Adaptive Time-Based Dispatching of Distributed Real-Time Tasks

Sameh M. Elsharkawy
Department of Electrical Engineering
and Computer Science
The Catholic University of America
Washington, DC 20064
sharkawy@ieee.org

Ashok K. Agrawala, Tamer M. Nadeem
Department of Computer Science
University of Maryland
College Park, MD 20740
(agrawala,nadeem)@cs.umd.edu

Abstract

This paper describes a system model for dispatching non-preemptive periodic hard real-time tasks with inter-task temporal dependencies in a distributed environment. The model dispatches soft real-time and non-real-time tasks in the background on a best effort basis. The model is based on using the parametric cyclic scheduling method. This method verifies the feasibility of the task set and generates parametric dispatching calendars that adapts to run-time execution parameters. The model uses the parametric calendars generated by a dynamic time based off-line scheduler that verifies the feasibility of the distributed hard real-time task set. The distributed parametric run-time kernel guarantees the temporally determinate dispatching of hard real-time tasks and best effort performance for soft real-time tasks. The use of the dynamic time based scheduling method, provides off-line guarantees for all the timing requirements of the hard real-time tasks while the parametric dispatching mechanism maintains a flexible run-time environment that makes use of the slack time with a limited overhead while satisfying all the system’s timing constraints.

Keywords: Real-time, Scheduling, Dispatching, Distributed systems, Parametric, Time-based.

1 Introduction

Distributed applications require a wide range of Quality of Service (QoS) guarantees from the underlying system. QoS guarantees range from best effort performance required by non-real-time and soft real-time applications to the prior guarantee to meet all timing requirements and deadlines requested by hard real-time applications. To provide timing guarantees for real-time distributed applications, both individual nodes’ operating systems as well as the network management system must collaborate to provide an end-to-end QoS enforcement of system timing feasibility.

The dynamic time-based parametric scheduling method was introduced by M. Saksena et al [8] for single-node execution environments. It is further extended to include periodic tasks by S. Choi [2]. This method uses Fourier-Motzkin variable elimination technique [3] to verify the schedulability of a real-time task set and calculate a dynamic calendar for dispatching jobs at runtime. The dynamic calendar represents the start time of each job \( \tau_i \) with two parametric functions \( F^\text{min}_{\tau_i}, F^\text{max}_{\tau_i} \) whose evaluation generate the minimum and maximum feasible starting times of the corresponding job. The parameters to these functions consist of time event variables, like jobs’ start and finish times, whose values are generated at runtime by previously executed jobs. By deferring the calculation of tasks’ actual starting times and parameterizing its calculation by run-time values, this method maintains a flexible run-time environment that makes use of the slack time with limited overhead.

Time-based scheduling of hard real-time tasks across a network of distributed nodes requires addressing several issues that does not exist in the single-node execution environment. These issues include time-constrained guaranteed delivery of communication messages, parallel execution of tasks on the distributed nodes, synchronization of distributed tasks’ execution, and handling of external time events. Several algorithms have been proposed for scheduling real-time tasks in distributed environments. Scheduling methods can be classified as either priority-based (static or dynamic), and time-based. Existing priority-based scheduling algorithms [4, 13, 10, 12] are not suitable for non-preemptive, periodic, hard real-time tasks, with inter-tasks relative timing constraints because they do no provide sufficient timing guarantees. On the other hand, existing time-based scheduling algorithms [1, 11, 9, 14] use heuristics or branch-and-bound search to produce static calendars for hard real-time tasks. These methods lack efficiency and flexibility be-
because they base their schedules on the worst-case timing requirements of the real-time tasks.

Our model extends the Dynamic time-based parametric scheduling method so that it can be applied to a distributed set of real-time tasks. The model maps the distributed scheduling problem into a single-node problem, applies the parametric cyclic scheduling method to that problem, and finally maps the schedulability results back to the distributed problem. Therefore, our model provides a schedulability verification method for distributed, non-preemptive, periodic, hard real-time tasks. On the other hand, it maintains flexibility and efficiency by dynamically adapting the run-time dispatching calendars according to generated timing values at run-time. The model supports various types of timing constraints among the given tasks like relative timing constraints, absolute timing constraints, and communication constraints.

In this paper we present algorithms and structures for dispatching real-time task instances in distributed environments at run-time. Distributed system node dispatchers need to coordinate and propagate run-time execution information in order to maintain and enforce the system’s timing requirements. This coordination is achieved by means of the presented parametric calendar synchronization and distributed time-based event handling mechanisms, which ensure proper temporal dispatching of distributed real-time tasks.

2 Problem Description

In this section, we define the task and the network models for the problem under consideration.

2.1 Task Model

The environment under consideration consists of a set of $M$ computer nodes \{Node$_1$, Node$_2$, ..., Node$_M$\}. On each node, runs a group of non-preemptive periodic hard real-time tasks. The least common multiple (LCM) of tasks’ periods on all nodes is $L$, which is also known as the scheduling window on all nodes. In each scheduling window, there is $N_m$ task instances (jobs) that run on node $m$, such that $1 \leq m \leq M$. The total number of jobs running on all nodes in one scheduling window is $N = \sum_{m=1}^{M} N_m$.

Let $\Gamma_m^j = \{\tau_{i,m}^j \mid i = 1 \ldots N_m\}$ denote the ordered set of $N_m$ jobs to be dispatched sequentially in the $j$th scheduling window $[(j-1)L, jL]$ on node $m$. Jobs are non-preemptively executed for every scheduling cycle. The execution order for this job set is predetermined, and enforced by precedence timing constraints. The set of tasks to be dispatched on all nodes in the $j$th scheduling window is represented by $\Gamma^j = \{\Gamma_1^j \cup \Gamma_2^j \cup \ldots \Gamma_M^j\}$. Each periodic real-time task in the system needs to specify the parameters that are common for all its instances (jobs) such as the task’s period $P$. In addition to the parameters inherited from the task, there exist a number of parameters for each job $\tau_{i,m}^j$ that specify its timing behavior and characteristics, these parameters are the instance’s start time $s_{i,m}^j$, execution time $e_{i,m}^j$, finish time $f_{i,m}^j$, minimum execution time $t_{i,m}^j$, maximum execution time $u_{i,m}^j$, release time $r_{i,m}^j$, and deadline $d_{i,m}^j$.

For every job, only two time event points can be used as time variables, the start time $s$ and the finish time $f$. Between any two time variables on the same node, there can be at most two relative timing constraints. These constraints form the lower and upper bounds on the time period between the two variables. A relative timing constraint involving only two time variables is referred to as Standard [2]. The set of all relative timing constraints among jobs running on physical node $m$ is represented by $C_m$. The system timing constraints set consists of the union of all local timing constraint sets on all the separate nodes $C_m, 1 \leq m \leq M$ plus the communication constraints $C_c$ (Definition 2.1).

\begin{equation}
C = C_1 \cup C_2 \ldots \cup C_M \cup C_c
\end{equation}

2.2 Network Model

The network model considered in this problem consists of $M$ processor nodes connected by point-to-point dual simplex links. A link connecting Node$_i$ to Node$_j$ is referred to as Link$_{ij}$. A node in the system can have several incoming and outgoing links attached to it, each of which can operate in parallel with the others. Nodes’ clocks do not have to be synchronized, but the clocks’ rates are assumed to be equal. The maximum skew between clocks’ rates on different nodes is assumed to be very small compared to message transfer delays.

Between two tasks running on two different nodes, a periodic communication channel can be established which can transfer periodic messages. Communication channels can span multiple network point-to-point links. The links that a channel goes through are determined using a static routing algorithm to ensure transfer time predictability. Communication channels can only be established from a source task instance in one scheduling window and a destination instance in the same or next scheduling window, that executes on a different physical node. A communication channel is
specified by its source task instance, destination task instance, message generation scheme (maximum message size ($S$), maximum message rate ($R$), and burst size ($B$)), and desired end-to-end maximum message delay ($q$). A communication channel imposes an upper bound on the message delivery time experienced by each message transferred on this channel. The delay limitations imposed by communication channels are referred to as Communication constraints (Definition 2.1). The set of all communication constraints among the $M$ nodes is represented by $C_c$.

**Definition 2.1 (Communication Constraints)**

A communication constraint is the upper limit $q^{k,i,n}_{i,j,m}$ imposed on the delivery time of a message sent over a communication channel established from one job $\tau^j_{i,m}$ to another job $\tau^i_{k,n}$ on two different nodes $m, n$. Therefore, a lower limit is imposed on the time distance between the finish-time of the source job $f^i_{i,m}$ and the start-time of the target job $s^k_{k,n}$ to accommodate the worst case message delivery time. Communication messages are assumed to be periodic. Each message is assumed to be sent at the end of the job execution, and completely received by the destination node before beginning the execution of the target job.

$$s^k_{k,n} - f^i_{i,m} \geq q^{k,i,n}_{i,j,m}$$

(2)

### 3 Schedulability Analysis

The schedulability of a task set is established if and only if we can find starting times for all jobs that satisfy all timing constraints for all possible execution times. In [6] we presented the scheduling model which includes definition of schedulability conditions, schedulability verification algorithms, and run-time calendar construction methods. In that paper we showed how a separate cyclic dynamic calendar is calculated for each one of the distributed nodes. A node calendar consists of two parametric functions for each one of the jobs, minimum start time $F^{min}$ and maximum start time $F^{max}$. Communication timing constraints are appended to the generated dynamic calendars by adding the arrival time of communication messages as parameters to the minimum start time functions of communication channels’ target jobs. The resulting parametric calendars are used by the nodes’ dispatchers to start the execution of hard real-time jobs according to their timing constraints as described in section 4.

### 4 Dynamic Time-based Dispatching

A single real-time dispatcher runs on each one of the system nodes. The function of the on-line dispatcher is to start the execution of the real-time task instances according to the calculated calendars as well as the timing information generated at run-time. Therefore, enforcing the system schedulability established by the off-line scheduler while being flexible enough to make use of the slack CPU time. Slack time can be used to execute soft and non real-time task loads in the context of a lower priority kernel thread. The dynamic time-based dispatcher processes the parametric calendars generated by the off-line scheduling module to create and populate the run-time data structures that are used in the process of determining the absolute dispatch time for the different task instances according to run-time parameters. This section describes the data structures used by the on-line component, and then explains the use of these data structures to handle the task dispatching process.

#### 4.1 Run-time data structures

Scheduling information needed for the dispatching process are transferred to the on-line component. This information consists of task descriptions, task-node assignment, task relative ordering on each node, and relative timing constraints in the form of parametric calendars. The calendars consist of functions used to determine the minimum and maximum feasible bounds on the execution start times for the task instances (figure 1). The Run-time information is stored in the form of a dependency graph of the tasks and their timing properties. Along with the dependency graph, the Time Ordered List (TOL) and the External Event Queue (EEQ) are used to dispatch hard real-time task instances at run-time. The structure of the run-time components is described as follows:

![Figure 2: Dependency Graph](image)

**Dependency Graph (DG):** A graph-like structure, shown within Figure 2, that includes all task instances that are active in the system at the current time along with all their timing requirements,
inter-task relative timing constraints, inter-node communication constraints, and task instance dependencies. The Dependency graph is represented as a list of task objects each containing task related parameters such as Task’s ID and Execution period. Additionally, each task object contains a list of the task’s instance profiles, each containing the following information:

- Instance ID.
- Minimum execution time.
- Maximum execution time (WCET).
- Activation counter, that describes the number of life cycles of the task that this instance is going to remain active in.
- Instance functions, a list of parametric functions, each containing a pointer to the function’s evaluation code, a list of the function parameters, and an Evaluation counter for the unresolved parameters in the function.
- Result lists, which are lists of Evaluation pointers from the timing values generated by the execution of a task instance to the locations of the corresponding parameters in the parametric function of other task instances. These pointers indicate that timing values from the source task instance are the actual parameters for the formal parameters in the target task instances’ functions. A separate list is maintained for each time value.
- Communication list, a list of the messages to be delivered to other task instances, running on different nodes, at the end of the execution of this task instance.

**Time Ordered List (TOL):** A time ordered list of task instances is maintained by the run-time module. Entries in the list represent task instances that the run-time module have full knowledge about their execution profile. The parameters of their parametric functions are all satisfied and the functions are evaluated to yield an absolute time to start the execution of the task instance. Entries in the TOL consist of the absolute minimum and maximum feasible starting times to start executing the task instance. It includes a pointer to the task instance profile in the dependency graph. Entries in this list are ordered according to their earliest feasible starting times.

**External Event Queue (EEQ):** This is a First-In-First-Out (FIFO) queue of the incoming communication messages received from external nodes. Each message should include the message ID, target task ID, target instance ID, event arrival time, information about instance function parameters to be substituted by the message arrival time, and the message’s data payload.

### 4.2 Run-time Execution Model

The major functionality of the on-line dispatcher is to propagate time event values generated at run-time to their corresponding parameters in the parametric functions, and dispatch the correct task instances according to the guidelines of the calendar generated by the off-line scheduler. The dispatcher system has three major phases: The **Initialization phase**, the **Calendar synchronization phase**, and the **Task execution phase**.

#### 4.2.1 Initialization phase

The **Initialization phase** starts by processing the calendar information passed in by the off-line scheduler in the form of parametric functions. The scheduling information is used at run-time to populate the dispatcher’s dependency graph. The TOL is initialized with one task instance, which is the task marked by the off-line scheduler to be executed first. This instance execution time does not depend on time values generated by other task instances.

#### 4.2.2 Calendar synchronization phase

The purpose of the **calendar synchronization phase** is to make all the distributed nodes dispatchers start executing their real-time calendars at the same reference time $t_0$. The calendar synchronization process is maintained by a single node called the **Time-Reference Node** which repeats the synchronization process for each one of the distributed nodes (**Client Nodes**). The process assumes that the message delivery time of the synchronization communication messages between the Time-Reference node and the Client node will always be the same ($\delta_c$) during the calendar synchronization process.
The calendar synchronization process between the Time-Reference node $A$ and a Client node $B$ is illustrated in Figure 3. Time-Reference node $A$ starts by measuring the process start time $T_{A1}$ according to its own local clock. Next, node $A$ sends a message $M_{A1}$ to node $B$ containing $T_{A1}$ and time-stamped by send time $s_{A1}$. When node $B$ receives message $M_{A1}$ coming from node $A$, it measures its arrival time $r_{B1}$, which is measured using node $B$ clock. Node $B$ then replies by sending a message $M_{B1}$ back to node $A$ containing $r_{B1}$ and time-stamped by send time $s_{B1}$. Finally, Node $A$ records message $M_{B1}$ arrival time $r_{B2}$, and sends a third message $M_{A2}$ back to node $B$ containing $r_{B2}$ and time-stamped by send time $s_{B2}$. At this moment, each one of the two nodes can calculate three time intervals $\delta_1, \delta_2, \text{and } \delta_3$ according to equation set 3. All three time intervals are generated as the difference between two time measurements generated by the same node clock to avoid errors resulting from the differences among the nodes’ system clocks.

\begin{align*}
\delta_1 &= s_{A1} - T_{A1} \\
\delta_2 &= s_{B1} - r_{B1} \\
\delta_3 &= s_{A2} - r_{A2}
\end{align*}

When the client node $B$ receives the third message $M_{A2}$ at time $r_{B2}$, it can calculate an estimate of the synchronization message delivery time $\delta_c$ according to equation 4. Using $\delta_c$, node $B$ can calculate its equivalent version of the time instance $T_{A1}$ according to its clock $T'_{A1}$ using equation 5.

\[ \delta_c = \frac{r_{B2} - s_{B1} - \delta_3}{2} \]

\[ T'_{A1} = s_{B1} - \delta_2 - \delta_1 - \delta_c \]

By choosing a synchronization waiting period $W$ long enough for all the nodes to finish their calendar synchronization process, all nodes can calculate the reference time $t_0$ according to their own system clock using the following equations.

For the client node $B$:

\[ t_0 = T'_{A1} + W \]

For the time reference node $A$:

\[ t_0 = T_{A1} + W \]

### 4.2.3 Task execution phase

At the unified calendar start reference time $t_0$, run-time dispatcher extracts the first task instance in $TOL$, and start executing it in the earliest possible time between its minimum and maximum feasible starting times. The kernel schedules an interrupt at the end of the WCET of that task instance in order to be able to gain control and maintain the schedule of the remaining tasks execution.
the unresolved parameters counter in any one of target task instances reaches zero, this means that the parameters to its functions are all satisfied and functions can be evaluated at this point. The absolute boundaries on the starting times for the ready task instances are calculated, the instances are inserted in the TOL, and their evaluation counters are reset to their original values in the instance profiles. The dispatcher also maintains the information in the task-instance profiles regarding the number of cycles the instance is going to be active in, this counter is decremented every time the instance is executed. If this counter was initialized with a negative value, this will cause the dispatcher to run this task periodically for as long as the operating system kernel is running this particular application. The on line dispatcher time complexity is $O(N)$, where $N$ is the total number of task instances in one scheduling window. The main steps for the On-line dispatcher are shown in Figure 4. The detailed algorithm for on-line task dispatching can be found in [5].

5 Conclusion

We presented a model for scheduling and dispatching a set of non-preemptive periodic hard real-time tasks with inter-task relative timing constraints and time-based communication constraints in a distributed environment. The model includes the task and network model description, schedulability condition’s definition, schedulability verification algorithms, calendar construction algorithms, run-time calendar synchronization algorithm, and run-time task dispatching mechanism. The scheduling algorithm depends on mapping the distributed scheduling problem into a single node problem, solving it using the parametric cyclic scheduling scheme, and finally mapping the schedulability results back to the distributed task set in order to create calendars for the distributed system nodes. The complete model was implemented and tested using simulated sample real-time distributed systems. Model implementation details, testing environments, and simulation results were published in [7].

We believe that the distributed parametric scheduling and dispatching methods can be applied to many distributed real-time applications that have strict and complex timing constraints in resource management and task communication. The method provides such applications with timing guarantees and flexibility to manage slack-time at runtime.

References