Knowledge-Based Real-Time Control: A Parallel Processing Perspective

D. D. Sharma and N. S. Sridharan
FMC Corporation
Artificial Intelligence Center
1205 Coleman Avenue, Box 580,
Santa Clara, CA 95052

Abstract

Knowledge-based real-time control problems can be usefully viewed as dynamic resource allocation problems. Analysis of various real-time applications and real-time AI models reveals that real-time control problems require the problem solving capability of knowledge intensive methods coupled with the control mechanisms of operating systems. Moreover, there is an opportunity and need to exploit parallelism inherent in real-time control problems. We describe a user-programmable concurrent computation model which blends the capabilities of knowledge-based systems and operating systems. We also propose a novel set of performance measures useful for real-time AI systems.

1. Introduction

Knowledge-based real-time problems occur in diverse areas such as process control, autonomous land vehicles, and operation of complex systems (e.g., situation assessment, tactical planning, path planning, mission control, decision aiding, and risk control). AI solutions to these problems require blending knowledge-intensive approaches with control mechanisms of operating systems such as concurrent and coordinated performance of multiple tasks, quick reaction to high priority tasks, adaptation to variable rate of incoming data, and the ability to suspend and resume tasks. Existing parallel computation models in Lisp are mostly oriented toward achieving greater speed-up at the algorithmic level. RT problems pose different requirements. Thus the focus of our research is on developing new architectural models. In this paper we describe problem areas, extract problem requirements, describe a model for knowledge-based concurrent computation, and indicate a different set of performance measures useful for RT AI systems.

2. Real-Time Control Problem

Real-time problems involve performing tasks in a dynamic environment under time stress. The control problem requires managing available resources while accomplishing these tasks. In these problems time is both a resource and constraint. Time is resource in the sense that we can allocate different amounts of time to various tasks and also reclaim time by controlling other resources. For example, consider the case of simultaneously exploring several alternatives where the quality of solution is proportional to the time spent in finding the solution. In this case if we have more time then instead of accepting the earliest solution we can afford to wait and find a better quality solution. Another instance is a combat situation where an agent encountering threat may only have a few seconds to protect itself. If it has appropriate resources (extra fuel, other friendly agents able to help, opportunity) it can attempt to make an evasive move or distract the attacker and thereby gain time. The amount of available time cannot be easily quantified and often depends upon the context defined by the activities of other agents and status of other resources.

We view real-time control as a problem of managing resources in performing specified tasks. This involves both reasoning about how to allocate resources and reasoning about what tasks need to be performed. These two problems involve several difficult reasoning and control issues. Solving a problem requires allocation of various computational resources. When resource requirements are predictable resource allocation is algorithmic, fixed and declared in the program, and is expected to be consistent with the available resources. However, in real-time control problems unpredictable external influences tend to disturb the existing balance of resource allocation, and it is required to re-allocate resources to satisfy the needs of the external demands within the constraints of available resources while satisfying the specified goals. *We consider this problem of dynamic resource allocation as a basic problem of real-time control.*

3. Information Processing in Real-Time Control Problems

Our working model of real-time control is as follows. The operation of real-time system involves performing a stream of tasks. The tasks are generated by the demands of the external environment or by the activities of the real-time

system itself. Thus information processing in RT control problems involves describing and controlling the *flow* of tasks through the various stages of concurrent computation. Most real-time AI problems share these characteristics.

As an example let us consider the blackboard models. Blackboard architectures have been used for various real-time applications such as real-time process control [D'Amborsio 87] sensor data interpretation, and speech recognition [Nii 86]. In this paper we will use a specific blackboard model called HCVM (Heuristic Control Virtual Machine). HCVM is an object-oriented blackboard system developed by FMC and Teknowledge and used in knowledge-intensive real-time problem solving [D'Ambrosio 87]. Figure 1 shows a concurrent view of the information flow in HCVM. The top level control of HCVM consists of several control modules. The communications manager (CM) handles input/output interactions with the external world. CM reads data from the external world, generates data handler (DH) tasks to *update data*, and sends these tasks to a buffer called *DH-Queue*. The use of the buffer enables CM to run asynchronously with other modules. Module *Data Handler Execution* (DHE) reads a DH task from the buffer, executes the tasks, and updates appropriate data in the data space, i.e., the blackboard. A data handler can do routine processing such as data validation and trend calculation. HCVM supports several knowledge source modules called knowledge handlers (KH). A KH has a trigger condition which is evaluated against changes in the data space to determine if the KH should be scheduled for execution. A module called *Trigger Condition Evaluation* (TCE) evaluates the trigger conditions of all the KHs and puts the triggered KHs on the *Agenda*. The module *Agenda Manager* prioritizes and schedules KHs for execution. In this model the top priority KH gets executed. Execution of a KH can generate other tasks, called *Task Handlers*, which are put in a buffer for further execution. Encoding a KH as a sequence of THs allows interleaved execution of several KHs. The execution of knowledge handlers or task handlers can result in data update tasks which are sent to DH-Queue or output to CM.

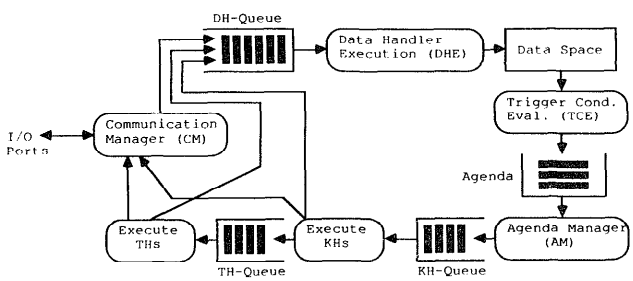


FIGURE 1. Information Flow in HCVM

4. Computational Requirements of Real-Time Control

Based on an analysis of the HCVM system and other real-time control applications we have arrived at the following requirements. These requirements have guided the development of a concurrent real-time model described in next section.

o **Concurrent Execution of Tasks:** In real-time problems there is both an opportunity and need for concurrent execution of tasks (e.g., the top level control tasks and the execution of KHs in HCVM). With suitable parallel hardware support this can lead to the desired performance in terms of speed and quality of solution.

o **Knowledge-Based Task Scheduling:** Tasks are scheduled for execution either because they are required by an active plan or they are needed due to the changes in the environment.

o **Knowledge-Based Task Prioritization:** Certain tasks have a higher priority over other, e.g., tasks pertaining to the survivability and safety of a system. The priority of a task depends upon several factors such as the inputs from the environment, the tasks already under execution, and the current goals of the system. Task priorities are computed dynamically by the Agenda Manager.

o **Task Interruption/Resumption:** Task interruption is required to shift resources from a low priority task to a higher priority task in order to achieve the desired level of responsiveness. Suspended tasks may have to be resumed after high priority tasks have been executed. The decision to suspend, resume, or abort the current task is a knowledge-based decision and amenable to parallel processing.

o **Communication Between the Tasks:** Real-time control needs cooperative problem solving, therefore, individual tasks should be able communicate to share data in order to achieve goals.

o **Resource Constraint/Contention:** Real-time applications often have finite amount of non-renewable resources, for example, to complete a certain mission a vehicle has fixed amount of time and fuel. Often the tasks may compete for the same resources. Thus a key problem is to determine which constraints are rigid and which can be relaxed.

o **Knowledge-Based Resource Management:** Given a certain set of resource constraints and various tasks competing for them a key problem is to determine how the resources should be shifted between the tasks.

o **Risk Reduction/Graceful Degradation:** The system should be able to handle the following two situations: (1) reduction in capability due to partial system failures; and (2) demands exceeding

the designed capability. In such situations the real-time systems should still provide the best performance it can.

o Concurrent Exploration of Alternatives: The constraints of available resources and the need to produce an acceptable solution under time stress requires that several alternative solutions be explored in parallel. This feature is an important component of risk-reduction strategies.

o Computational Resource Allocation: Computing systems have finite resources (memory and computational power). Real-time control systems need capability to allocate processors flexibly to the tasks and be able to change the allocation dynamically to address the needs of the problem.

5. QP-Net: A Computational Model

The computational model proposed here provides a mechanism to generate *tasks* and to *allocate* the tasks to processors in a flexible and machine independent manner. Real-time computation is described as a flow of tasks in a network of *task queues* and *task processors*, along with various control strategies for resource allocation.

5.1 The QP-Net Model

The basic model consists of three elements: tasks, task queues, and servers which are task processors. The real-time problem is modeled as a network of task queues and servers. Typically, servers read a task from a task queue, process the task, and if appropriate put new tasks on the same queue or another queue.

The model supports multiple task queues for tasks with different priorities. The policies for scheduling and prioritizing tasks are defined in the context of these prioritized task queues. This model of task queues is expressed by defining a *q-manager* object shown in Figure 2a. A q-manager has prioritized task queues in local memory and methods defined for returning *next-task* for execution and *enqueue-task* for prioritizing and storing incoming task in the proper task queue. The q-manager also has probes to measure various performance parameters such as the number of tasks waiting execution, the rate at which tasks are being removed. Information from these probes is used as parameters in control strategies. The control strategies are local to a q-manager and may consist of prioritization and scheduling policies, resource allocation mechanisms, and synchronization mechanisms.

A server is a process (shown in Figure 2b) which requests a task from a specified q-manager. The q-manager served by a server can be determined in one of the following ways:

- a single q-manager is specified thus the server is dedicated;
- select a q-manager from an ordered list of q-managers. For example, the server shown in Figure 2b approaches q-managers in the following order Q1, Q2, ..., Qn until a q-manager returns a task. Thus as long as one of the q-managers on the list has a task to be performed the server will do useful work.

In option 1 it is possible to have more than one server for a q-manager. If the result of the execution of a task is another task to be evaluated then the server sends this task to an appropriate q-manager.

5.2 Characteristics of QP-Net Model

The proposed QP-Net model is conceptually simple and can support various requirements of building real-time knowledge based systems. Here we discuss a few of the characteristics.

o QP-Net Can Support Real-Time Requirements

As discussed in the requirements real-time control problems require features and power of operating systems (parallel processing, scheduling, resource allocation), and the capability to solve knowledge intensive problems. QP-Net combines operating system features with object oriented programming and can support a concurrent blackboard model. Thus QP-Net is good for developing knowledge based systems and real-time applications.

o Flexible Allocation of Processors

As an example let us again consider the blackboard architecture shown in Figure 1. Using the QP-Net model we can design three types of

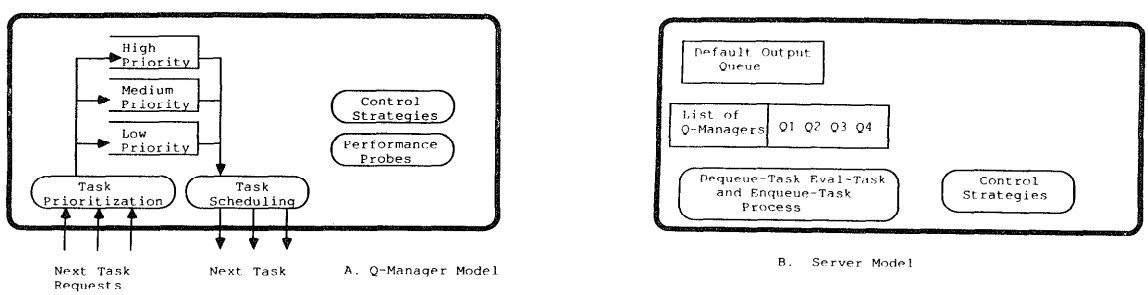


FIGURE 2. Elements of QP-Net

multiprocessor architectures depending upon the allocation of q-managers and servers: (1) Static Allocation; (2) Dynamic Allocation; and (3) Hybrid Allocation. The *Static Allocation* architecture (Figure 3a) has a fixed number of servers dedicated to various concurrent tasks. By experimenting with the allocation of different number of servers it is possible to fine tune the architecture for the desired performance. Such an architecture suffers from the problems of load unbalance and inability to dynamically reallocate resources to meet the demands of the problem. *Dynamic allocation* (Figure 3b) is a response to the load balancing problem. Dynamic allocation also enables designs that are independent of specific hardware configuration, i.e., number of processors. In dynamic allocation free servers are assigned to q-managers. This is same as the *futures* model of MultiLisp [Halstead 86]. The dynamic allocation has a drawback: it is difficult to guarantee the availability of resources when needed or dedicate a fixed amount of resources. *Hybrid allocation* (Figure 3c) offers a blend of the good characteristics of the static and dynamic allocation. In a hybrid allocation scheme certain q-managers (e.g., highly critical tasks) are pre-assigned fixed number of servers and others are assigned servers dynamically. This flexible design can also enable re-assigning some of the servers from the dynamically allocatable cluster to the high priority q-managers.

o Resource Allocation

The resource allocation problem can be viewed as: (1) balancing processor load, and (2) controlling the size of task queues. Strategies to balance processor load can be quite expensive because they require monitoring processor utilization and then migrating tasks or objects to the under-utilized processor. Implementing optimal migration strategies in itself can be very expensive.

In QP-Net model load balancing is easily achieved by using a dynamic allocation scheme along with an ordered list of q-managers. The load balancing thus achieved does not guarantee that the processors are busy doing useful work. In other models (such as futures and parallel object oriented models) there is no easy way to detect busy waiting during the processing. However, in QP-Net model the effect of busy waiting is quickly detected in terms of increased congestion at some of the q-managers. This problem can be solved by changing the ordered list of q-managers of a certain number of processors. Another approach is to view servers as *logical processors* and the processor allocation is done dynamically as shown in Figure 4a. If server S1 needs twice the processing capability then instead of shifting a processor from some where else we can change the server to processor allocation table in the manner shown in Figure 4b. If S2 no longer need processors then we can remove S2 from the table.

6. Performance Measures for Parallel Real-Time Programs

A structure of real-time algorithm (a network of tasks), an initial allocation, and control strategies to dynamically change the allocation defines a *dynamic real-time architecture.* Given two such architectures we need to compare their performance and discuss the impact of various design changes. The question to be addressed is: how do we measure the impact of design changes?

There are two aspects of a real-time design that a designer can change: (1) the architecture as defined by the flow of tasks and the allocation of resources to the tasks; and (2) the control strategies. Effects of architectural changes are reflected in terms of the overall usage of resources which in turn gets reflected as execution time or speed-up, the congestion in task queues or the number of tasks waiting at various q-managers, and the processor utilization. In addition to these three parameters the control strategies also affect responsiveness and graceful degradation.

o Speed-Up

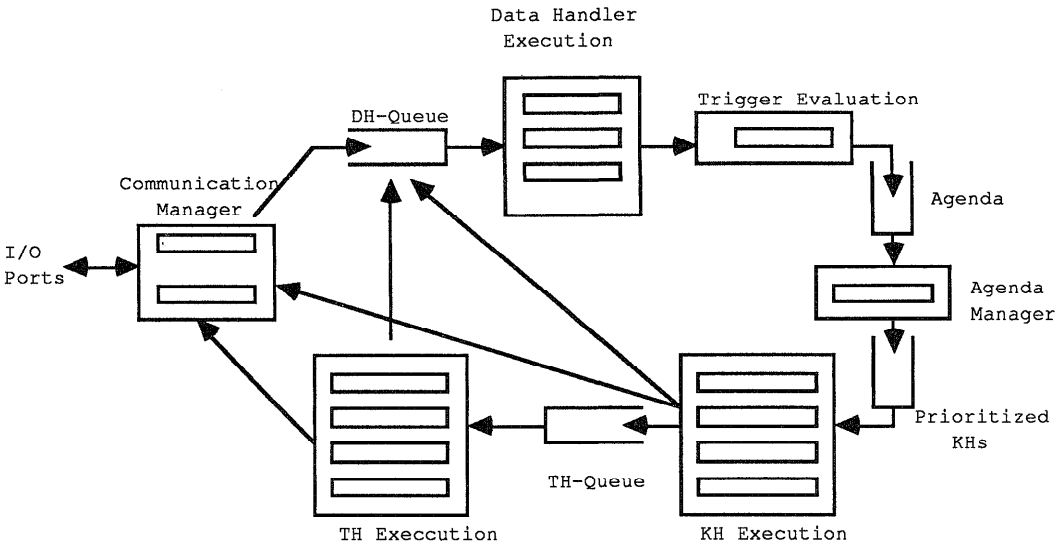
Speed-up is a good measure of performance for algorithms where the computations consists of a finite number of tasks of predictable size. RT problems are not amenable to the algorithmic approach and are better modeled as a flow of tasks through a network of task queues and servers. Speed-up is not a useful measure for the flow of tasks because:

- The flow of tasks is an indefinite process.
- The time it takes to complete a task is unpredictable because it depends on several factors which can affect the time at which the task is scheduled, possible suspension of tasks, and subsequent resumption at an indefinite time in future. Performance depends not only on the number of processors and synchronization effects, but also on the rate at which tasks arrive.
- The network of tasks queues and processors itself is dynamic because it changes to accommodate the demands of the external environment.

The network of tasks requires control algorithms to respond to the demands of the external environments. Since speed-up is useful at the algorithmic level and it can provide a useful measure for the control algorithms.

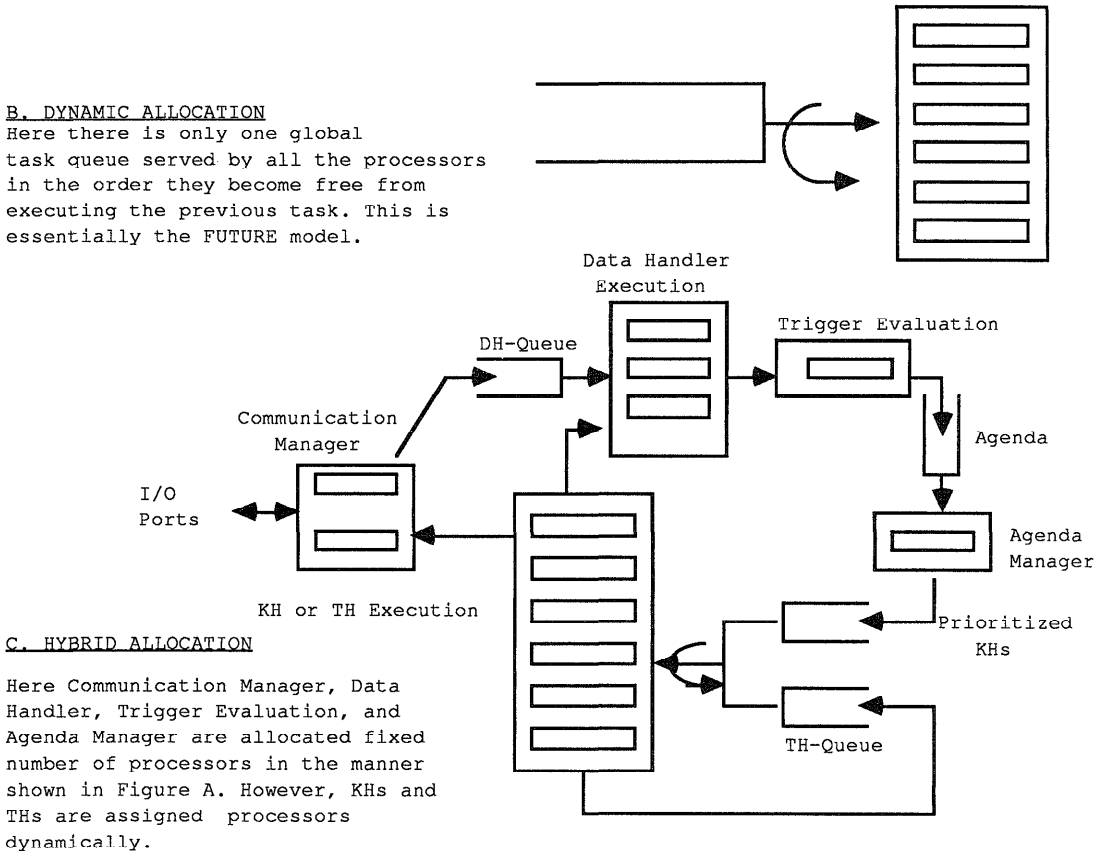
o Congestion: The Number of Tasks Waiting for Execution

Congestion is a measure of the flow of tasks and a useful parameter for resource allocation. The size of the queue tends to increase and decrease (since the arrival times and the processing times of tasks cannot be totally controlled or predicted) and one can compute the average size. The mean queue size is a useful parameter; it is a result of how the system performs, it is measurable, and it is



A. STATIC PROCESSOR ALLOCATION

Here 2 processors are dedicated to Communications Manager, 3 to Data Handlers, 1 each to Trigger Evaluation and Agenda manager, and 4 each to KHs and THs.



B. DYNAMIC ALLOCATION

Here there is only one global task queue served by all the processors in the order they become free from executing the previous task. This is essentially the FUTURE model.

C. HYBRID ALLOCATION

Here Communication Manager, Data Handler, Trigger Evaluation, and Agenda Manager are allocated fixed number of processors in the manner shown in Figure A. However, KHs and THs are assigned processors dynamically.

FIGURE 3. Schemes for Allocating Processors to Tasks

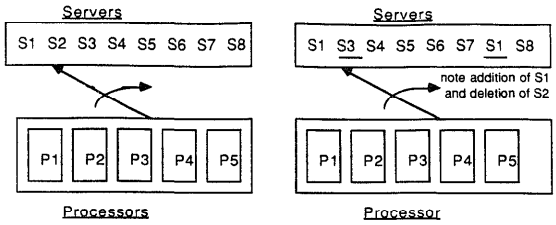


Figure 4. Allocation of Servers to Processors

possible to analytically relate the mean queue size to the number of processors [Cox 61]. For a large network of queues and processors it also provides a means to pinpoint hot areas of congestion.

o Processor Utilization

The third parameter is the fraction of time processors are doing useful work. This parameter provides a measure of how economical the system is and if there is room for improvement. It can be measured, provides an indication of system performance, and the overall system utilization can be analytically expressed in terms of the properties of individual processors [Cox 61].

o Responsiveness

Responsiveness indicates how deftly the system can respond to dynamic task demands. Responding to dynamic tasks may require detecting the need to take some action (e.g., an overgrown task queue), undertake new tasks, or shift resources from the current tasks to new tasks. A useful measure of responsiveness is the latency of tasks in a certain priority class, i.e., the mean queuing-time.

o Graceful Degradation

Graceful Degradation refers to the ability of the system to adapt to workloads exceeding the processing capability of the system. Specific measures will be number of critical tasks correctly completed and the solution quality. Solution quality can be measured by measuring how many unacceptable solutions were generated and how many acceptable solutions were missed.

7. Conclusion

Real-time control is an important and challenging research area. In our view important research problems are:

- developing an understanding of the role of knowledge and control strategies for achieving real-time performance;
- developing high level concurrent computational models to support the required knowledge-intensive problem solving;
- developing an understanding of important performance measures and how to use them for designing better solutions.

Viewing real-time control as a resource allocation problem provides a useful framework for studying the issues mentioned above. Our research group at FMC is studying these problems in the context of problems of interest to us. We believe that the QP-Net model provides a useful framework to study the implications of developing concurrent architectures for real-time applications and for understand their performance characteristics. We have implemented the basic elements on a 16 node Butterfly multiprocessor in Butterfly Scheme and are currently implementing the refinements on a Symbolics using an object-oriented system with simulation of concurrency.

References

[Cox 61] D.R.Cox and W.L. Smith.*Queues*. John Weily and Sons Inc., 1961.

[D'Ambrosio 87] B. D'AMbrosio, M.R. Fehling, S. Forrest, P. Raulefs, and B.M. Wilber. Real-Time Process Management for Material Composition in Chemical Manufacturing. *IEEE Expert* 2(2), Summer, 1987.

[Halstead 86] R.H. Halstead. Parallel Symbolic Computing. *IEEE Computer*, August, 1986.

[Nii 86] H.P. Nii. Blackboard Application Systems. *AI Magazine* 3(3), August, 1986.