Dynamic Real-Time Scheduling in Distributed Environments

Sameh M. Elsharkawy Ashok Agrawala
{sharkawy, agrawala}@cs.umd.edu
Department of Computer Science, University of Maryland at College Park

September 24, 2001

Abstract

Real-time applications are becoming increasingly popular in distributed environments. These real-time applications range from hard real-time applications with periodic or aperiodic tasks and intertask relative timing constraints to soft real-time applications with best effort timing requirements. This paper introduces a complete system model for scheduling and dispatching hard as well as soft real-time tasks with intertask temporal dependencies in distributed environments. The model uses a dynamic time based off-line scheduler to verify the feasibility of a distributed hard real-time task set, and a parametric run-time kernel that guarantees the temporally determinate dispatching of hard real-time task instances and best effort performance for soft real-time task instances. The use of the dynamic time based scheduling, 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.

1 Introduction

A *distributed system* is collection of inter-connected processors that does not share memory or clock . The system provides users with access to various resources maintained by the system [1]. Distributed computing environments have become the dominant operation environment in educational as well as industry sites. This increasing popularity is due to what this environment offers in the sense of improved performance through multi-processing, connectivity through geographical location distribution, scalability and portability through modularity, availability and reliability through resource replication, and cost effectiveness [2]. Distributed applications running on these environments 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. Among the common real-time distributed applications are telecommunications systems, command and control, multimedia systems and distributed simulations (figure 1).

To provide timing guarantees for real-time distributed applications, both individual nodes operating systems and the network management system must collaborate to provide an end-to-end QoS enforcement of global system timing feasibility. Therefore, a complete system design model needs to be developed that takes into consideration resource scheduling on computation nodes and communication resources management in the underlying network connecting them.

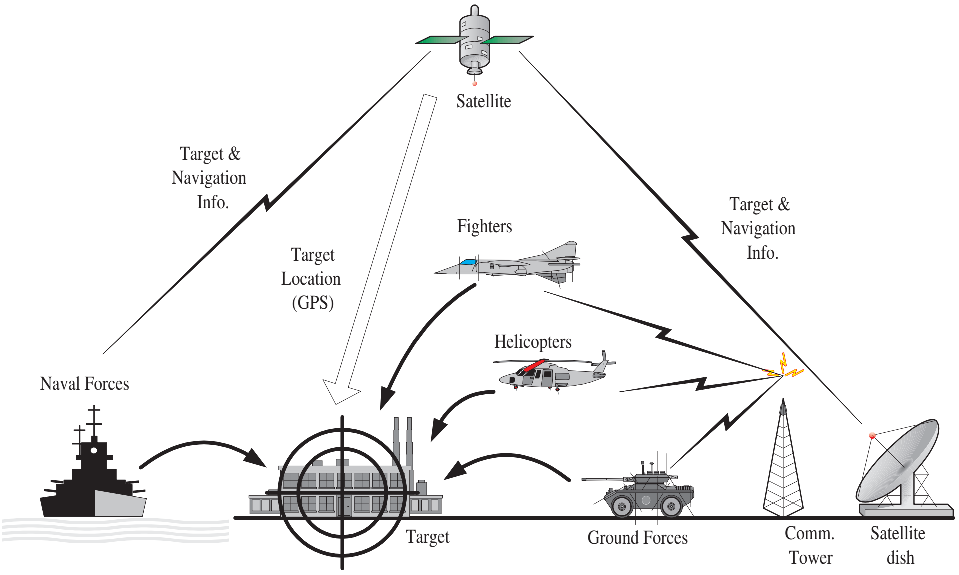

Figure 1: Distributed Command and Control

1.1 Motivation

This paper addresses the problem of distributed real-time scheduling with various types of QoS requirements and architectural models.

Many real-life distributed applications have strict timing requirements and require timely interaction among the various tasks in the application. The existing solutions for such hard real-time applications use *static* (off line), *dynamic* (on line) or hybrid scheduling method to allocate resources to the different tasks in the system. Static scheduling algorithms provide time predictability at the expense of flexibility and performance at run time. Dynamic algorithms, on the other hand, provide a more flexible and time efficient solution but do not provide acceptable timing guarantees to many hard-real time applications. The use of the dynamic time-based parametric scheduling method provides off-line timing guarantees for hard real-time tasks and flexible parametric dispatching mechanism at run-time that makes use of the slack time with a limited overhead. Some of the currently existing applications that closely match the system environment under consideration are:

1. *Distributed interactive simulations (DIS):* Interactive simulations are used to duplicate the experience of situations that are too expensive, dangerous, or impractical to facilitate in the real world. For example, in the case of *Synthetic theater of war training (STOW)* problem, military units around the globe can participate a joint exercise that involve a simulation of real global combat situations [3]. STOW applications usually involve the timely initiation of several distributed events and reactions that require a strict timing and QoS guarantees from the underlying system.

2. *Mission-critical real-time distributed systems:* These systems include avionics mission computing systems, tactical command and control systems, and manufacturing process control systems. This type of real-time applications require the support for various types of QoS aspects such as bandwidth, latency, jitter, and time-dependability [4]. Recent large-scale mission-critical applications require the interaction among large numbers of distributed tasks that are running on several distributed computing nodes. For instance, in avionics mission computing systems, the aircraft controller must collaborate with remote command and control systems, provide on-demand browsing capabilities for human operators, interact with satellite systems to calculate geographical position, and respond, in a timely manner, to

unanticipated factors that might arise in the run-time environment [4].

3. *Distributed electronic medical imaging systems (EMIS):* Advances in the areas of high-speed networking and hierarchical storage management facilitate the building of large-scale, distributed, performance-sensitive EMISs. Distributed EMISs require a great deal of flexibility, performance and QoS from the underlying communication infrastructure in order to be able to provide message-oriented and stream-oriented media on-demand to any of the distributed diagnostic stations across local and wide area networks [5].

1.2 Approach

The main problem addressed in this paper is that of scheduling and dispatching real-time tasks running on a network of distributed computing nodes. The major scheduling methodology used is the Dynamic time-based parametric scheduling method initially introduced by M. Saksena et al [6] and further extended to include periodic tasks by S. Choi [7]. This method uses Fourier-Motzkin variable elimination technique [8] in the off-line phase to verify the schedulability of the real-time task set and calculate a dynamic calendar for dispatching jobs at runtime. The dynamic calendar represents the start time of each job τ_i with two parametric functions $(\mathcal{F}_{s_i}^{min}, \mathcal{F}_{s_i}^{max})$ 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. The parametric scheduling method was chosen because it provides hard real-time schedulability guarantees, as well as, flexibility to manage slack times without affecting the task set schedulability [7].

1.3 Outline

The rest of this paper is organized as follows. Section 2, summarizes prior work in the areas of hard and distributed real-time scheduling. The problem of *Distributed Hard Real-Time Scheduling* is presented in the remainder of the paper's sections. We start by describing the problem definition and the system model for this problem in section 3. Next, we define the parametric schedulability condition of the global system as well as each of the distributed nodes in section 4. Then we introduce the solution algorithms for verifying global system schedulability, calculating dispatching calendars for distributed nodes, and timely dispatching task instances in conformance with the system timing requirements in section 5. Section 6 provides the correctness proof for the scheduling algorithms. The structure of the run-time dynamic dispatcher is described in section 8. Finally, we describe the complete model implementation, practical experiments, and results in section 9.

2 Related Work

The area of real-time scheduling has been an active research topic for a relatively long period of time due to the wide and changing demands of the real-time applications. With the distributed workstations environment becoming the dominant operation environment, real-time scheduling work in distributed environments is rapidly growing.

In the following sections, we briefly present some of the work that have been done in the closely related areas to the presented problem.

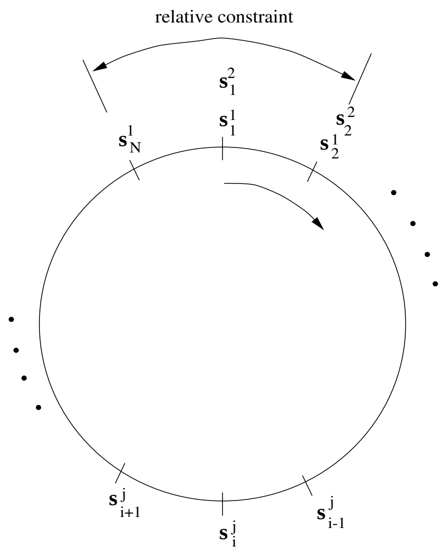

Figure 2: Static Cyclic Scheduling

2.1 Hard Real-Time Scheduling

Many real-time scheduling algorithms have been presented for various task models and characteristics. The major parameters according to which the scheduling methods can be classified are the possibility of task preemption, task periodicity and criticality of meeting tasks timing requirements and deadline. A comprehensive description of the various scheduling algorithms and their applicable task models is presented by Giorgio C. Buttazzo in [9].

Hard real-time tasks require all their deadlines and timing requirements to be strictly enforced to ensure correct behavior. This needs feasibility tests to be performed prior to run-time in order to guarantee all their timing requirements to be met. The problem of guaranteeing timing constraints in hard real-time systems has received significant attention, however, few techniques have addressed the problem of guaranteeing inter-task temporal dependencies such as relative timing constraints. Most real-time scheduling techniques consider the scheduling of real-time tasks with ready times and deadlines [10, 11, 12, 13, 14, 15, 16, 17]. These constraints impose constant intervals in which a task must be executed. In contrast, in the presence of relative time constraints, the time window within which a task must execute may depend on the scheduling and execution parameters of the other tasks in the system. Some of the systems that consider the problem of scheduling periodic and aperiodic tasks with relative timing constraints were introduced in [18, 6, 7].

Some of the scheduling method that are closely related to the presented work are described below. They all involve scheduling non-preemptive hard real-time tasks.

Static Cyclic Scheduling: Presented by S. Cheng and Ashock Agrawala in [19]. The algorithm studies periodic tasks with release time, deadline, and jitter constraints. It constructs a static calendar for the tasks. The calendar is invoked repeatedly by wrapping around again to its starting point as shown in figure 2.

Parametric Scheduling: Gerber et al. [20] presents a scheduling scheme for aperiodic tasks with relative timing constraints. The algorithm uses Fourier-Motzkin variable elimination technique [8] to calculate a parametric calendar in the off-line scheduler, and uses it to dispatch the task instances at run-time. In this calendar, the start time of each task is presented by two parametric bound functions. A. Mok el al.

in [18] presented a method that uses graph representation of tasks and their relative timing constraints to test the schedulability of a real-time task set.

Dynamic Cyclic Scheduling: Presented by S. Choi et al. in [7, 21]. Extends the parametric scheduling algorithm to periodic tasks with relative timing constraints. The algorithm uses a graph to represent the tasks and their timing constraints. It calculates a cyclic parametric calendar to be used repeatedly to dispatch task instances at run-time.

2.2 Distributed Real-Time Scheduling

The area of real-time scheduling in distributed environment has become an active area of research due to the increasing demand on distributed applications with various QoS and timing requirements. Providing temporal predictability across a network of distributed computing nodes requires the support of QoS-sensitive resource scheduling of the CPU time on the nodes, transfer time in the underlying network, and the network interfaces on the host computers. All scheduling systems has to collaborate to be able to provide end-to-end quality of service guarantees. Some of the major projects involving real-time scheduling in distributed environments are described in the following subsections.

2.2.1 HARTS and ARMADA Projects

The HARTS project, developed in the real-time computing laboratory of University of Michigan, involved the design and implementation of a real-time multi-computer system. The work mainly focused on the hardware and software support for time-constrained communication in point-to-point networks. The project studies real-time communication in multi-hop point-to-point networks [22]. It provides the design and evaluation for a QoS sensitive communication subsystem architecture that is mainly based on the use of real-time channels [23]. A real-time channel is a simplex, ordered, unreliable, virtual connection between two networked hosts that provides deterministic or statistical bound on the end-to-end delay by analyzing the traffic rates and timing requirements on every link on the message delivery route. The network nodes are running a common distributed real-time operating system which is responsible for network control as well as maintaining a global time base by synchronizing clocks on the nodes.

The ARMADA project is mainly the continuation of its predecessor project (HARTS). The goal of this research project is to develop and demonstrate an integrated set of techniques and software tools for designing, implementing, and integrating computation, I/O, or communication intensive embedded real-time application on a parallel or distributed environments. The main methodology to achieve this goal is the development of modular and composable middleware services for constructing distributed real-time applications on a standard RTOS like Mach-RT from the open software foundation (OSF). The ARMADA project inherits the real-time communication architecture from the HARTS project and also uses a fault-tolerant real-time multicast communication service (RTCAST) [24]. The RTCAST method supports bounded time message transport by simulating a time based token-ring protocol on point-to-point networks. The ARMADA project is currently under development in University of Michigan and Honeywell.

2.2.2 EPIQ Project

Developed at University of Illinois at Urbana-Champaign, the EPIQ project was designed with the meta-computing framework in mind. It supports end-to-end quality of service control and resource management strategies. The EPIQ project adopts an open environment for real-time applications, which allows for the applications to be developed and validated independent of each other and configured dynamically to run on the same platform. This scheduler analyzes the schedulability of an application based on the assumption that it runs alone on a processor with a speed that is a fraction of the speed of the target processor. The key component of the open system is the two level hierarchical scheduler, which consists of an OS scheduler responsible for dispatching the processor to the different applications and a second layer of server schedulers, one for each application, which are responsible for scheduling the different tasks and threads within each application according to its specified scheduling algorithm [25].

In order to provide end-to-end QoS guarantees in a networked environment, the EPIQ project extends the Fast Messages (FM) high performance network software model developed in University of Illinois to support predictable performance in terms of deterministic latencies and guaranteed bandwidth. The FM-QoS model incorporates feedback-based synchronization (FBS) of senders and self-synchronizing communication schedules to avoid resource conflicts for network links and outputs. Elimination of such resource conflicts leads to predictable communication performance. FM-QoS uses a Petri net model to characterize the structure of the self organizing schedules and to tolerate the clock drifts [26].

3 Problem Description

All the existing variations of the parametric time-based scheduling method are based on a *single node model*. They mainly focus on non-preemptive periodic/aperiodic hard real-time tasks with inter-task relative timing constraints. Our basic objective is to extend the single node parametric time-based scheduling method to be used with a *distributed* hard real-time task set with inter-task relative timing and communication constraints. This distributed algorithm is then used as a basis to develop a complete time-based scheduling and dispatching model for a distributed set of hard/soft real-time tasks. In order to develop such model, several sub-problems have to be addressed:

- Defining the task and network model.
- Defining schedulability conditions to achieve global, network, and single node local schedulability.
- Designing off-line algorithms for verifying these schedulability conditions.
- Proving the correctness of the schedulability conditions and verification algorithms.
- Developing a time-based dispatching mechanism to ensure the correct timely execution of the real-time tasks.

To better understand the distributed time-based parametric scheduling problem, we present the task model under consideration followed by the model description for the network that connects the distributed computing nodes.

3.1 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 periodic non-preemptive hard real time tasks. The least common multiple (LCM) of tasks periods on all the 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 $1 <= m <= M$. 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 jth scheduling window $[(j-1)L, jL]$ on node m. Jobs are non-preemptively executed in the given order for every scheduling cycle. The execution order for this job set is predetermined, and enforced by order timing constraints. The set of tasks to be dispatched on all nodes in the jth 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). These parameters are:

1. Task period P
2. Low jitter λ
3. High jitter η

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:

1. Start time $s_{i,m}^j$
2. Execution time $e_{i,m}^j$
3. Finish time $f_{i,m}^j$
4. Minimum execution time $l_{i,m}^j$
5. Maximum execution time $u_{i,m}^j$
6. Release time $r_{i,m}^j$
7. Deadline $d_{i,m}^j$

The values of some of the parameters vary according to the runtime behavior of the task, such as start-time, execution time, and finish time. The rest of the task parameters are constants for each job and are determined prior to the schedulability test phase.

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 or upper bounds on the time period between the two variables. A relative timing constraint involving only two time variables is referred to as *Standard*. A standard relative timing constraint can be defined as follows.

Definition 3.1 (Standard Constraints) *A standard constraint involves the variables of at most two jobs running on the same node, $\tau_{a,m}^j$ and $\tau_{b,m}^l (1 \leq a \leq b \leq N_m, \mid j - l \mid \leq 1)$, where $s_{a,m}^j$(or $s_{a,m}^j + e_{a,m}^j$) appears on one side of "$\leq$", and $s_{b,m}^l$(or $s_{b,m}^l + e_{b,m}^l$) appears on the other side of the "$\leq$". For the two jobs, $\tau_{a,m}^j$, $\tau_{b,m}^l$, the following constraints are permitted(where c_i is an arbitrary constant) and called* relative *standard constraints (the node number m is eliminated in this example for clarity purposes):*

$$
\begin{array}{rclcrcl}
s_a^j - s_b^l & \leq & c_1 & \quad & s_b^l - s_a^j & \leq & c_5 \\
s_a^j - (s_b^l + e_b^l) & \leq & c_2 & & s_b^l - (s_a^j + e_a^j) & \leq & c_6 \\
s_a^j + e_a^j - s_b^l & \leq & c_3 & & s_b^l + e_b^l - s_a^j & \leq & c_7 \\
s_a^j + e_a^j - (s_b^l + e_b^l) & \leq & c_4 & & s_b^l + e_b^l - (s_a^j + e_a^j) & \leq & c_8
\end{array} \tag{1}
$$

In addition, release time and deadline constraints for each job are called Absolute *standard constraints. A job $\tau_{a,m}^j$ has the following absolute constraints:*

$$
c_9 \leq s_{a,m}^j \qquad s_{a,m}^j + e_{a,m}^j \leq c_{10} \tag{2}
$$

Any constraint that can be rewritten in one of the above forms is also considered a standard constraint; e.g., $s_{a,m}^j \geq s_{b,m}^l + e_{b,m}^l - e_{a,}^j + c$ falls into this category [7].

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 3.3).

$$
C = C_1 \cup C_2 \ldots \cup C_M \cup C_c \tag{3}
$$

3.2 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 $Link_{i,j}$. 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. Each link end is connected to a front end processor that performs all the data transfer functionality. Figure 3 shows the point-to-point network model under consideration. The nodes are assumed to maintain synchronized clocks according to a global time-base for the system. The maximum skew between clocks on different nodes is assumed to be very small compared to message transfer delays. An algorithm for synchronizing the distributed nodes' calendars is presented as part of the run-time dispatcher in section 8.2. There exist different schemes for achieving distributed clocks synchronization such as the method presented in [27].

3.2.1 Communication Channels

Between two tasks running on two different nodes, a periodic *communication channel* can be established which can transfer periodic messages from one task to another. 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 the following parameters:

- Source task instance.
- Destination task instance.

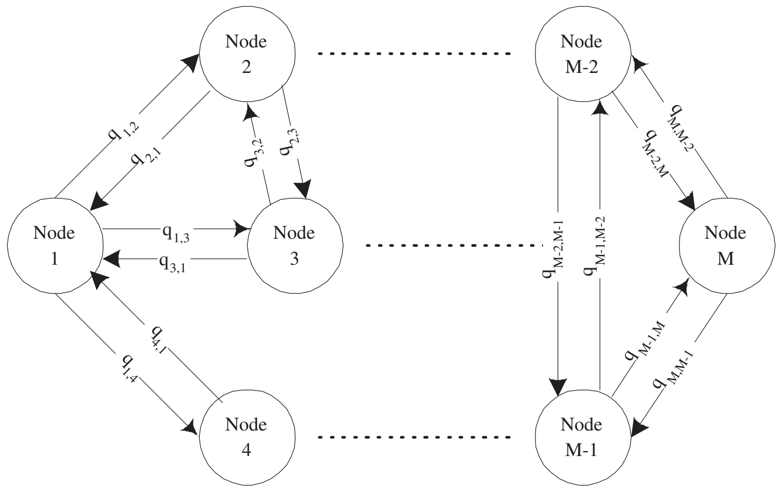

Figure 3: Network Model

- Message generation scheme
  - Maximum message size (S).
  - Maximum message Rate (R).
  - Burst size (B).
- Desired end-to-end maximum message delay (q).

The communication channel imposes an upper bound on the *message delivery time* (Definition 3.2) experienced by each message transferred on this channel. The delay limitations imposed by communication channels on transferred messages is referred to as *Communication constraints* (Definition 3.3). The set of all communication constraints among the M nodes is represented by C_c.

Definition 3.2 (Message Delivery Time (MDT)) *The Message Delivery Time MDT for a communication channel is defined to be the total time elapsed from the time the source job starts sending out the message till the message is completely received by the target job. This is equal to the sum of the following components [28]:*

1. Communication processing time t_C*: Which is the time required for preparing the information for transmission. For example, the time taken to organize data into packets.*
2. Queuing time t_Q*: This is the time spent by the packets waiting in queues for different resources.*
3. Transmission time t_T*: This is the time it takes for the complete information to be transmitted from the source.*
4. Propagation time t_P*: Which is time taken by a single bit in the packet to travel from the communication channel source to the destination.*

Therefore, The overall MDT can be represented as the sum of all these components. The worst case message delivery time MDT_{wc} has to be less than or equal to the maximum delay required for the communication channel q.

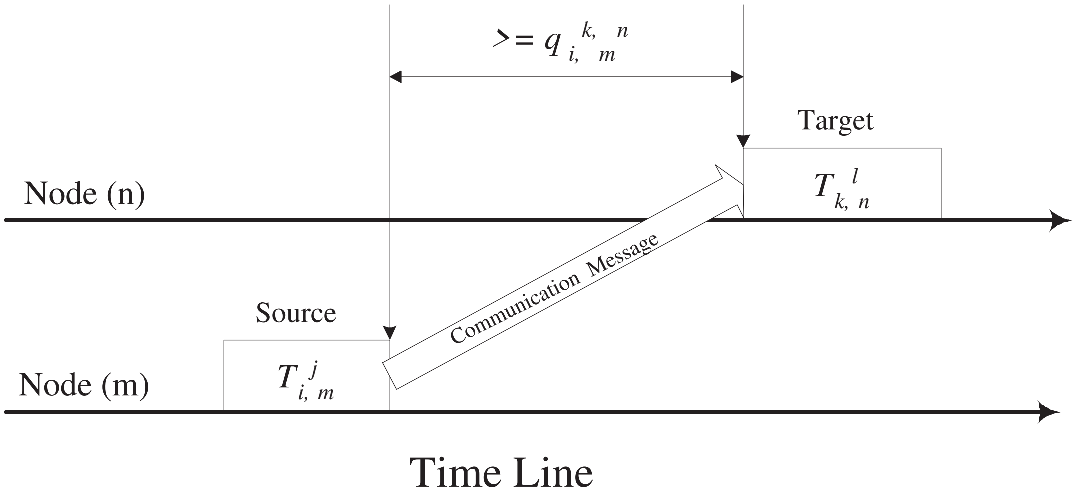

Figure 4: Communication constraints

$$MDT = t_C + t_Q + t_T + t_P \leq MDT_{wc} \leq q$$

Definition 3.3 (Communication Constraints) *A communication constraint is the upper limit* $q_{i,j,m}^{k,l,n}$ *imposed on the delivery time of a message sent over a communication channel established from one job* $\tau_{i,m}^{j}$ *to another job* $\tau_{k,n}^{l}$ *on two different nodes* m, n. *Therefore, a lower limit is imposed on the time distance between the finish-time of the source job* $f_{i,m}^{j}$ *and the start-time of the target job* $s_{k,n}^{l}$ *to accommodate the worst case delivery time. Communication messages are assumed to be periodic. Each message is assumed to be sent at the end of the source job execution, and completely received by the destination node before beginning the execution of the target job. Figure 4 shows a communication constraint on the time-line corresponding to constraint equation 4.*

$$s_{k,n}^{l} - f_{i,m}^{j} \geq q_{i,j,m}^{k,l,n} \tag{4}$$

If a worst case message delivery time MDT_{wc} *can be obtained, it can be used as an upper limit for the communication constraint if it is less than or equal to the original constraint upper limit* q *as shown in equation 5.*

$$s_{k,n}^{l} - f_{i,m}^{j} \geq (MDT_{wc})_{i,j,m}^{k,l,n} \tag{5}$$

4 Schedulability

The Global schedulability of the whole task model is established if and only if we can find starting times for all jobs that will satisfy all timing constraints for all possible execution times. The possible execution time for each task lies between the lower and upper bounds for its execution $[l, u]$. 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.

$$C = C_1 \cup C_2 \ldots \cup C_M \cup C_c \tag{6}$$

The schedulability test predicates are presented in definitions 4.1 through 4.3. The schedulability of a set of N tasks holds if and only if there exist a start time assignment that preserves all required task ordering and timing constraints. Ordering information is normally given as precedence constraints represented as part of the timing constraints set C. Therefore, the necessary and sufficient condition for the task set schedulability ($Sched_1$) is defined in 4.1.

Definition 4.1 (Static Schedulability of Γ [6]) *The set of N tasks Γ is schedulable if and only if the following predicate holds:*

$$Sched_1 \equiv \exists s_i :: \forall e_i \in [l_i, u_i] :: C \ldots \forall i : 1 \leq i \leq N \tag{7}$$

where C is the set of relative timing constraints defined on $\{s_1, e_1, \ldots s_N, e_N\}$.

$Sched_1$ represents the static schedulability condition for a fixed set of aperiodic tasks. The necessary and sufficient schedulability condition for a set of N tasks repeating k times is represented as $Sched_2$ and defined in 4.2.

Definition 4.2 (Static Schedulability of $\Gamma^{1,k}$) *The $k-fold$ distributed set of N tasks Γ is schedulable if and only if the following predicate holds:*

$$Sched_2^{1,k} \equiv \exists s_{i,m}^j :: \forall e_{i,m}^j \in [l_{i,m}^j, u_{i,m}^j] :: C^{1,k} \ldots [\forall i : 1 \leq i \leq N, \forall j : 1 \leq j \leq k, \forall m : 1 \leq m \leq M] \tag{8}$$

where $C^{1,k}$ is the set of relative timing constraints defined on $\{s_1^1, e_1^1, \ldots s_N^k, e_N^k\}$.

Static schedulability offers simpler off-line temporal correctness verification of the task set as well as faster run-time dispatching which, merely needs to do table look up to figure out the next task instance to be dispatched and its dispatch time. The drawback of the static approach, is that it doesn't account for variations in run-time behaviour of various tasks and uses their worst case execution time in the correctness analysis which leads to inefficiency in the resulting dispatching calendars. Parametric scheduling introduced by [6, 7] generalizes static scheduling by deferring the actual start-time calculation process to the run-time dispatcher, which can use the actual execution times of the previous tasks in the dispatching process.

The parametric schedulability condition of a distributed set of N tasks repeating k times is represented by $Sched_3$ defined in 4.3. The steady state schedulability of a set of N periodic tasks repeating indefinitely can be established by testing $Sched_3^{1,k}$ for large values of k, specifically as $k \rightarrow \infty$. The steady state correctness verification predicate $Sched^{1,\infty}$ is defined in 4.4.

Definition 4.3 (Parametric Schedulability of $\Gamma^{1,k}$) *The $k-fold$ distributed set of N tasks Γ is schedulable with respect to parametric scheduling if the following predicate holds:*

$$
\begin{array}{l}
Sched_3^{1,k} \equiv \\
\exists s_{1,1}^1 :: \forall e_{1,1}^1 \in [l_{1,1}^1, u_{1,1}^1] :: \exists s_{2,1}^1 :: \forall e_{2,1}^1 \in [l_{2,1}^1, u_{2,1}^1] :: \ldots :: \exists s_{N_1,1}^k :: \forall e_{N_1,1}^k \in [l_{N_1,1}^k, u_{N_1,1}^k] :: C_1^{1,k} \\
\wedge \\
\vdots \\
\wedge \\
\exists s_{1,m}^1 :: \forall e_{1,m}^1 \in [l_{1,m}^1, u_{1,m}^1] :: \ldots :: \exists s_{N_m,m}^k :: \forall e_{N_m,m}^k \in [l_{N_m,m}^k, u_{N_m,m}^k] :: C_m^{1,k} \\
\wedge \\
\vdots \\
\wedge \\
\exists s_{1,M}^1 :: \forall e_{1,M}^1 \in [l_{1,M}^1, u_{1,M}^1] :: \ldots :: \exists s_{N_M,M}^k :: \forall e_{N_M,M}^k \in [l_{N_M,M}^k, u_{N_M,M}^k] :: C_M^{1,k} \\
\wedge \\
C_c
\end{array}
\tag{9}
$$

Definition 4.4 (Parametric Schedulability of $\Gamma^{1,\infty}$ [7]) *The periodic task set $\Gamma^{1,\infty}$ is schedulable if and only if*

$$Sched^{1,\infty} = \lim_{k\to\infty} Sched_3^{1,k} = True \tag{10}$$

The terms $Sched^{1,\infty}$ and $Sched$ will be used interchangeably to represent the global schedulability of the distributed constrained task set.

5 Distributed Dynamic Scheduling

This section introduces the framework for establishing the global schedulability of the distributed task model described in section 3.1, and creating a separate dynamic parametric calendar for each of the distributed nodes in the distributed system that satisfies all system timing and communication constraints.

5.1 Schedulability Verification

The schedulability verification process is performed off-line and assumes that the timing constraints and communication parameters of all hard real-time tasks are known prior to runtime. In case the global schedulability of the system is proven, the algorithm produces dynamic calendars which are then used by the on-line dynamic dispatcher described in section 8 to control the dispatching and execution of all hard real-time tasks according to their specified timing requirements. The schedulability verification process steps are described in the following subsections.

5.1.1 Communication feasibility

Establishes the feasibility of all the communication channels to be established between periodic real-time tasks residing on different nodes. This is performed by means of *Real Time channels* [22], a method for establishing time-constrained communication in multi-hop networks.

This process starts by calculating the optimum static route $\{Node_s, Node_1, ..., Node_n, Node_t\}$ for each one of the communication channels. The messages of the communication channels are added to the message-table

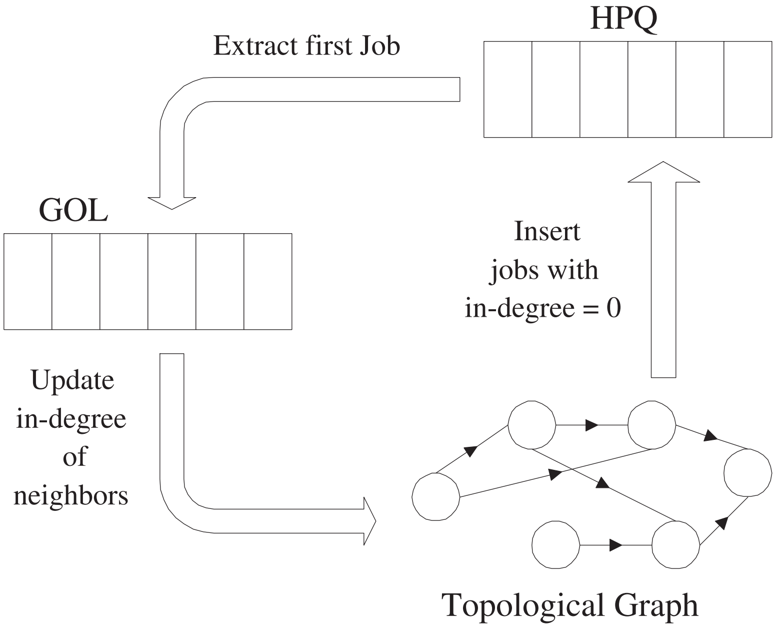

Figure 5: Global task order generation

of each one of the point-to-point links along the calculated route $\{Link_{s,1}, ..., Link_{n,t}\}$. Using the Message generation scheme [Maximum message size (S), Maximum message Rate (R), and Burst size (B)], the Real Time channels method is used for each point-to-point link to establish the feasibility of the messages passing through each link in the communication network, and calculating the worst case delay each message experiences to pass through the link. By adding the message delays on all the communication channel route links, the end-to-end worst case message delivery time $(MDT_{wc})_s^t$ experienced by messages of the communication channel is obtained. If the channel's calculated worst case message delivery time is less than or equal to the message delivery time upper limit q_s^t required by the communication channel, then this channel is feasible, otherwise, it is not.

Total communication feasibility of the system is established if all the system communication channels as well as point-to-point network links are feasible.

The detailed steps for the communication feasibility verification process are shown in step (1) of the *Dynamic Off-Line Scheduler* algorithm 5.3.

5.1.2 Global task order

The *Global task order* is a single ordered list containing all the distributed task instances (jobs) in the system. Jobs in this list are ordered in the reverse order in which they will be eliminated using the variable elimination algorithm used to verify the global system feasibility.

The Global task order maintains all job orders specified by system precedence constraints, relative timing constraints, jitter constraints, as well as all communication constraints. A heuristic is used to insert jobs in the Global order list when system timing requirements does not uniquely identify a specific order. Some of the heuristics used are Earliest Deadline First (EDF), Earliest Ready-time First (ERF), Least Laxity First (LLF), or Rate Monotonic (RM).

In order to be able to apply the variable elimination technique, jobs need to be ordered such that the start time of each job depends only on the preceding jobs in the global order. In the same time, we need to maintain each node's local jobs order. To achieve the required global order, we construct a *Topological graph* which consists of $2N$ nodes representing all jobs in two consecutive scheduling windows and *Precedence links* which specify that the job represented by the source node of the link must precede the job represented by the target node of the same link.

To generate the Global task order, the in-degree (number of links entering a graph node) of each job node is calculated. The jobs with in-degree equal to 0 are inserted into a priority queue ordered according to the heuristic to be used (Heuristic Priority Queue HPQ). We repeatedly extract the job at the head of the HPQ, add it to the Global Order List (GOL), decrement the in-degree of all the target nodes of precedence links originating from the extracted job, and insert the nodes with in-degree equal to 0 into the HPQ. The process is stopped once there is no more jobs in the HPQ. The global task order generation process is illustrated in figure 5.

If the GOL does not contain all the system jobs, then the Global task order cannot be constructed due to circular dependencies in the system timing requirements, and therefore, the system is *infeasible.*

The detailed steps for constructing the *Global task order* are presented in algorithm 5.1.

Algorithm 5.1 (Global Task Order) *The algorithm for finding the Global Jobs Order is as follows:*

1. *Create a node in the Topological graph for each job in two consecutive scheduling windows of the task set.*
2. *Create a Precedence link from each node to the node directly after it in its local node job order.*
3. *For (each communication channel)* {
   - *(3.1) Create a Precedence link from the node representing the source job of the communication channel to the channel's target job.*

   }
4. *Calculate the in-degree for each job node in the Topological graph.*
5. *Insert jobs with in-degree=0 into the Heuristic Priority Queue (HPQ).*
6. *While (HPQ is not empty) do* {
   - *(6.1) Extract first job in the HPQ.*
   - *(6.2) Insert job into the Global Order List (GOL).*
   - *(6.3) For (each precedence link originating at the extracted job)* {
     - *6.3.1. Decrement the in-degree of the target job of the Precedence link by 1.*
     - *6.3.2. If job's in-degree is equal to 0, insert job into the HPQ.*

     }

   }
7. *If an order containing all the system jobs is found, it is used as the Global order for the variable elimination process.*
8. *Else, Task set timing constraints contain circular dependencies and the system is* **not feasible.**

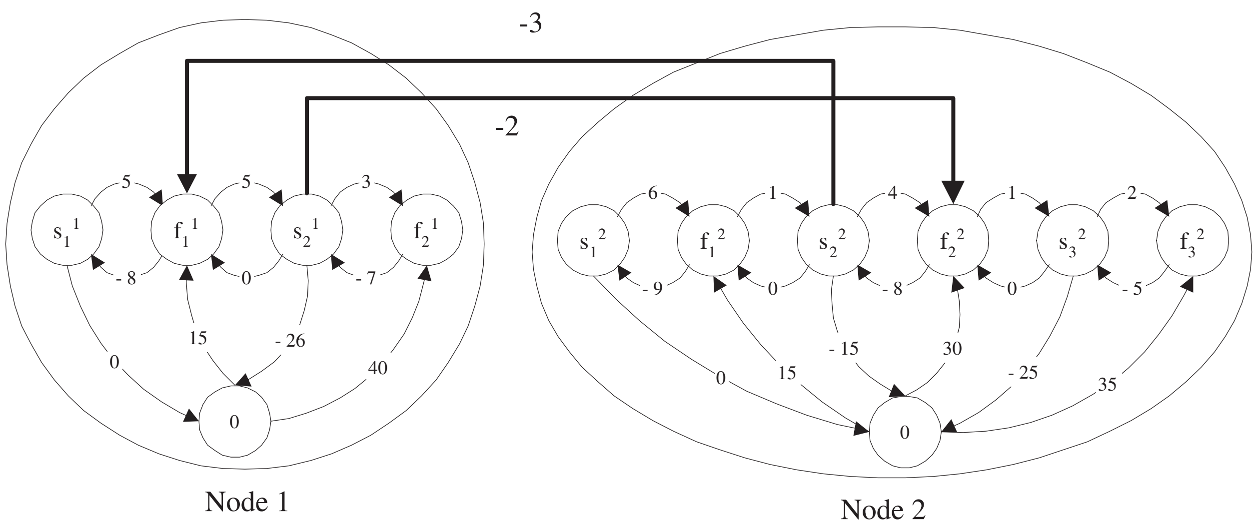

Figure 6: Distributed timing constraints

5.1.3 Global constraint graph G_g

The next step in the schedulability verification process is to construct the *Global Constraint Graph*, which represents all the system task instances, absolute and relative timing requirements, as well as communication constraints of the system in two consecutive scheduling windows. An example of the graph representation of the system model is shown in figure 6.

The global constraint graph G_g consists of $4N+1$ time event nodes representing the task model jobs. Each job is represented by four time event nodes in the graph $[s^j_{i,m}, f^j_{i,m}, s^{j+1}_{i,m}, f^{j+1}_{i,m}]$ which represent the start and finish times for that job in two consecutive scheduling windows. A v_0 node is added to the graph to represent the reference time t_0 starting from which all time values are measured.

The graph weighted links represent the local relative timing constraints among the jobs if they link two time events that belong to jobs on the same system node, and they represent communication constraints if they link time events belonging to two jobs on separate system nodes. Links to or from the v_0 node represent absolute timing constraints such as the ready-time or deadline of a specific task instance. Timing constraints between jobs within one scheduling window are repeated in the two represented scheduling windows. Timing constraints between jobs in different scheduling windows is represented once between the first and second scheduling windows. The different types of timing constraints are represented in the graph as shown in figure 7 according to the following rules:

- **Minimum/Maximum execution times** are represented by two mutually exclusive links between the start time and finish time events of the same job. The minimum execution time is represented by a link from the job's start time node s to the job's finish time node with a weight l. The maximum execution time is represented by a link in the reverse direction with a weight $-u$.
- **Relative timing constraints** are represented in the graph by a single link for each constraint. A constraint $v_2 - v_1 \leq w$ is represented by a link from event node v_1 to event node v_2 with a weight w.
- **Ready-times and deadlines** are absolute timing constraints and are therefore represented by links to/from the time reference node v_0. The ready time r is represented by a link from the job's start time

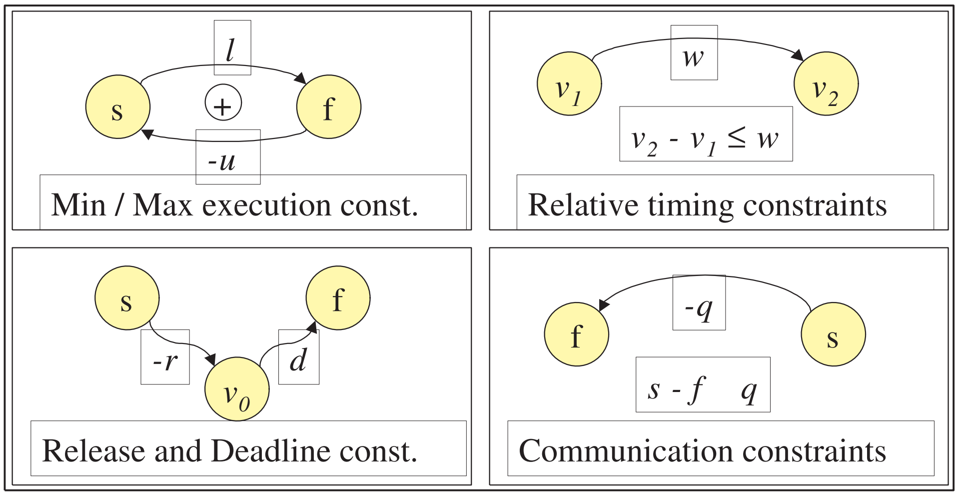

Figure 7: Graphical representation of timing constraints

node s to the v_0 node with a weight $-r$, and the deadline is represented by a link from the v_0 node to the finish time node f of the same job with a weight d.

- **Communication constraints** are represented by a link from the target job's start time node s_t to the target job's finish time node f_s. The link weight is the negative value of the worst case message delay required by the communication constraint $-q$.

A detailed description of the rules to construct a graph representation of hard real-time tasks with inter-task timing constraints is presented in details in [7].

5.1.4 Global system feasibility

After constructing the Global constraint graph, it is used in this step by the *Cyclic Variable Elimination Method* [7] to verify the global feasibility of the timing requirements of all the system tasks.

The Method repeatedly eliminates the time event nodes in the second scheduling window of the constraint graph. The elimination process usually results in changes in the timing constraints of the first graph scheduling window. These updates are copied to the second scheduling window, after it has been re-constructed, and the elimination process of the second scheduling window nodes is repeated.

If an infeasibility condition is detected after the elimination of any time event node, the process is stopped, and the system is declared **infeasible**. The infeasibility condition arises when a negative weight cycle is generated after the elimination of a time event node.

The system is **schedulable** if the elimination process reaches a steady state in which no updates are introduced to the graph after eliminating all the second scheduling window time event nodes. If the maximum number of iterations is reached before the Cyclic Variable Elimination process iterations reach a steady state, it is concluded that the system task set with the given timing constraints are **not schedulable**. The maximum number of iterations is $n^2 - n + 2$, where n is the number of jobs in the first scheduling window that have timing constraints with jobs in the second scheduling window.

The detailed steps for the Distributed Cyclic Variable Elimination Method are described in algorithm 5.2.

Algorithm 5.2 (Cyclic Variable Elimination technique)

1. *Initialize Global constraint graph* G_g.
2. *Let* $iteration = 0$.
3. *Let* n = *number of jobs in the first scheduling window that have timing constraints with jobs in the second scheduling window.*
4. *While* $(iteration < n^2 - n + 2)$ *do* {

   (4.1) *For* $(j = 2N$ *to* $1)$ *do* {

   4.1.1. *Eliminate graph event node* j *from second scheduling window.*

   4.1.2. *If (resulting graph contains a* negative weight cycle*)* {
   - *Task set is* **not feasible**
   - **Exit**

   }

   }

   (4.2) *If (New updates were generated in the graph's first scheduling window as a result of the elimination process)* {

   4.2.1. *Reconstruct graph second scheduling window.*

   4.2.2. *Add elimination graph updates to the second scheduling window.*

   4.2.3. $iteration = iteration + 1$

   }

   (4.3) *Else* {

   4.3.1. *Task set is* **feasible**

   4.3.2. **Exit**

   }

   }
5. *Task set is* **not feasible**
6. **Exit**

5.2 Local node calendars

In the distributed environment under consideration, each node has to operate independently and interact with the tasks running on other nodes only through the communication channels specified. Therefore, we calculate a separate cyclic dynamic calendar for each one of the distributed nodes. Each of the local node calendars must have all the information required to dispatch the local real-time jobs according to their timing constraints, and in the same time, adhere to the global system feasibility and communication timing constraints.

To construct local node calendars, a separate local constraint graph is built for each one of the nodes using only its local jobs and local timing constraints as described in section 5.1.3. By applying the Cyclic Variable Elimination Method on the local constraint graph, the local parametric calendar for the node is generated. The node calendar consists of two parametric functions for each one of the jobs, minimum start time $\mathcal{F}^{min}()$ and maximum start time $\mathcal{F}^{max}()$. The communication timing constraints are then appended to the generated local dynamic calendars by adding the arrival time of the communication messages as parameters to the minimum start time functions of jobs that are targets of communication channels. The resulting parametric calendar is then used by the dynamic dispatcher to start the execution of hard real-time jobs according to their timing constraints as described in section 8.

Algorithm 5.3 (Dynamic Off-Line Scheduler) *The Off-line scheduling and calendar calculation is performed by the following algorithm:*

1. **Communication feasibility:**
   - *(1.1) For (every communication channel)* {
     - *1.1.1. Calculate the optimum rout*
     - *1.1.2. Add the communication channel messages to the message tables of all the links on channel rout*

     }
   - *(1.2) For (Every network link)* {
     - *1.2.1. Check the feasibility of this link using the real-time channel method*
     - *1.2.2. If (any link is not feasible)* {
       - *1.2.2.1. Record infeasible link*
       - *1.2.2.2. Real-time communication is* **not feasible**
       - *1.2.2.3.* **Exit**

       }
     - *1.2.3. Else* {
       - *1.2.3.1. Calculate the worst case delay that each message experience passing through this link*

       }

     }
   - *(1.3) For (Every communication channel)* {
     - *1.3.1. Calculate the end-to-end worst case message delivery time* $(MDT_{wc})_s^t$
     - *1.3.2. If* $((MDT_{wc})_s^t \geq q_s^t)$ {
       - *1.3.2.1. Record infeasible communication channel*
       - *1.3.2.2. Real-time communication is* **not feasible**
       - *1.3.2.3.* **Exit**

       }

     }
   - *(1.4) System Real-time communication is* **feasible**
2. **Global constraint graph (G_g):**
   - *(2.1) Create a reference node v_0 representing the global time $t_0 = 0$*
   - *(2.2) For (All source/target nodes of Real-time channels)* {
     - *2.2.1. Let the global node v_0 represent the node's reference time*
     - *2.2.2. For (each job)* {
       - *2.2.2.1. Add four time event nodes to the constraint graph*
         - *Start time in first scheduling window*
         - *Finish time in first scheduling window*
         - *Start time in Second scheduling window*
         - *Finish time in Second scheduling window*

2.2.2.2. Add job's minimum and maximum execution times constraint links to the graph

2.2.2.3. Add job's ready time and deadline constraint links to the graph

}

2.2.3. Add jitter constraints as links between consecutive jobs of the same real-time task

2.2.4. Add relative timing constraints as links in the graph

2.2.5. Add communication constraints to the graph as links from the start time of the target job to the finish time of the source job with a weight equal to the negative value of the worst case message delivery time $(-MDT_{wc})_s^t$ *of the corresponding communication channel*

}

3. **Global system feasibility:**

(3.1) Use algorithm 5.1 to find the Global Jobs Order.

(3.2) If (a global order is not found) {

3.2.1. Task set is **not feasible**

3.2.2. **Exit**

}

(3.3) Apply Cyclic Variable Elimination technique described in algorithm 5.2 to establish the global schedulability of the system

(3.4) If (System is not feasible) {

3.4.1. Mark constraints that lead to infeasibility

3.4.2. Task set is **not feasible**

3.4.3. **Exit**

}

(3.5) Global system is **feasible**

4. **Local node calendars:**

(4.1) For (All nodes) {

4.1.1. Build the local constraint graph for the node using only local jobs and local timing constraints

4.1.2. Apply Cyclic Variable Elimination technique to establish the local schedulability of the node

4.1.3. If (Node is not schedulable) {

- *Mark the node as infeasible for re-evaluation and re-scheduling*

}

4.1.4. Calculate the minimum and maximum start-time parametric functions for each job

4.1.5. For (Each job that is a target of a Communication channel) {

- *Add the arrival-time of the communication message as a parameter to the job's minimum start time parametric function*

}

4.1.6. Store the node's dynamic parametric functions into its local calendar to be used by the run-time dispatcher.

}

6 Algorithm Correctness

To prove the correctness of the distributed scheduling method, we need to establish the correctness of few sub-problems which constitute the total correctness of the main algorithm. Each of the individual sub-problems is introduced separately in one of the following sub-sections.

6.1 Cyclic Variable Elimination and Parametric Scheduling

The use of variable elimination techniques for non-periodic dynamic parametric scheduling has been introduced in [6]. The extension of the parametric scheduling method to include periodic and sporadic tasks in a single-node environment has been presented in section 5.1.4. The algorithm details as well as its proof of correctness are described in [7]. Therefore, we consider the single-node cyclic variable elimination method to be correct, and basically use it as a black-box.

6.2 Real-time Channels Feasibility Testing

Real-time channels is an algorithm used to verify the timing feasibility of communication messages on each network link and calculate the worst case delivery time for each message. The algorithm is described briefly in section 5.1.1. The details of the algorithm as well as the proof of correctness are described in [22]. We also consider this method to be correct and use in a black-box manner as well.

6.3 Global Schedulability Validation

This sub-section establishes the correctness of the global feasibility analysis step performed by the off-line scheduler. In order to verify the global schedulability correctness we need to establish few major points as described in the sub-sections below.

6.3.1 Representation completeness

All system components are represented in the global constraint graph G_g used for schedulability analysis.

Each of the distributed jobs is represented in the global constraint graph by two time event nodes (s, f), and each of the timing constraints is represented in the global constraints graph as follows:

- Ready time of a job is represented by a link from the start time event node s to the reference time node v_0.
- Deadline is represented by link from node v_0 to the finish time event node f of the job.
- Minimum execution time is represented as a link from the start time event node s to the finish time event node f of the job.
- Maximum execution time is represented as a link from the finish time event node f of the job to the start time event node s.
- Low jitter constraint is represented as a link from the finish time of a task instance f_i to the start time of the next instance in the same task s_{i+1}.
- High jitter constraint is represented as a link from the start time of a task instance s_i to the finish time of the previous instance in the same task f_{i-1}.
- Precedence constraints are represented as a link from the finish time of a job f to the start time of the next job s.

- Local relative timing constraints between different jobs are included as links between the appropriate time event nodes of these jobs.
- Communication constraints are included as a link from the finish time event node of the source job f_s to the start time event node of the destination job s_t.

These itemized categories represent all system absolute timing constraints C_A, relative timing constraints C_R and communication constraints C_c. Where:

$$\begin{aligned} C_A &= \textstyle\bigcup_{m=1}^{M} C_{A,m} \\ C_R &= \textstyle\bigcup_{m=1}^{M} C_{R,m} \end{aligned} \tag{11}$$

From equation (6), the complete set of system constraints can be represented as:

$$C = C_1 \cup C_2 \ldots \cup C_M \cup C_c$$

Since:

$$C_m = C_{A,m} \cup C_{R,m} \qquad \forall m = 1 \ldots M$$

Then:

$$C = C_{A,1} \cup C_{R,1} \quad \cup C_{A,2} \cup C_{R,2} \quad \ldots \cup C_{A,M} \cup C_{R,M} \quad \cup C_c$$

Rearranging:

$$C = C_{A,1} \cup C_{A,2} \ldots \cup C_{A,M} \quad \cup C_{R,1} \cup C_{R,2} \ldots \cup C_{R,M} \quad \cup C_c$$

Therefore form equation 11:

$$C = C_A \cup C_R \cup C_c \tag{12}$$

From equation (12), we can conclude that the timing constraints represented in the global constraint graph constitute all the system timing requirements. Therefore, we established that all system jobs and timing requirements are completely represented in the global constraint graph and consequently are included in the global schedulability analysis.

6.3.2 Mapping problem into single-node domain

The global schedulability analysis maps to a cyclic variable elimination problem, whose correctness is already established in ([7, 6]).

The global constraints graph G_g used in the feasibility analysis consists of a set of nodes representing the start and finish times of all the jobs in the task set, a single reference time node v_0, and links to represent absolute and relative timing constraints among the system task instances. Communication constraints are represented by relative timing constraints among the source and destination jobs. The start time of any job instance depends only on the timing characteristics of the previous job instances since the global jobs order topologically sorts jobs according to their timing dependencies. Therefore, we can conclude that the global constraint graph used for the global feasibility analysis represents a valid single-node dynamic scheduling model, on which cyclic variable elimination techniques can be applied.

6.3.3 Mapping schedulability output into distributed domain

The outcome of the single-node cyclic variable elimination problem on the global constraint graph represents the global distributed system schedulability. This can be established based on two assertions:

Positive schedulability assertion: if the single-node problem is schedulable, then the distributed system is also schedulable.

Negative schedulability assertion: If the single-node problem is not schedulable, then the distributed system is not schedulable as well.

Since the global constraint graph on which the single-node cyclic dynamic scheduling method is applied includes:

- All model jobs
- Absolute timing constraints
- Relative timing constraints
- Communication constraints

and the relative order of the jobs is the same as in all local nodes orders, finding a feasible solution to that dynamic scheduling problem guarantees a feasible starting time for each of the system jobs that satisfies all the node's local absolute and relative timing requirements as well as global communication timing requirements. The starting times are guaranteed feasible for all periodic repetitions of system scheduling window (k, $\forall k = 1 \rightarrow \infty$).

$$
\begin{aligned}
&\exists s_1^1 :: \forall e_1^1 \in [l_1^1, u_1^1] :: \exists s_2^1 :: \forall e_2^1 \in [l_2^1, u_2^1] :: \ldots :: \exists s_N^1 :: \forall e_N^1 \in [l_N^1, u_N^1] \\
&\wedge \\
&\vdots \\
&\wedge \\
&\exists s_1^k :: \forall e_1^k \in [l_1^k, u_1^k] :: \exists s_2^k :: \forall e_2^k \in [l_2^k, u_2^k] :: \ldots :: \exists s_N^k :: \forall e_N^k \in [l_N^k, u_N^k] \\
&\wedge \\
&C_A \wedge C_R \wedge C_c \\
&\forall k = 1 \rightarrow \infty
\end{aligned}
\tag{13}
$$

Equation 13 guarantees the satisfaction of all global model timing requirements of the distributed real-time system and consequently establishes its global schedulability. As a result, the *Positive schedulability assertion* is established.

In order to establish the negative schedulability assertion, we assume that the global constraint graph G_g built using the method described in section 5.1.3 (using a specific heuristic like EDF as a secondary sorting criteria for the global task order) was found to be infeasible by the single-node cyclic variable elimination process.

If the negative assertion is not correct, then we can construct a different global constraint graph G'_g that represents all the characteristics of the distributed real-time system, in addition to being feasible with respect to the single-node cyclic variable elimination method. In order for the feasible constraint graph to represent the distributed system, it is required to satisfy the following requirements:

- Contain a single reference time node v_0.
- Include all the system's jobs, with all their absolute timing requirements.
- Satisfy all local nodes job ordering requirements.
- Include all nodes' local precedence and relative timing constraints.
- Include all system's communication constraints.
- Have a global job order that satisfies all system global precedence requirements directly or indirectly resulting from the system's precedence, timing, and communication constraints. This order is to be used by the cyclic variable elimination method.

For the feasible graph G'_g to satisfy the described requirements, it has to be similar to the original global constraint graph G_g except for the global ordering of its jobs. Since the global job order of G'_g has to satisfy all global precedence requirements mandated by the system's timing constraints and consequently, its jobs must be topologically sorted. Therefore we can conclude that the two constraint graphs G_g and G'_g are identical except for the ordering of jobs that do not have any direct or indirect precedence relations among them, and whose relative order is determined using the secondary global ordering heuristic method.

Since the heuristic ordering criteria is used as a secondary criteria after using the topological order, then we can conclude that the jobs ordered using the heuristic criteria are placed in consecutive places within the global task order. And since these jobs do not have any relative timing constraints among them, therefore, they adhere to the job requirements of theorem A.1 presented in appendix A. Consequently, we can conclude that the relative order of jobs ordered using the secondary ordering heuristic method does not affect the final outcome of the Fourier-Motzkin variable elimination process. This conclusion was also enforced by the simulation results generated in section 9.3.

This result indicates that the constraint graph G'_g cannot be feasible while the original graph G_g is infeasible. Consequently, We can conclude that a feasible global constraint graph G'_g that represents the requirements of the distributed system cannot be generated if the original global constraint graph G_g was verified to be infeasible by the cyclic variable elimination method.

By contradiction, we conclude that if the system was declared to be infeasible in the single-node domain, it must be due to infeasibility in the distributed system timing requirements not the mapping procedure and consequently, we establish the *Negative schedulability assertion*, which indicates that if the single-node problem was verified not to be schedulable, then the distributed system is also not schedulable.

By establishing both the positive and the negative schedulability assertions, we can conclude that the distributed system is schedulable if and only if the mapped single-node system is also schedulable. In other words, the outcome of the single-node variable elimination analysis represents the schedulability of the original distributed real-time system.

6.4 Local Calendars Feasibility

This sub-problem presents the feasibility of local calendars calculated for each of the systems' nodes separately, with added communication constraints and its conformance with global schedulability. To establish this point, we need to show that the parametric bound functions in the local calendars satisfy global timing requirements. In other words the local calendars should satisfy local timing constraints as well as inter-node communication constraints.

1. **Local timing constraints**

   Since

   - All local timing constraints were included in constraint graphs used to establish the global feasibility as well as the local calendars.
   - Local order of the jobs in the local constraint graphs is the same as that in the global constraint graph.

   It is clear that local timing constraints that are proven to be locally feasible by the local calendar calculation process are also guaranteed to be globally feasible.

   Therefore, it is concluded that the local calendars satisfy all local timing constraints previously proven feasible in the global schedulability phase.

2. **Communication constraints**

   The minimum start time function $\mathcal{F}_s^{min}()$ of a job is of the form:

   $$\mathcal{F}_s^{min}() = Max(p_1, p_2, p_3, \ldots)$$

   Where, $(p_1, p_2, p_3, \ldots)$ are linear functions of the execution timing parameters of previously executed jobs.

   Therefore, adding the arrival time of a communication message to the minimum start time of the target job guarantees that the start time of that target job s_t is more than or equal to the message arrival time. In other words, the target start time s_t is guaranteed to be larger than the source job finish time f_s by at least the message delivery time MDT, for any feasible value of the MDT that is less than or equal to the worst case message delivery time MDT_{wc}.

   $$s_t \geq f_s + MDT \qquad \forall\, MDT \leq MDT_{wc}$$

   And since the communication channel delay (MDT) is already proven to be less than or equal to the worst case message delivery time MDT_{wc} by the Real-time channels method, which is in turn less than or equal to the maximum channel delay q as established in the communication feasibility verification step (section 5.1.1).

   $$MDT \leq MDT_{wc} \leq q$$

Therefore, it is concluded that the local node calendars guarantee that communication channels' target jobs cannot start before the arrival of their corresponding communication messages which are, in turn, guaranteed by the communication feasibility process to arrive in a feasible time. As a result, all communication timing constraints proven feasible in the global schedulability test are satisfied by the local node calendars.

So far, it has been proven that local calendars satisfy

- Jobs absolute timing requirements
- Relative timing constraints
- Communication timing constraints

which means that each of the nodes' local calendars guarantee a feasible start time for each of its local jobs that conforms to its ready time and deadline, satisfies local node timing requirements, and conforms to the inter-node communication constraints.

$$\begin{array}{l} \exists s^1_{1,m} :: \forall e^1_{1,m} \in [l^1_{1,m}, u^1_{1,m}] :: \ldots :: \exists s^k_{N_m,m} :: \forall e^k_{N_m,m} \in [l^k_{N_m,m}, u^k_{N_m,m}] :: C^{1,k}_m \qquad \forall m = 1 \ldots M \\ \wedge \\ C_c \end{array} \tag{14}$$

By adding equation (14) for all the nodes in the system $(1 \ldots M)$, we get:

$$\begin{array}{l} \exists s^1_{1,1} :: \forall e^1_{1,1} \in [l^1_{1,1}, u^1_{1,1}] :: \exists s^1_{2,1} :: \forall e^1_{2,1} \in [l^1_{2,1}, u^1_{2,1}] :: \ldots :: \exists s^k_{N_1,1} :: \forall e^k_{N_1,1} \in [l^k_{N_1,1}, u^k_{N_1,1}] :: C^{1,k}_1 \\ \wedge \\ \vdots \\ \wedge \\ \exists s^1_{1,m} :: \forall e^1_{1,m} \in [l^1_{1,m}, u^1_{1,m}] :: \ldots :: \exists s^k_{N_m,m} :: \forall e^k_{N_m,m} \in [l^k_{N_m,m}, u^k_{N_m,m}] :: C^{1,k}_m \\ \wedge \\ \vdots \\ \wedge \\ \exists s^1_{1,M} :: \forall e^1_{1,M} \in [l^1_{1,M}, u^1_{1,M}] :: \ldots :: \exists s^k_{N_M,M} :: \forall e^k_{N_M,M} \in [l^k_{N_M,M}, u^k_{N_M,M}] :: C^{1,k}_M \\ \wedge \\ C_c \end{array} \tag{15}$$

Which basically constitutes the Parametric Schedulability condition ($Sched^{1,k}_3$) of a distributed set of tasks $\Gamma^{1,k}$ (definition 4.3). Therefore we conclude that applying local node calendars on all distributed system nodes satisfy global system schedulability requirements.

7 Example

In this example, we demonstrate the application of the variable elimination method on a simple distributed system with two nodes and two communication channels established between them. The example system is represented graphically in figure 6. The timing and communication constraints in the figure translates to the following constraint equations:

$$
\begin{array}{rr}
\textbf{Node1} & \textbf{Node2} \\
e_1^1 \geq 5 & e_1^2 \geq 6 \\
e_1^1 \leq 8 & e_1^2 \leq 9 \\
s_1^1 \geq 0 & s_1^2 \geq 0 \\
s_1^1 + e_1^1 \leq 15 & s_1^2 + e_1^2 \leq 15 \\
e_2^1 \geq 3 & e_2^2 \geq 4 \\
e_2^1 \leq 7 & e_2^2 \leq 8 \\
s_2^1 \geq 26 & s_2^2 \geq 15 \\
s_2^1 + e_2^1 \leq 40 & s_2^2 + e_2^2 \leq 30 \\
s_2^1 - (s_1^1 + e_1^1) \geq 5 & e_3^2 \geq 2 \\
 & e_3^2 \leq 5 \\
 & s_3^2 \geq 25 \\
\textbf{Communication} & s_3^2 + e_3^2 \leq 35 \\
s_2^2 - (s_1^1 + e_1^1) \geq 3 & s_2^2 - (s_1^2 + e_1^2) \geq 1 \\
s_2^1 - (s_2^2 + e_2^2) \geq 2 & s_3^2 - (s_2^2 + e_2^2) \geq 1
\end{array}
\tag{16}
$$

Arranging the jobs according to their timing and precedence constraints. The Earliest Deadline First (EDF) method is used as a secondary sorting heuristic. The global order generated is shown in the following sequence of variables:

$$s_1^1,\quad e_1^1,\quad s_1^2,\quad e_1^2,\quad s_2^2,\quad e_2^2,\quad s_3^2,\quad e_3^2,\quad s_2^1,\quad e_2^1$$

We start by eliminating variables in the reverse order of their global order.

Eliminate e_2^1 (substituted by 7)

$$s_2^1 + e_2^1 \leq 40 \qquad \Longrightarrow \qquad s_2^1 \leq 33$$

Eliminate s_2^1

$$
\begin{array}{rcr}
20 \leq s_2^1 & & \\
s_1^1 + e_1^1 + 5 \leq s_2^1 & & 20 \leq 33 \\
s_2^2 + e_2^2 + 2 \leq s_2^1 & \Longrightarrow & s_1^1 + e_1^1 \leq 28 \\
 & & s_2^2 + e_2^2 \leq 31 \\
s_2^1 \leq 33 & &
\end{array}
$$

Eliminate e_3^2 (substituted by 5)

$$s_3^2 + e_3^2 \leq 35 \qquad \Longrightarrow \qquad s_3^2 \leq 30$$

Eliminate s_3^2

$$\left.\begin{array}{r} 25 \le s_3^2 \\ s_2^2 + e_2^2 + 1 \le s_3^2 \\ \\ s_3^2 \le 30 \end{array}\right. \implies \begin{array}{r} 25 \le 30 \\ s_2^2 + e_2^2 \le 29 \end{array}$$

Eliminate e_2^2 (substituted by 8)

$$s_2^2 + e_2^2 \le 29 \implies s_2^2 \le 21$$

Eliminate s_2^2

$$\left.\begin{array}{r} 15 \le s_2^2 \\ s_1^2 + e_1^2 + 1 \le s_2^2 \\ s_1^1 + e_1^1 + 3 \le s_2^2 \\ \\ s_2^2 \le 21 \end{array}\right. \implies \begin{array}{r} 15 \le 21 \\ s_1^2 + e_1^2 \le 20 \\ s_1^1 + e_1^1 \le 18 \end{array}$$

Eliminate e_1^2 (substituted by 9)

$$s_1^2 + e_1^2 \le 15 \implies s_1^2 \le 6$$

Eliminate s_1^2

$$\left.\begin{array}{r} 0 \le s_1^2 \\ \\ s_1^2 \le 6 \end{array}\right. \implies 0 \le 6$$

Eliminate e_1^1 (substituted by 5)

$$s_1^1 + e_1^1 \le 15 \implies s_1^1 \le 7$$

Eliminate s_1^1

$$\left.\begin{array}{r} 0 \le s_1^1 \\ \\ s_1^1 \le 7 \end{array}\right. \implies 0 \le 7$$

Since no contradictions were introduced in the variable elimination process, we conclude that the global system is **Schedulable**.

8 Dynamic Time-based Dispatching

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.

The dynamic time-based dispatcher processes the information transferred to it from 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. The Dispatcher can also determine the schedulability of new aperiodic real-time tasks introduced to the system at run-time. This is done by moving task instances around in accordance with their parametric functions to preserve total system schedulability. The algorithm to insert an aperiodic task at run-time is described in [7].

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.

$\mathcal{F}_{s_1}^{min}()$	$\leq$	s_1	$\leq$	$\mathcal{F}_{s_1}^{max}()$
$\mathcal{F}_{s_2}^{min}(s_1, e_1)$	$\leq$	s_2	$\leq$	$\mathcal{F}_{s_2}^{max}(s_1, e_1)$
	$\vdots$		$\vdots$	
$\mathcal{F}_{s_N}^{min}(s_1, e_1, s_2, \ldots, s_{N-1}, e_{N-1})$	$\leq$	s_N	$\leq$	$\mathcal{F}_{s_N}^{max}(s_1, e_1, s_2, \ldots, s_{N-1}, e_{N-1})$

Figure 8: Parametric Calendar Structure

8.1 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 calendar consisting of functions used to determine the minimum and maximum feasible bounds on the execution start times for the task instances. The general structure of the parametric calendar generated by the off-line scheduler is shown in figure 8. The Run-time information is stored in the form of a dependency graph of the tasks and their timing properties. The Dynamic Calendar built by the on-line dispatcher has *three* main components:

Dependency Graph (DG) shown in Figure 9, it consists of a graph like structure that contains 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 inter-node task instance dependencies. A separate Dependency graph is constructed for each of the distributed nodes in the system. The Dependency graph is represented as a list of task objects each containing the following information:

- Task ID.
- Execution period.
- Low jitter.
- High jitter.
- A linked list of the task's *instance profiles*, each containing the following information:
  - Instance ID.
  - Minimum execution time.

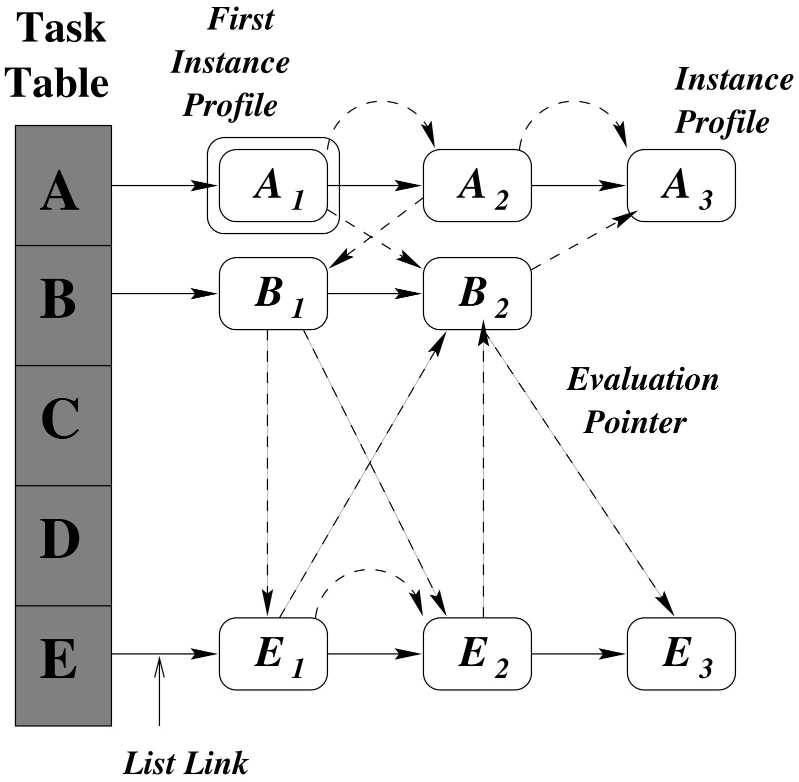

Figure 9: Dependency Graph

- 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 function code, a list of the function parameters, and an *Evaluation counter* for the unresolved parameters in the function.
- *Result lists*, which are lists of pointers *(Evaluation pointers)* to the locations of parameters for the parametric function of other tasks instances. These pointers indicate that timing values from this task instance are the actual parameters for the formal parameters in the other tasks instances functions. A separate list is maintained for each value to be propagated.
- *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, its entries represent task instances that the run-time module have full knowledge about their execution profile. This means that the parameters to 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 times that this task instance can start its execution. It also include 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 following information:

- Message ID.

- Target Task ID.
- Target Instance ID.
- Event arrival time.
- Information about instance function parameters to be substituted by the message arrival time.
- Message application data.

8.2 Run-time Execution Model

The major functionality of the on-line dispatcher is to propagate parameters of 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 consists of three major phases: The *Initialization phase*, the *Calendar synchronization phase*, and the *Task execution phase*.

8.2.1 Initialization phase

The *Initialization phase* of the run-time module starts by processing the calendar information passed by the off-line scheduler in the form of parametric functions. The scheduling information is used at run-time to populate 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 is not dependent on time values generated by the other task instances.

8.2.2 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 (δ_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 10. The Time reference node A starts by measuring the process start time T_{A1} according to its own clock. Next, node A sends a message M_{A1} to node B containing T_{A1} and the message 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 sends a message back M_{B1} to node A containing r_{B1} and the the message send time s_{B1}. Finally, Node A records message M_{B1} arrival time r_{B2}, and sends a third message back M_{A2} to node B containing r_{B2} and the the message send time s_{B2}. At this moment, each one of the two nodes can calculate three time intervals $\delta_1, \delta_2, and \delta_3$ according to equation set 17. 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.

$$\begin{aligned} \delta_1 &= s_{A1} - T_{A1} \\ \delta_2 &= s_{B1} - r_{B1} \\ \delta_3 &= s_{A2} - r_{A2} \end{aligned} \tag{17}$$

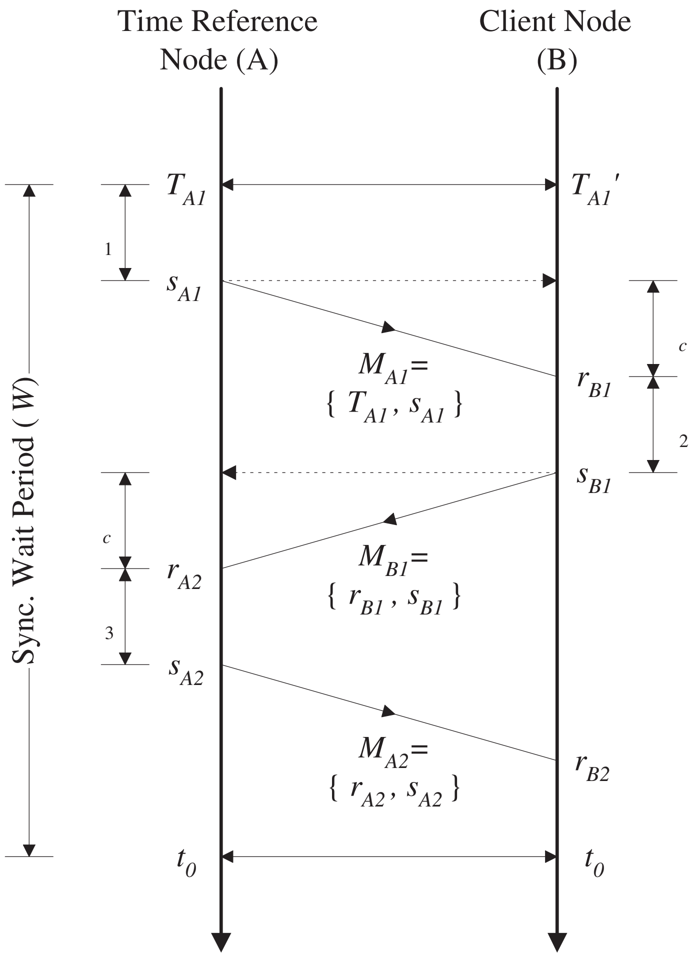

Figure 10: Calendar synchronization phase

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 δ_c according to equation 18. Using the δ_c value, node B can calculate its equivalent version of the time instance T_{A1} according to its clock T'_{A1} using equation 19.

$$\delta_c = \frac{r_{B2} - s_{B1} - \delta_3}{2} \tag{18}$$

$$T'_{A1} = s_{B1} - \delta_2 - \delta_1 - \delta_c \tag{19}$$

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:

$$t_0 = T_{A1} + W$$

The steps of the calendar synchronization process for the Time Reference node are described in algorithm 8.1. The steps for a client node are described in algorithm 8.2.

Algorithm 8.1 (Calendar synchronization for Time Reference node)

1. *Let* $T_{A1} =$ *Current time.*
2. *For* $(M-1$ *client nodes) do* {
   - *(2.1) Send a synchronization message* M_{A1} *containing* T_{A1} *and the message send time* s_{A1}.

   }
3. *For* $(M-1$ *client nodes) do* {
   - *(3.1) Wait for synchronization message* M_{B1} *from client node.*
   - *(3.2) Record message arrival time* r_{A2}.
   - *(3.3) Send a second synchronization message to the client node containing* r_{A2} *and the message send time* s_{A2}.

   }
4. *Let* $t_0 = T_{A1} + W$.
5. *Sleep for* ($t_0 -$ *Current time) time units.*
6. *Start Local calendar execution phase.*

Algorithm 8.2 (Calendar synchronization for Client node)

1. *Wait for synchronization message* M_{A1} *from the time reference node.*
2. *Record the message arrival time* r_{B1}.
3. *Send a message back to the time reference node containing* r_{B1} *and the message send time* s_{B1}.
4. *Wait for a second synchronization message* M_{A2} *from the time reference node.*
5. *Record the second message arrival time* r_{B2}.
6. *Calculate* $\delta_1, \delta_2,$ *and* δ_3 *according to equation set 17.*
7. *Calculate synchronization message delivery time:*
   $\delta_c = (r_{B2} - s_{B1} - \delta_3)/2$
8. *Calculate synchronization process starting time according to local clock:*
   $T'_{A1} = s_{B1} - \delta_2 - \delta_1 - \delta_c$
9. *Let* $t_0 = T'_{A1} + W$.
10. *Sleep for* ($t_0 -$ *Current time) time units.*
11. *Start Local calendar execution phase.*

8.2.3 Task execution phase

At the unified calendar start reference time t_0, the run-time dispatcher extracts the first task instance in the 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.

After the current task instance finishes execution, kernel gains control again, it starts by propagating the timing information generated from the finished task instance to all the function parameters that are dependent on these values using the results lists of these values in the task instance profile. The kernel then inspects the external event queue (EEQ) and delivers all the messages in the queue to their destination task instances. The dispatcher maintains the unresolved parameter counter for the task instance parametric functions to which the parameters were propagated. If 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 these 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 $O(N)$, were N is the total number of task instances in one scheduling window.

The main steps for the On-line dispatcher is shown in the following algorithm.

Algorithm 8.3 (On-Line Dispatcher) *The on-line dispatching of the hard real-time jobs is performed by the following algorithm:*

1. *Populate the Dependency graph using the tasks parametric functions generated by the off-line scheduler.*
2. *If (Time reference node)* {
   - *Run the time reference node calendar synchronization process as described in algorithm 8.1.*

   } *Else* {
   - *Run the client node calendar synchronization process as described in algorithm 8.2.*

   }
3. *Insert the first task instance in the TOL.*
4. *While (TOL not empty)* {

   (4.1) Get first task instance in TOL (I_{top}).

   (4.2) Calculate actual starting time of instance $s_{top} = Currenttime$.

   (4.3) Schedule a time interrupt to occur immediately after $s_{top} + WCET(I_{top})$.

   (4.4) Yield control to I_{top}.

   (4.5) When I_{top} finishes or the scheduled interrupt occurs

   4.5.1. Stop the execution of I_{top} if it is still running.

   4.5.2. Record its finishing time $f_{I_{top}}$.

   4.5.3. Substitute the start time s_{top} in all items in its evaluation list.

4.5.4. Decrement the evaluation counters of all the elements on the evaluation list of s_{top}.
4.5.5. Substitute the finish time f_{top} in all items in its evaluation list.
4.5.6. Decrement the evaluation counters of all the elements on the evaluation list of f_{top}.
4.5.7. while (EEQ not empty) {

- *Get first message it the EEQ (M_{top}).*
- *Substitute the arrival time of M_{top} in its target instance parametric functions.*
- *Decrement the evaluation counter of target instance.*

}

4.5.8. If the evaluation counters of any instance reaches zero, then {

- *Insert this instance in TOL.*
- *Decrement its activation counter by 1, if it reaches 0, the instance is removed from the dependency graph.*
- *Restore all its evaluation counters to their initial values.*

}

}

9 Implementation and Results

The goals of the experiments conducted on the distributed dynamic scheduling and dispatching model are:

- Verify the correctness and completeness of the scheduling and dispatching methods, and determine all the fine details required for the presented model to be used as a complete distributed real-time scheduling and dispatching environment.
- Investigate the effect of varying the heuristic used as a secondary sorting criteria to generate the global task order on the schedulability verification process.
- Measure the impact of varying the different system parameters such as the number of tasks or the average tasks execution time on the parametric schedulability of a real-time system.

The following subsections describe the implementation and simulation experiments conducted to achieve the previous goals. The results of these experiments follows along with a discussion of the conclusions derived from them.

9.1 Implementation

The dynamic time-based scheduler is implemented as a central object. The scheduler object provides methods to add system nodes, communication links, tasks, relative timing constraints, precedence constraints, and communication constraints. After all the real-time system components have been entered, the feasibility verification process can be started. The scheduler also provides methods for the distributed node dispatchers to query system parameters and to retrieve their local run-time parametric calendars. The run-time calendars are available only if the system is concluded to be schedulable.

The run-time dispatchers are implemented as multiple objects, one for each of the system nodes. Each dispatcher object starts its initialization phase by retrieving its dynamic calendar from the scheduler and

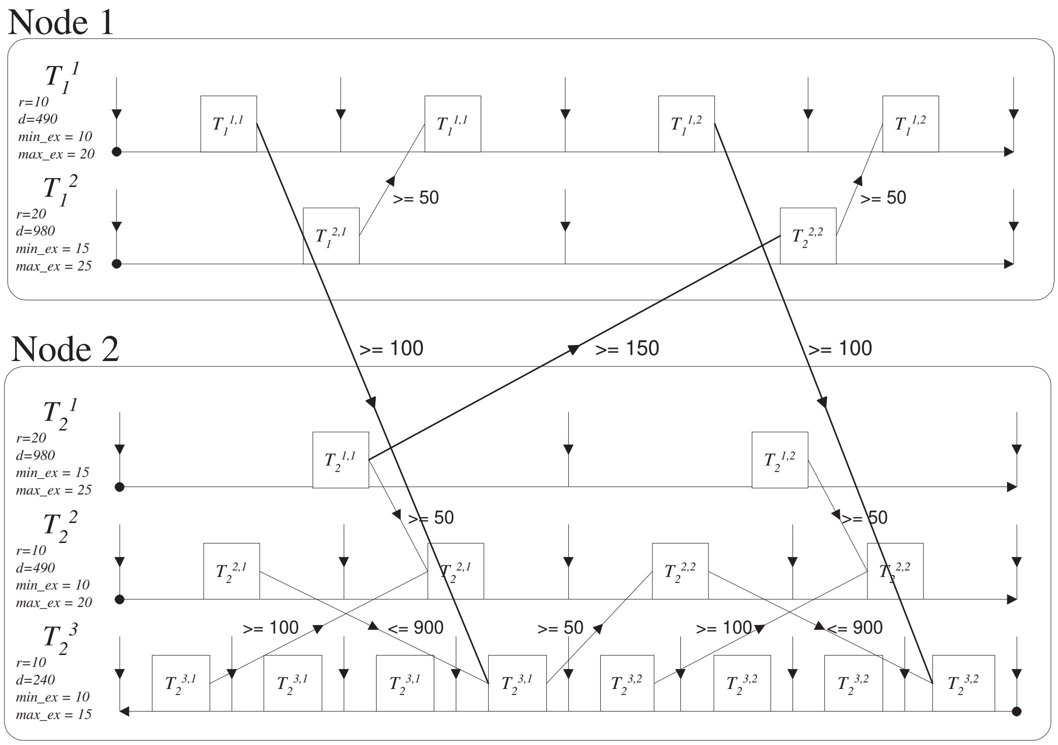

Figure 11: Simulation system configuration diagram

initializing its run-time data structures. The task execution phase starts by executing the calendar synchronization process which unifies the starting time for the execution of all node calendars. The node dispatchers start dispatching the first task instance at the reference time $t_0 = 0$ and continue as described in section 8.2.3.

The scheduler and dispatcher objects are used to implement an example system that consists of two nodes connected using a bidirectional link. The example system's configuration, task structure, and timing constraints are shown in figure 11. The example system feasibility was verified in the off-line analysis phase using the scheduler object that generated two separate node calendars, which were saved into node calendar configuration files. Two dispatcher objects were run on two Windows NT machines, and initialized with their corresponding calendar configuration files. They start by the clock synchronization process, and at the reference time $t_0 = 0$ they start dispatching the local jobs using their dynamic calendars. Communication channels were simulated using socket connections over an Ethernet network. The system tasks were dummy tasks that announced their existence, consumed a random CPU time (Limited by their minimum and maximum execution times), and finally generate a communication message to their target if they are the source of a communication channel.

9.2 Experiments Description

In order to test the effect of parameter variation on the schedulability verification procedure, few simulation experiments were conducted. The experiments are based on generating a random set of distributed real-time tasks. Absolute as well as relative timing constraints among the tasks are randomly generated as well. The task set is generated using the following steps:

Truncated Normal Distribution: Most of the timing parameters for the tasks and constraints are generated using a truncated Normal distribution. The distribution is specified by the minimum truncation point, maximum truncation point, mean μ, and standard deviation σ. The mean μ is considered to be the median point between the minimum and maximum truncation points. The parameter used to control the time intervals generation in the conducted experiments is the Normal distribution standard deviation to mean ratio σ/μ.

Nodes generation: The simulation environment consists of M nodes that form a fully connected network with all links having the same communication delay.

Tasks generation: A set of N tasks are generated. The tasks timing parameters are generated according to the following rules:

- **Period (P):** The task repetition period is selected randomly from a pre-specified ordered set of periods. The period set consists of n_p entries which are generated by assigning the first period in the set a pre-specified time period, and calculating each subsequent period in the set to be equal to half of its predecessor. The number of entries in the period set n_p is referred to as *Number of Period Levels*.
- **Execution Node:** Randomly selected from the M distributed nodes.
- **Jitter constraints (λ, η):** Jitter constraints are randomly generated using a truncated Normal distribution. The minimum and maximum truncation points are calculated as percentages from the task's period, J_{min} and J_{max} respectively.
- **Ready time (r):** Generated as a truncated Normally distributed percentage of the task's period with truncation percentages r_{min} and r_{max}.
- **Deadline (d):** Generated as a truncated Normally distributed percentage of the task's period with truncation percentages d_{min} and d_{max}.
- **Minimum execution time (l):** Generated as a truncated Normally distributed percentage of the task's period with truncation percentages l_{min} and l_{max}.
- **Maximum execution time (u):** Generated as a truncated Normally distributed percentage of the task's period with truncation percentages u_{min} and u_{max}.

Relative timing constraints: N_{rc} relative timing constraints are generated at random in each experiment according to the following guidelines:

- **Node:** The relative timing constraint constraint is assigned to a randomly selected node out of the M system nodes.
- **Source and destination jobs:** The source and target task instances are two randomly selected jobs that reside on the same node. The source and target jobs are not allowed to be the same. The earlier of the two jobs is considered to be the source of the timing constraint. However, since the actual start times of jobs are not known until run-time, the heuristic used as the secondary ordering criteria in the global order generation process is also used to decide which of the two jobs is earlier, and therefore to be considered the source of the relative timing constraint.
- **Constraint time interval:** The relative timing constraint interval is considered to the distance between the finish time of the source job to the start time of the target job. The interval is generated as a truncated Normally distributed percentage of the task's period with truncation percentages RC_{min} and RC_{max}.

Communication constraints: N_{cc} communication constraints are generated at random in each experiment according to the same guidelines as those used for the relative timing constraints, except that the source and target jobs are not allowed to be on the same system node. The communication constraint time interval is also generated as a truncated Normally distributed percentage of the task's period with truncation percentages CC_{min} and CC_{max}.

Parameter description	**Symbol**	**Nominal value**
Truncated Normal distribution standard deviation to mean ratio	σ/μ	0.70
Number of nodes	M	20
Number of tasks	N	50
Number of period levels	n_p	3
Jitter constraints minimum period percentage	J_{min}	0.40
Jitter constraints maximum period percentage	J_{max}	0.60
Ready-time minimum period percentage	r_{min}	0.01
Ready-time maximum period percentage	r_{max}	0.10
Deadline minimum period percentage	d_{min}	0.90
Deadline maximum period percentage	d_{max}	0.99
Minimum execution time minimum period percentage	l_{min}	0.01
Minimum execution time maximum period percentage	l_{max}	0.05
Maximum execution time minimum period percentage	l_{min}	0.05
Maximum execution time maximum period percentage	l_{max}	0.15
Number of relative timing constraints	N_{rc}	40
Relative timing constraint time interval minimum period percentage	RC_{min}	0.20
Relative timing constraint time interval maximum period percentage	RC_{max}	0.50
Number of communication constraints	N_{cc}	20
Communication constraint time interval minimum period percentage	CC_{min}	0.20
Communication constraint time interval maximum period percentage	CC_{max}	0.50

Figure 12: Real-time system random generation parameters

Therefore, the parameters that control the real-time system generation process can be summarized in figure 12 that includes the parameter name, symbol, as well as a nominal value for the parameter in order to achieve an average *Schedulability Success Ratio* (Definition 9.1).

The *Criteria for performance evaluation* used in the conducted experiments to measure the of the distributed scheduling algorithm under various parameter setup is the *Schedulability Success Ratio (SSR)* (Definition 9.1).

Definition 9.1 (Schedulability Success Ratio (SSR)) *The percentage of real-time systems (task sets) verified to be schedulable by a scheduling algorithm over the randomly generated set of real-time systems.*

In the following subsections, we present the simulation experiments that are based on the distributed real-time system random generation method described here.

9.3 Experiment 1

In this experiment, we investigate the effect of varying the secondary ordering heuristic used in the global order generation process on the distributed dynamic hard real-time schedulability verification method described in section 5.

In order to achieve this goal, 500 distributed real-time systems are generated randomly. Each one of the generated systems is run through the dynamic hard real-time scheduler to check its schedulability few times. Each time a different ordering heuristic method is used. The heuristic methods tested are:

- Earliest Deadline First (EDF).

Heuristic	EDF	ERF	LLF	RM	RANDOM
Total	273	273	273	273	273
SSR	% 54.6	% 54.6	% 54.6	% 54.6	% 54.6

Figure 13: SSR for different heuristic methods

- Earliest Ready-time First (ERF).
- Lease Laxity First (LLF).
- Rate Monotonic (RM).
- Random selection ($RANDOM$).

The SSR is calculated for each of the heuristic methods. Throughout the test, we keep track of the differences in the outcome of the schedulability test for the same systems as measured using the different heuristics.

9.3.1 Results

Repeating the experiment several times, we noted that there were no systems rendered schedulable by one heuristic method and not schedulable by another. We also measured the number of schedulable systems in each run, and found out that the numbers are always identical for all heuristic methods. The results of this experiment are shown in figure 13.

9.3.2 Conclusions

From this experiment, we conclude that the heuristic used as a secondary sorting criteria to get the distributed system global order *does not affect* the outcome of the schedulability verification process. This result enforces the correctness of theorem A.1 presented in appendix A, and consequently the correctness of the schedulability verification algorithm established in section 6. Therefore, the distributed parametric schedulability condition $\Gamma^{1,k}$ (definition 4.3) is the necessary and sufficient condition for the parametric schedulability of a distributed periodic real-time system, and the real-time scheduling algorithm described in section 5 is sufficient to verify the dynamic schedulability of distributed set of periodic real-time tasks with intertask relative timing and communication constraints.

9.4 Experiment 2

In this experiment we investigate the effect of varying some of the parameters governing the real-time systems generation process on the schedulability verification algorithm. The parameter variation effect on the scheduling algorithm is measured by its effect on the schedulability success ratio (SSR) as measured by the distributed scheduling algorithm. A nominal value is fixed for each of the parameters as shown in figure 12. Then we start varying each of the parameters separately in a range around its nominal point using small steps. The parameters varied in this experiment are shown in figure 14 along with their nominal values, variation range, and step. In each step of a parameter variation, we generate 500 real-time systems for each of the heuristics used (EDF, ERF, LLF, RM). The schedulability of each of the generated systems is checked using the dynamic scheduler, and the average SSR is calculated for each one of the heuristics separately.

Parameter	Nominal value	Range	Step
N	50	10 - 100	1
M	20	2 - 50	1
n_p	3	1 - 6	1
N_{rc}	40	2 - 100	1
N_{cc}	20	2 - 50	1
	0.70	0.10 - 1.45	0.05

Figure 14: Parameter variation ranges and steps

9.4.1 Results

The variation in the schedulability success ratio (SSR) as a result of varying each of the parameters (N, M, n_p, N_{rc}, N_{cc}, σ/μ) are shown in figures 15, 16, 17, 18, 19, 20 respectively.

9.4.2 Conclusions

From the results of this experiments we can draw the following conclusion in regards of the dynamic scheduling algorithm under consideration:

- The schedulability of a distributed real-time system is directly proportional to the number of nodes and inversely proportional with the number of tasks. Therefore we can conclude that the schedulability of a distributed system is inversely proportional with the density of tasks on the distributed system nodes.
- By increasing the number of system period levels allowed to the control tasks, the system schedulability dramatically decreases. This is due to the increased variability in the system tasks periods and instanciation frequency, which produces high probability for generating infeasible relative timing constraints among task instances with large difference between their frequencies.
- Increasing the number of timing constraints, whether they represent relative timing constraints or communication constraints, decreases the schedulability of the system. Which is due to the increased number of feasibility conditions that the system will have to satisfy to achieve schedulability. The tighter the system timing constraints, the larger their effect on the system feasibility.
- Increasing the standard deviation to mean ratio of the truncated Normal distribution used to generate the time interval values decreases the system schedulability. This is due to the increased variability in the system timing constraints time interval values, and consequently the probability of generating contradicting constraints is also increased.

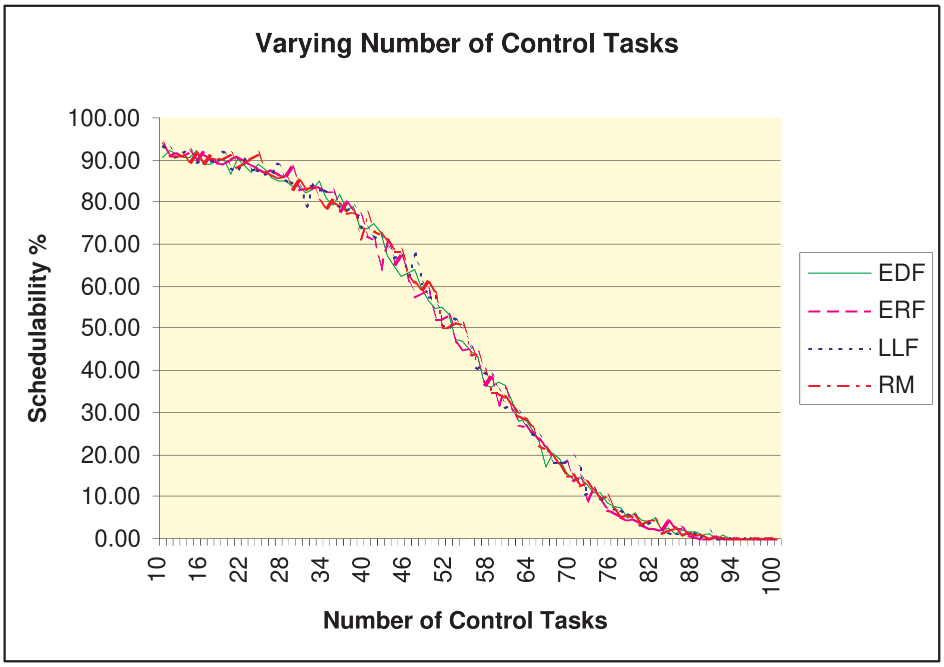

Figure 15: Varying the number of real-time control tasks

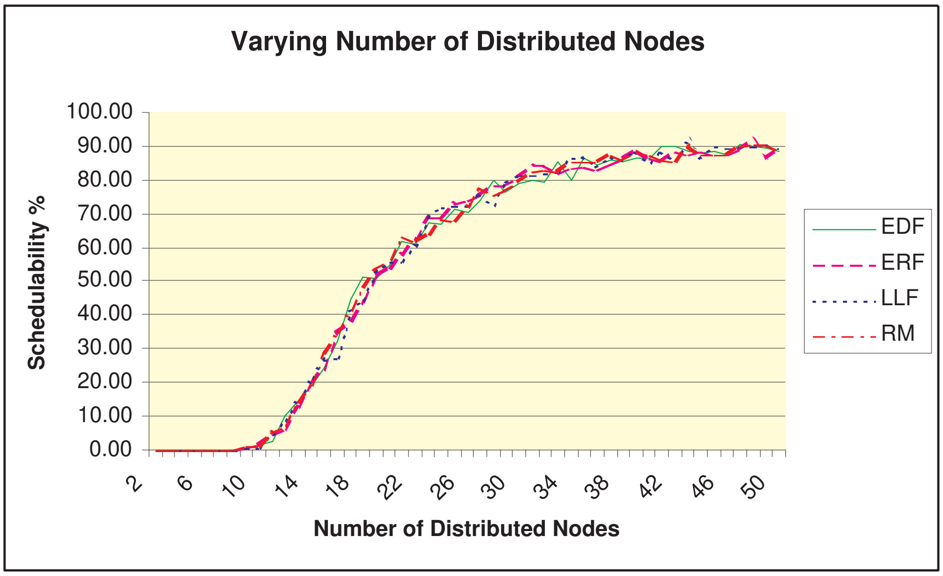

Figure 16: Varying the number of real-time system nodes

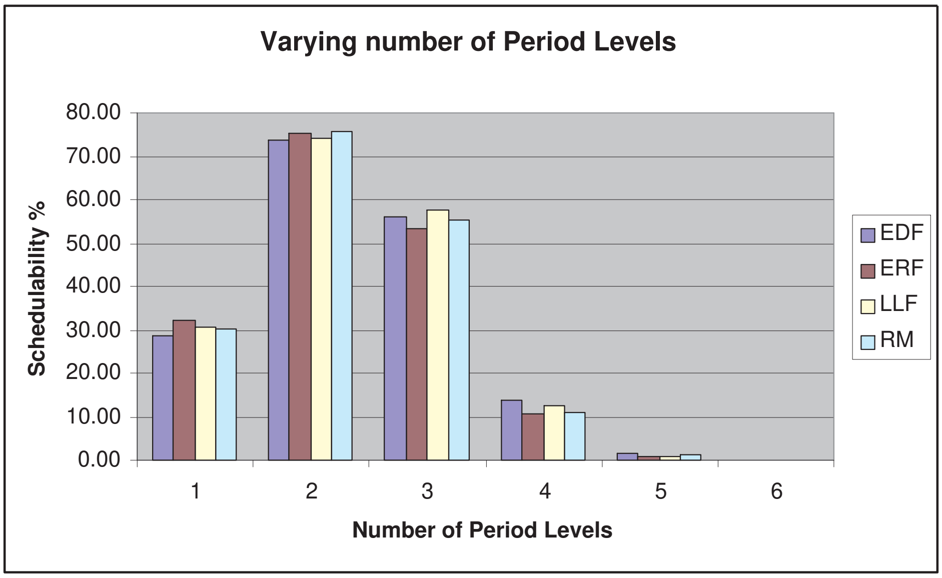

Figure 17: Varying the number of period levels

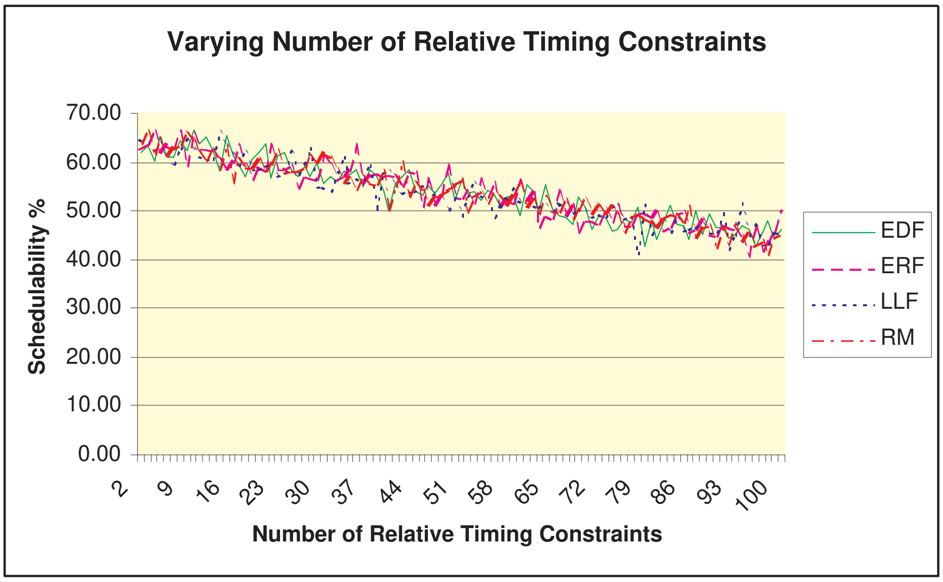

Figure 18: Varying the number of relative timing constraints

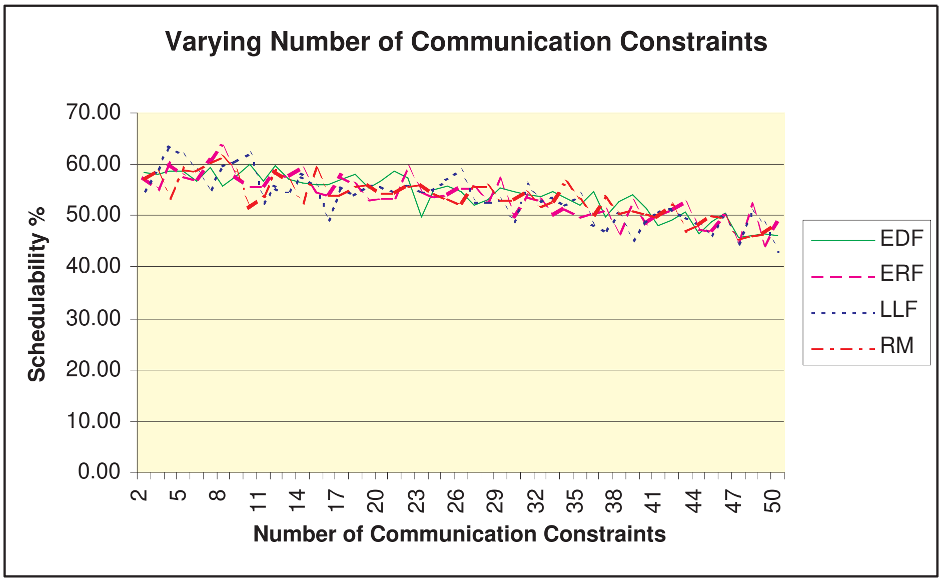

Figure 19: Varying the number of communication constraints

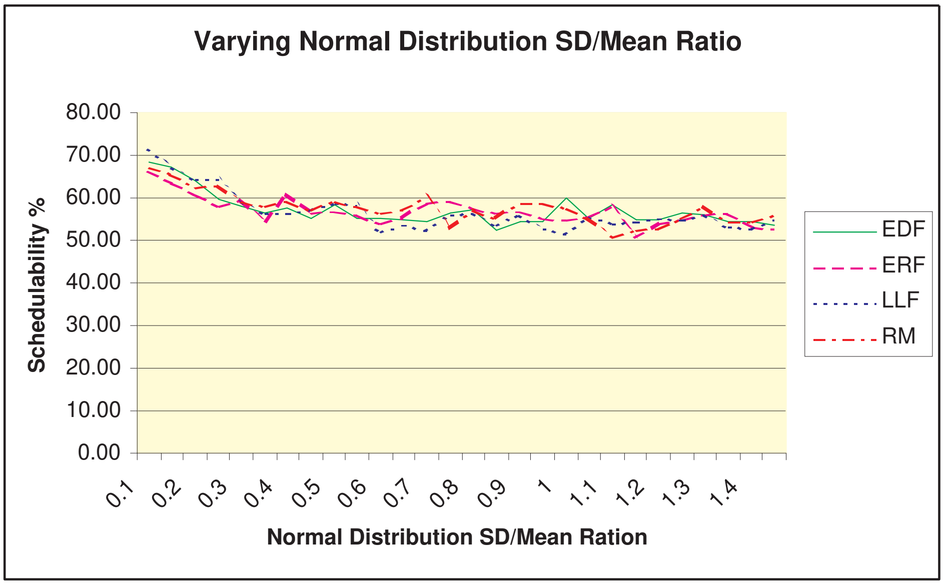

Figure 20: Varying Normal Distribution SD/Mean Ratio

References

[1] A. Silberschatz and Peter B. Galvin. *Operating Systems Concepts.* Addison Wesley, fifth edition, November 1998.

[2] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The Design of the TAO Real-Time Object Request Broker. Technical report, Department of Computer Science, Washington University, St. Louis, MO 63130, USA, January 1999.

[3] Carlos O'Ryan, David L. Levine, Douglas C. Shmedit, and J. Russel Noseworthy. Applying a Scalable CORBA Events Service to Large-scale Distributed Interactive Simulations. In *Proceedings of the 5th Workshop on Object-oriented Real-time Dependable Systems*, November 1999.

[4] Christopher D. Gill, Fred Kuhns, David L. Levine, and Douglas C. Shmedit. Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems. In *Proceedings of the 1st International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems, IEEE, Phoenix, Arizona*, November 1999.

[5] Irfan Pyarali, Timothy H. Harrison, and Douglas C. Shmedit. Design and Performance of an Object-Oriented Framework for High-Speed Electronic Medical Imaging. *Computing Systems Journal*, December 1996.

[6] Manas Saksena. *Parametric Scheduling for Hard Real-Time Systems.* PhD thesis, University of Maryland at College Park, 1994.

[7] Seonho Choi. *Dynamic Time-Based Scheduling for Hard Real-Time Systems.* PhD thesis, University of Maryland at College Park, 1997.

[8] G. Dantzig and B. Eaves. Fourier-Motzkin Elimination and its dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

[9] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems, Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers, 1997.

[10] M. L. Dertouzos and A. K. Mok. Multiprocessor On-Line Scheduling of Hard Real-Time Tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, 1989.

[11] G. Fohler and C. Koza. *Heuristic Scheduling for Distributed Real-Time Systems.* PhD thesis, Techniche Universitat Wien, Vienna, Austria, April 1989.

[12] J. Xu and D. L. Parnas. Scheduling Processes with Release Times, Deadlines, Precedence and Execlusion Relations. *IEEE Transactions on Software Engineering*, SE-16(3):360–369, 1990.

[13] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In *10th International Conference on Distributed Computing Systems*, pages 108–115, 1990.

[14] T. Shepard and J. A. M. Gagne. A Pre-Run-Time Scheduling Algorithm for Hard Real-Time Systems. *IEEE Transactions on Software Engineering*, 17(7):669–677, 1991.

[15] J. P. C. Verhoosel, E. J. Luit, D. K. Hammer, and E. Jansen. A Static Scheduling Algorithm for Distributed Hard Real-Time Systems. *Real-Time Systems*, 6(2):227–246, 1991.

[16] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority. In *IEEE Real-Time Systems Symposium*, pages 116–128, December 1991.

[17] K. Tindell, A. Burns, and A. Willings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6(2), March 1994.

[18] Aloysius K. Mok and Duu-Chung Tsou. The MSP.RTL Real-Time Scheduler Synthesis Tool. In *IEEE Real-Time Systems Symposium*, pages 118–128, December 1996.

[19] S. Cheng and A. K. Agrawala. Scehduling of Periodic Tasks with Relative Timing Constraints. Technical report, CS-TR-3392, UMIACS-TR-94-135, Department of Computer Science, University of Maryland, December 1994.

[20] R. Gerber, W. Pugh, and S. Saksena. Parametric Dispatching of Hard Real-Time Tasks. *IEEE Transactions on Computers*, 44(3), March 1995.

[21] S. Choi, S. Elsharkawy, Bao Trinh, and A. Agrawala. Dynamic Time-Based Scheduling in Hard Real-Time Systems. 1998.

[22] Dilip D. Kandlur, Kang Shin, and Domenico Ferrari. Real-Time Communication in Multi-hop Networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 1044–1056, October 1994.

[23] A. Mehra, A. Indiresan, and K. Shin. Design and Evaluation of a QoS-Sensitive Communication Subsystem Architecture. *IEEE Transactions on Parallel and Distributed Systems*, CSE-TR(280-96), January 1996.

[24] Tarek Abdelzaher, Anees Shaikh, Farnam Jahanian, and Kang Shin. RTCAST: Lightweight Multicast for Real-Time Process Groups. Technical report, Real-time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, 1996.

[25] Z. Deng and J. W. Liu. Scheduling real-time applications in an open environment. In *Proceeding of the Real-Time System Symposium, San Francisco, California*, pages 308–319, December 1997. IEEE.

[26] Kay Connelly and Andrew A. Chien. FM-QoS: Real-time communication using self-synchronizing schedules. In *High Performance Networking and Computing, San Jose, California*, pages 15–21, November 1997. ACM SIGARCH and IEEE-CS-TCCA.

[27] P. Ramanathan, D. D. Kandlur, and K. G. Shin. Hardware assisted software clock synchronization for homogeneous distributed systems. *IEEE Transactions on Computers*, C-39:514–524, April 1990.

[28] Veeravalli Bharadwaj, Dedasish Ghose, Venkataraman Mani, and Thomas G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE Computer Society Press, 1996.

A Variable Elimination Order

Theorem A.1 *Given a set of ordered variables $X_N = [x_1, x_2, \dots, x_{n-1}, x_n, \dots, x_N]$ with standard relative constraints among them. If the two consecutive variables x_{n-1} and x_n do not have any constraints between them, then their relative order does not affect the final outcome of the Fourier-Motzkin variable elimination process.*

Proof:
After eliminating the variables x_N through x_{n+1}, the remaining variables are:

$$X_n = [x_1, x_2, \dots, x_{n-1}, x_n]$$

The corresponding set of standard relative timing constraints

$$C_n \equiv AX_n \leq b$$

After eliminating the variable x_n, the remaining variable vector is:

$$X'_n = X_{n-1} = [x_1, x_2, \dots, x_{n-1}]$$

Eliminating x_{n-1} as well, the remaining variable vector is:

$$X'_{n-1} = X_{n-2} = [x_1, x_2, \dots, x_{n-2}]$$

Since there is no timing constraints between x_{n-1} and x_n, then there is no constraint in the equation set C_n that has both variables. Therefore, we can partition the constraint set into the following disjoint sets:

$$C_n \equiv C_{P_n} \wedge C_{N_n} \wedge C_{P_{n-1}} \wedge C_{N_{n-1}} \wedge C_Z$$

where:

- C_{P_n}: is the set of constraints that contain the variable x_n with a positive coefficient.

$$C_{P_n} \equiv \{x_n \geq D_i(X'_n), \quad 1 \leq i \leq p\}$$

- C_{N_n}: is the set of constraints that contain the variable x_n with a negative coefficient.

$$C_{N_n} \equiv \{x_n \leq E_j(X'_n), \quad 1 \leq j \leq q\}$$

- $C_{P_{n-1}}$: is the set of constraints that contain the variable x_{n-1} with a positive coefficient.

$$C_{P_{n-1}} \equiv \{x_{n-1} \geq F_k(X'_{n-1}), \quad 1 \leq k \leq r\}$$

- $C_{N_{n-1}}$: is the set of constraints that contain the variable x_{n-1} with a negative coefficient.

$$C_{N_{n-1}} \equiv \{x_{n-1} \leq G_l(X'_{n-1}), \quad 1 \leq l \leq s\}$$

- C_Z: which is the set of constraints that contain neither x_{n-1} nor x_n.

$$C_Z \equiv \{0 \leq H_m(X'_{n-1}), \quad 1 \leq m \leq t\}$$

Eliminating x_n using the Fourier-Motzkin elimination process leads to a new equivalent system of constraints:

$$C'_n \equiv \exists x_n :: C_n \equiv \left\{ \begin{array}{rcll} D_i(X'_n) & \leq & E_j(X'_n), & 1 \leq i \leq p, \quad 1 \leq j \leq q \\ x_{n-1} & \geq & F_k(X'_{n-1}), & 1 \leq k \leq r \\ x_{n-1} & \leq & G_l(X'_{n-1}), & 1 \leq l \leq s \\ 0 & \leq & H_m(X'_{n-1}), & 1 \leq m \leq t \end{array} \right.$$

Eliminating x_{n-1}, the new constraint system is:

$$C'_{n-1} \equiv \exists x_n :: \exists x_{n-1} :: C_n \equiv \left\{ \begin{array}{rcll} D_i(X'_n) & \leq & E_j(X'_n), & 1 \leq i \leq p, \quad 1 \leq j \leq q \\ F_k(X'_{n-1}) & \leq & G_l(X'_{n-1}), & 1 \leq k \leq r, \quad 1 \leq l \leq s \\ 0 & \leq & H_m(X'_{n-1}), & 1 \leq m \leq t \end{array} \right. \tag{20}$$

From equation 20, since the elimination of variables x_n and x_{n-1} affect two disjoint sets of constraints, it is obvious that applying the elimination process in the reverse order would result in the same set of constraints represented in equation 20. As a result, we conclude that the order of elimination of two consecutive variables that do not have any constraints between them does not affect the outcome of the Fourier-Motzkin variable elimination process.