Proceedings of the American Control Conference
Albuquerque, New Mexico June 1997
0-7803-3832-4/97/$10.00 © 1997 AACC

Task Assignment and Scheduling for Open Real-Time Control Systems

Byung Kook Kim
Department of Electrical Engineering
Korea Adv. Inst. Science and Technology
373-1 Kusongdong, Yusongku
Taejon, 305-701 Korea
bkkim@ee.kaist.ac.kr

Kang G. Shin
Real-Time Computing Laboratory
Department of Elec. Eng. Comp. Sci.
The University of Michigan
Ann Arbor, Michigan 48109-2122
kgshin@eecs.umich.edu

Abstract— A new problem for task assignment and scheduling on a network of processors is formulated and solved for open real-time control systems. In order to ensure smooth operation and good performance of open real-time control systems, one must analyze the problem of task assignment and scheduling during the conceptual system design stage. For this type of applications, we propose use of a new performance index called the *control latency*, a weighted sum of feedback, command, and monitoring latencies. Given a set of tasks for a specific control application, the execution time of each task, and intra/inter-processor communication latencies, we have developed an optimal task assignment and scheduling algorithm by minimizing this performance index. Since this problem is NP-hard, we have employed a branch-and-bound (B&B) algorithm to efficiently search for an optimal task assignment while maintaining task schedulability. A prototypical example of open-architecture control for CNC machines is presented to illustrate the good performance of the proposed algorithm.

Keywords: Open architecture control, real-time control, task assignment and scheduling.

I. Introduction

Time-critical industrial applications are usually run on a digital computer system composed of multiple processors joined by a certain interconnection network. Control, data logging, and monitoring functions are performed by these processors. These processors are also connected to management and production scheduling computers. The rapidly increasing power of modern microprocessors and networks at affordable prices has enabled us to implement large-scale, complex control functions on a network of processors, with a suitable hardware/software architecture design.

The idea of open architecture control is to build a control system with standard modular components, including off-the-shelf modular hardware such as buses (VME, PCI), boards (CPU, digital/analog input/output boards), networks (Ethernet, Fieldbus), and stand-alone systems (workstations, PC's). Also, modularized software can be utilized with well-defined I/O and other functions, which provide predictable behaviors. Open architecture controllers should also have well-defined module interfaces so that modules can be developed independently by different vendors then integrated by a third party. As defined by the IEEE 1003.0 Technical Committee of Open Systems, an open system provides capabilities that enable properly-implemented applications to run on a variety of platforms from multiple vendors, interoperate with other systems applications, and present the user with a consistent style of interaction [5]. By effectively combining these modular HW and SW components, a flexible and powerful control system can be built with minimum effort, and can be modified or upgraded with ease.

Control actions are usually taken periodically. According to system requirements, sampling rates can be assigned to various tasks. Hence, different control and monitoring tasks for real-time open-architecture control systems require different sampling rates, and should be assigned to different processors, so that the set of tasks assigned to each processor may be completed within a prespecified time limits called the *control system deadlines* [4]. These tasks should be able to run periodically and in a timely manner (i.e., the set of tasks is schedulable). Also, a communication channel is required between each pair of communicating tasks, in order to exchange the required data between them. The execution of tasks on each processor should be sequenced properly in order to send/receive data them in a timely manner. Considering these facts, tasks should be distributed to processors without overloading any of them (task assignment), and the tasks assigned to each processor should be scheduled properly (task scheduling). The overall system should provide good performance — a smooth flow of overall jobs including communication and control (a suitable performance index will be defined).

The issues of task assignment and scheduling for multiple-processor systems are significantly harder to solve than the uniprocessor case, as it requires to determine when and where to execute a given task [10]. The task assignment problem in distributed systems that minimizes the sum of task processing and interprocessor communication costs can be solved by graph-theoretic, integer programming, or heuristic approaches [2]. Real-time constraints are difficult to impose when a graph-theoretic approach is used. Integer programming methods allow for constraints on task completion time, but do not account for task precedence constraints.

Since this problem is generally NP-hard, we need to

The work reported in this paper was done as part of Thrust Area 2 of the NSF Engineering Research Center on Reconfigurable Machining Systems.

develop enumerative optimization or heuristic approximation. One can use the popular branch-and-bound (B&B) method [7] solve the problem. For example, Peng *et al.*[8] solved a combined task assignment and scheduling problem for communicating periodic tasks executing on a heterogeneous distributed system. The maximum normalized task response time, called the *system hazard*, is minimized by utilizing a B&B algorithm. However, it is not easy to use this method for modular real-time control systems with various sampling time requirements and minimizing the overall performance index, the control latency. A method for optimal combined task and message scheduling in distributed real-time systems was proposed in [1], where communicating tasks with precedence constraints for each processor are scheduled off-line, without considering task assignment.

The main goal of this paper is to formulate and solve a new task assignment and scheduling problem for multiprocessor-based open real-time control systems. We propose a new performance index called the *control latency*, which is a weighted sum of feedback, command, and monitoring latencies. Given a set of tasks for a control application, execution time of each task, and intra/inter-processor communication time, we develop an optimal task assignment and scheduling algorithm by minimizing the performance index. Since this problem is NP-hard, we use a B&B algorithm to efficiently search for an optimal task assignment while maintaining task schedulability. A typical example for open-architecture control for a CNC machine is presented, illustrating the good performance of the proposed algorithm.

The rest of this paper is organized as follows. Section 2 describes the real-time open-architecture control problem. Logical and physical architectures are described, and a new performance index proposed there. Section 3 presents the main algorithm for task assignment and scheduling. In order to demonstrate the utility and power of the proposed algorithm, we present an illustrative example of open-architecture control of a CNC system in Section 4. The paper concludes with Section 5.

II. Real-Time Open-Architecture Control

A. *Real-time control system*

The overall control function is decomposed spatially into several functional blocks with suitable communication channels between them. The multiprocessor controller acquires data from various sensors, processes the data, and delivers the processed data to actuators and/or display devices. Various control loops are needed to perform the required functions. Also, one or more graphic user interfaces may be added as a top-level man-machine interface, which is usually implemented on popular workstations or PCs. Several chasses can be networked, each containing multiple processor cards connected via a backplane bus (such as VME or PCI bus) suitable for the underlying applications. A set of OEM single-board computers for the specific bus can comprise processing nodes. The backplane bus, Ethernet board, or Fieldbus board can be used as communication channels.

There are several features which distinguish our task assignment and scheduling problem from others as follows.

F1. Control loops are executed periodically at specified sampling rates.

F2. Each control loop may contain several tasks. For example, an adaptive control loop can be decomposed into a system identification task to estimate process variables, an adaptation task to determine control parameters, and a communication task sending parameters to a specific controller.

F3. Each task can be decomposed into modules, which may require different sampling rates and different communication channels. Each module is associated with a different communication channel.

F4. There are precedence relations among the tasks and modules. The computation module for the control algorithm should be preceded by sensing, and actuation should be preceded by the control computation, etc.

The task system can be modeled with a task graph (TG), in which computation and communication modules, communication delays, and the precedence constraints among the modules can be clearly described.

B. *Performance index*

The sensing→control→actuation sequence should be executed within each sampling period in a feedback control system. A problem with the rate-monotonic scheduling is the difficulty in guaranteeing the execution sequence of tasks of the same period [12]. The order of task execution may result in actuation→control→sensing; the sensed data at a certain instant will affect the actuation about two sampling periods later, introducing an unnecessary delay to the closed-loop system and thus degrading the performance or even causing system instability. Hence, the delay from sensing to actuation must be minimized; this led us to use the control latency as a new performance index.

An objective function can be defined in various ways. Here we consider three types of latencies.

Feedback latency L_f**:** The time required for activating a sensor, computing the control command, then driving the actuator. The data sensed at time t, following its use for control computation, affects the process at time $t + L_f$. L_f appears as a pure *time delay* in the closed-loop system. Its length and time variation have detrimental effects on system performance and stability. L_f and its variation should therefore be kept as small as possible.

Command latency L_o**:** The time required from receipt of a command from the operator or host computer to the corresponding actuation. For a simple feedback-control system, this represents the total time required to get a reference command and a feedback input, compute the control signal, then actuate the controlled process. In a wider sense, it is the time required for receiving the task command from the operator, executing the corresponding tasks, and finally actuating the target process under control.

Monitoring latency L_m: The time required to preprocess a sensed data and report the result to the host computer. This latency may not affect the control performance, but monitoring, human operator decision & command, and command latencies may constitute a wider sense of control latency considering a human as well as a machine in the control loop, i.e., human-command loop latency.

We can define an overall performance index of the control latency L as a weighted sum of feedback, command, and monitoring latencies:

$$L = \left(\sum_{i=1}^{n_f} \omega_{fi} L_{fi} \right) + \omega_o L_o + \omega_m L_m \qquad (1)$$

where ω_{fi} are the weighting factors for feedback latencies $i = 1, \cdots, n_f$, where index i is used to denote each of multiple feedback loops. ω_o and ω_m are the weighting factors for the command and monitoring latencies, respectively. The weights can be determined by considering the relative importance of each latency. Usually, the control latencies are most important, and then command and monitoring latencies are next.

C. *The Problem of Task Allocation and Scheduling*

Tasks must be distributed to processors without overloading them (task assignment), and the tasks assigned to each processor should be scheduled to minimize the control latency (task scheduling).

The set of tasks assigned to each processor should be completed in time. One can use such scheduling policies as the Earliest-Deadline-First (EDF) or Rate Monotonic (RM) algorithm. The EDF algorithm requires to check the deadlines of all tasks very frequently, but the RM policy induces little OS overhead at the expense of lower schedulable utilization. We have chosen the RM policy due to its implementation simplicity and low runtime overhead, in spite of its lower schedulable utilization.

Task assignment and scheduling problem: Given a set of n_p processors, a set of n_t tasks T_i, $i = 1, \ldots, n_t$, each task T_i consisting of m_i modules T_{ij} with period P_{ij} and execution time C_{ij}^l for local communication or C_{ij}^r for remote communication, find a task assignment and schedule while minimizing the control latency L.

III. Task Assignment and Scheduling Algorithm

The tasks described by the TG are assigned to processors by using a B&B algorithm. The algorithm employs a polynomial-time bounding heuristic at non-terminal vertices by figuring the subsequent task scheduling in its estimation of assignment cost.

The B&B algorithm maintains a set of active vertices, which are the vertices searched and considered to contain an optimal solution. Initially, this set contains only the root vertex. The algorithm proceeds by alternating branching and bounding operations. Branching refers to expanding the minimum-cost vertex in the active set by generating its children, while bounding refers to the process of evaluating the cost of new vertices in the active set. The algorithm also evaluates the control latency and schedulability at each vertex. Whenever the schedulability condition is violated at a vertex, the vertex is discarded. Whenever a smaller control latency is found, all vertices with higher latencies are removed from the active set. The algorithm terminates when the active set contains only one element representing a complete solution. It is proven in [7] that there always exists a terminal vertex surviving the above process, which is the optimal solution to the original problem. Fig. 1 shows a pseudo-code form of the B&B algorithm used in this paper.

```
Let active set = { Root }.
Let cost(Root) = 0.
Let best terminal vertex $v^* = nil$.
Let best terminal cost $L^* =$ inf.

Repeat
  Find a vertex $v$ with minimum cost.
  if v is a terminal vertex then (Bound)
    if cost($v$) < best cost then
      L = cost($v$)
      $v^* = v$
      Delete all vertices x with cost(x) >= $L^*$ from
      active set, except for vertex $v^*$.
  else (Branch)
    Generate all children of $v$.
    for each child c do compute cost(c).
    Replace $v$ by its children in active set.
    Check schedulability of its children.
until active set = { $v^*$ } .
```

Fig. 1. The branch-and-bound algorithm.

At a certain vertex in level j of the tree, a set of tasks $T_i, i = 1, \ldots, j$ have already been assigned to processors. In the branching process at this vertex, task T_{j+1} is assigned to one of n_p processors, which has n_p children to branch to. Branch k corresponds to the case when task T_{j+1} is newly assigned to processor k.

For each vertex, schedulability is checked for every processor as follows. When task T_i is assigned to this processor, for each module T_{ij}, if the task containing the communication partner's module has not yet been assigned or assigned to the same processor, increase utilization by C_{ij}^l / P_{ij}. Otherwise, increase utilization by C_{ij}^r / P_{ij}. If the utilization exceeds unity for any processor, the assigned task set is definitely not schedulable on that processor. The vertex is discarded if any of the processor utilization exceeds unity.

The performance index, control latency, is computed for each vertex as follows. For each feedback/command/monitoring latency, select a processor which contains the first task contributing to the latency under consideration. If the next task in the sequence has not yet been assigned or assigned to the same processor, the (local) computation time C^l_{ij} is added to the latency. Otherwise, the remote computation time C^r_{ij} and the communication delay d are added to the latency. A weighted sum of all latencies are computed to give the control latency.

When the terminal vertex is reached (a vertex representing the case when all tasks have been assigned), all the vertices having larger costs than the terminal vertex are discarded (bounded).

The above algorithm utilizes efficient bounding by calculating the achievable minimum cost for each intermediate vertex; it is usually shown to have a polynomial-time bounding.

IV. An Illustrative Example

A. An Open Architecture Controller for Computer Numerical Control

We demonstrate the good performance of the proposed algorithm using the University of Michigan Open-Architecture Controller (UMOAC) testbed. It is designed to control a CNC milling machine as shown in Fig. 2. Its current physical configuration uses a PC (or a workstation) as the host computer for program development and operator interface. Control tasks are executed on VMEbus-based processor boards (currently, XYCOM XVME-675/19 VMEbus PC/AT processor modules), with commercial RTOS (QNX v4.22). Sensors and actuators on the milling machine are accessed through VMEbus-based I/O interface boards. This testbed architecture allows for easy adoption of new hardware and software components as they become available, and supports good hardware/software open-ness [9]. QNX uses a priority-based, preemptive kernel scheduler, with the real-time clock resolution adjusted to 50 μs [12].

We developed a prototype modular real-time milling machine controller on the UMOAC testbed as shown in Fig. 3 where rectangles represent tasks. There is a three-axis controller with a linear or circular interpolator on the X and Y axis, and with a one-axis interpolator on the Z axis. Also, attached is a spindle drive which is a constant-speed drive, but can be controlled to have variable speeds if required.

B. Task Graph

Specifically, we consider a real-time controller without the host computer as shown in Fig. 3, for the evaluation of task assignment and scheduling. The GUI and interpreter tasks are assumed to be executed by the host computer.

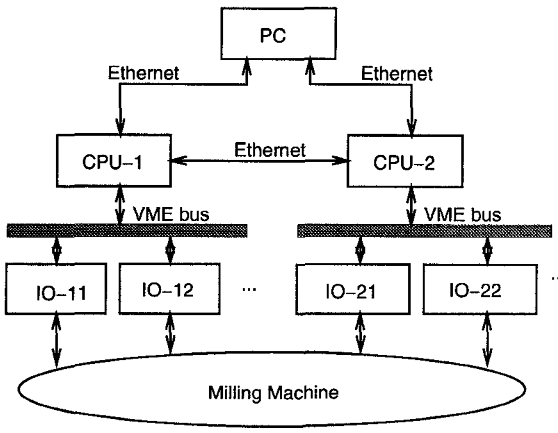

Fig. 2. The open architecture control system.

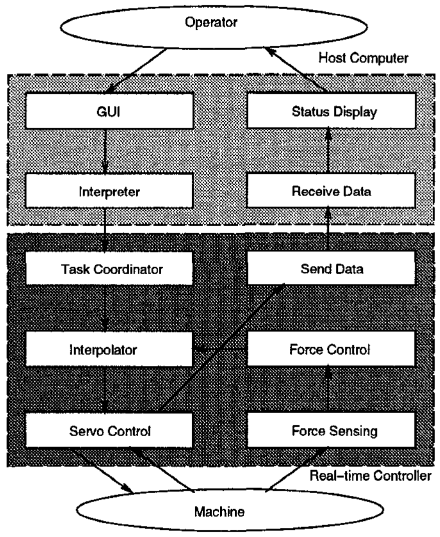

Fig. 3. Task configuration.

The host computer usually uses standard graphic packages for user interface, and we have little control on task scheduling on the host system.

Tasks of the real-time controller are constructed as shown in Fig. 4. The task coordinator gets command input from the host interface, which originates from the operator. It performs a coordination function, decomposes the command into a sequence of steps to be performed, sends each step command to the interpolator, and sends the reference force to the force controller. The interpolator performs linear or circular interpolation to generate reference positions for X, Y, and Z axes, which are then sent to the axis controllers. The axis controller gets the reference position from the interpolator and feedback from the machine about its position (and velocity), generates the control action, and sends it to the servo amplifier, hence forming the lowest-level feedback loop under computer control. The servo control of all axis uses the PID or fuzzy logic control law [3].

The servo amplifier is usually constructed with analog circuits and realized in the continuous-time domain. However, the controller is implemented in hardware and cannot be controlled by the controller software. Hence,

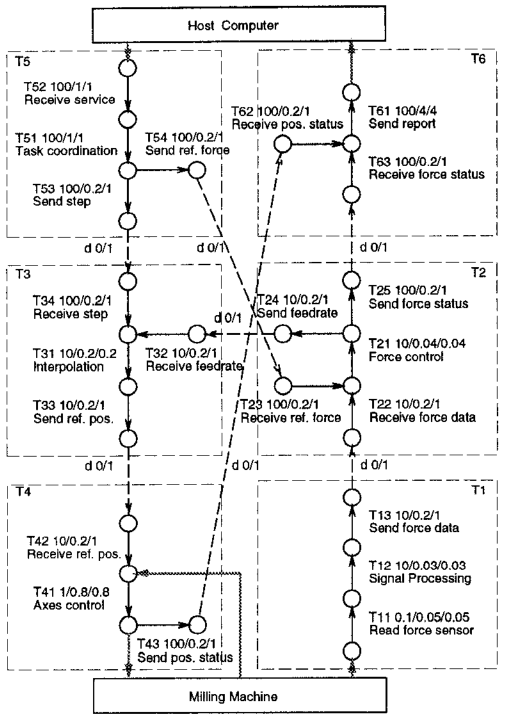

Fig. 4. Task graph for the real-time controller.

it is excluded from our consideration of task assignment and scheduling.

The force control is employed at a process-control level, to control process variables in order to maintain high production rates and good part quality. The sensor task reads the force sensor at a very fast rate (once every 0.1 ms), processes the signal to compute the average or the maximum of 100 sampled data, and sends the resultant processed force to the force controller.

In order to store the process data and present information to the operator, a graphic user interface is usually implemented on the host computer, and the real-time controller is required to collect and send the process data. The send task performs this function, by collecting process data from the axis controllers and the force controller, and sends data to the host computer.

Cross-coupling control can be added to minimize the contour error [6] to build a real-time contour error model based on feedback information, then feed back correction signals to the axis controller.

At the highest level in the control hierarchy, we can adopt supervisory control such as chatter detection, tool monitoring, machine monitoring, etc. [11].

The task assignment and scheduling problem is solved using the algorithm presented in this paper with the following Input data: number of processors $n_p = 2$; number of tasks $n_t = 5$; weighting factors $\omega_f = 1$, $\omega_o = 0.5$ and $\omega_m = 0$; and communication delay $d = 1$ ms.

For the real-time controller, each task consists of modules as shown in Fig. 4. In each module, numbers of the form $P/C^l/C^r$ denote period, local execution time, and remote execution time, respectively.

An optimal assignment is found after expanding 19 vertices out of a total $2^7 - 1 = 127$ vertices in the B&B tree. As expected, task 4 is assigned to processor 2 (91% utilization), and all other tasks are assigned to processor 1 (78.9% utilization).

V. Concluding Remarks

Task assignment and scheduling is one of the most important issues in designing distributed real-time systems. Unfortunately, this problem is generally known to be NP-hard even in the absence of precedence constraints. In this paper, we proposed a new performance index, control latency, which is a natural cost for real-time distributed control systems. We solved the task assignment and scheduling problem by using a B&B algorithm with the new performance index. The algorithm presented in this paper is shown to reduce the computational cost significantly. This fact has been confirmed via an illustrative example. The resultant task assignment and scheduling ensures smooth operation within each processor's capability, while minimizing the control latency, hence providing best control performance.

References

[1] T. F. Abdelzaher, K. G. Shin, "Optimal combined task and message scheduling in distributed real-time systems," Proceedings Real-Time Systems Symposium, Pisa, Italy, 1995.

[2] W. W. Chu *et al.*, "Task allocation in distributed data processing," *IEEE Computer*, vol. 13, pp. 57-69, Nov. 1980.

[3] Open-Architecture Controls Team, *Developer's Guide for Open-Architecture Control of the Robotool*, Department of Electrical Engineering and Computer Science and Department of Mechanical Engineering and Applied Mathematics, The University of Michigan, November 1995.

[4] Hagbae Kim, "Design and evaluation of real-time fault-tolerant control systems," Ph.D. Thesis, the University of Michigan, 1994.

[5] IEEE, *IEEE Guide to POSIX Open System Environment (IEEE 1003.0)*, 1995.

[6] Y. Koren and C. C. Lo, "Advanced controllers to feed drives," *Annals of the CIRP*, vol. 41, no. 2, pp. 1 - 10, 1992.

[7] W. H. Kohler and K. Steiglitz, "Enumerative and iterative computational approach," *in Computer and Job-Shop Scheduling Theory*, Coffman eds., Wiley and Sons, pp. 229-287, 1976.

[8] D. T. Peng, K. G. Shin, and T. Abdelzaher, "Assignment and scheduling of communicating periodic tasks in distributed real-time systems," *IEEE Trans. on Software Engineering* (in press).

[9] J. Park *et al.*, "An open architecture testbed for real-time monitoring and control of machining processes," *American Control Conference*, WA08-2, Seattle, June 1995.

[10] K. G. Shin and P. Ramanathan, "Real-time computing: A new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6-24, Jan. 1994.

[11] A. G. Ulsoy and Y. Koren, "Control of machining processes," *Journal of Dynamic Systems, Measurement and Control*, vol. 115, no. 6, pp. 301 - 308, June 1993.

[12] L. Zhou, M. J. Washburn, K. G. Shin and E. Rundensteiner, "Performance evaluation of real-time controllers," *1996 ASME International Mechanical Engineering Congress and Exposition*, Atlanta, Georgia, Nov. 1996.