. 127–147, Dec. 2002.

[42] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, MA, 1997.

[43] R. Balboni, A. Virgillito, and R. Petrassi, "A distributed mutual exclusion algorithm for mobile ad-hoc networks," in *Proceedings of International Symposium on Computers and Communications*, pp. 539 – 544, July 2002.