A Generalized Framework for Hierarchical Real-Time Scheduling

Shanmuga Priya Marimuthu

(B.E. Computer Science)

Supervised by

Samarjit Chakraborty

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2006

ABSTRACT

Most real-time systems allow concurrent execution of different applications and it is necessary to guarantee that the timing requirement of each application is met. One way to ensure this is to compose all the applications with a unique scheduling paradigm at the system level and modify the applications to suit the chosen paradigm. But most often, it is desirable to keep the implemented application and determine the feasibility of scheduling the application in conjunction with other applications. The problem gets more involved as we try to compose applications that come with their own scheduling strategy. An alternate approach to compose existing applications with different timing characteristics is to use a two-level scheduling paradigm, comprising of a global scheduler at the system level and a local scheduler for each application. The global scheduler selects the application that will be executed next and assigns to it a fraction of the total processor time according to certain criteria and the scheduler is feasible only if it preserves the temporal guarantees of the local scheduling models. Each local scheduler schedules tasks within the application. Such hierarchical composition of schedulers allows for maximum flexibility in the design of systems with a mix of tasks, each having different timing constraints. A considerable amount of work has been recently addressed to the analysis of these kind of hierarchical systems. Various resource reservation schemes have been proposed and the notion of real-time virtual resources gives a very flexible parameterization of resource partitions. We propose a generalized

framework for hierarchical scheduling that permits resource partitioning to be extended to multiple levels. In constructing the hierarchical scheduling framework we intend to combine the advantages offered by the notion of virtual resources with the flexibility of real-time calculus in accommodating non-standard event models and permitting re-use of unused computation capacity. The framework handles a wider range of task models and permits data dependencies among tasks and task groups.

Keywords: component-based design, virtual resources, real-time calculus, hierarchical scheduling

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my supervisor Dr. Samarjit Chakraborty. His wide knowledge, insights and suggestions have been of great value to me. His patience, encouragement and valuable feedback contributed greatly to this thesis.

I wish to express my warm and sincere thanks to Professor P. S. Thiagarajan, who kindled my interest in the embedded systems domain while I attended his module on Hardward Software Co-design.

I thank my thesis committee members: Dr. Roland Yap and Dr. Wong Weng-Fai who read the Graduate Research Paper and provided valuable comments at the time of proposal presentation.

I feel a deep sense of gratitude to my parents whose constant encouragement helped me fight the sense of solitude that used to creep in and stay focused on my work. I owe my loving thanks to my fiancé Dinesh Kumar for his understanding and patience throughout my candidature. I am grateful to my brother and sister-in-law for their loving support and constant goodwill.

I had the pleasure of interacting with other students of the embedded systems lab, particularly Ramkumar, Dinesh and Unmesh and I am thankful to them for having shared their experiences and thoughts through the last one year. My special thanks goes to my good friend Pavan who has been very supportive and helpful. My special gratitude is due to my local guardians Mr. David Lee and Mr. Michael Lee, for making my stay very comfortable.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1. INTRODUCTION

Component technology has been widely accepted as a methodology for designing large complex systems. Component-based design provides a means for decomposing a system into components, allowing the reduction of a single complex design problem into multiple simpler design problems, and finally integrating the components into the system. The central idea of component based design is to assemble components into the system without violating the principle of compositionality such that properties that have been established at the component level will also hold at the system level. To preserve compositionality, the properties at the system level need to abstract the collective properties at the component level.

Component-based design and development techniques are now being applied in real-time embedded systems. Owing to the advances in the field of computer architecture, it is now possible to concurrently execute different applications on the same processor. The motivation is to make even hand-held devices execute general purpose applications by re-using legacy applications. Thus a real-time system could have many functional components (applications) that share a single resource. Each application could come with its own scheduling strategy. The individual applications are developed separately and then integrated into the system. During integration there are two concerns facing the system developer and designer. The timing constraints of all the tasks within the application should be respected even after system integration. The individual applications should execute

in isolation with no interference from other applications. If an application has more resource requirements, it should not compromise the resource allocations to other applications.

When the individual applications have been independently developed there are two ways to compose the different applications at the system-level. One way to do composition is to adopt a flat approach, using a unique scheduling paradigm for the whole system and design all applications according to the chosen paradigm: then, it is possible to check the schedulability of the whole system by using already existing schedulability analysis tools. This would require knowing the timing requirements of all tasks in every application and the generation of a feasible schedule is a cumbersome task. Sometimes it is necessary to use an already implemented application, without redesigning it for the new system. In such cases, it is necessary to guarantee that the application still meets its timing requirements when it is scheduled along with other applications in the same system. This approach might pose problems when applications come with their own scheduling strategy. There is no single scheduler that is best for all kinds of application domains. For example, applications that are event-triggered are best served by on-line scheduling algorithms like fixed priority or earliest deadline first; time triggered applications are best handled by off-line schedulers like TTA [12]. Thus for composing applications with different temporal requirements handled by different scheduling policies, the flat composition approach is not feasible. Moreover this approach is inflexible as it is necessary to know the temporal requirements of individual tasks of all applications and the scheduler-design process would have to be initiated all over again anytime a new application is added to the system.

An alternate way of composing existing applications with different timing

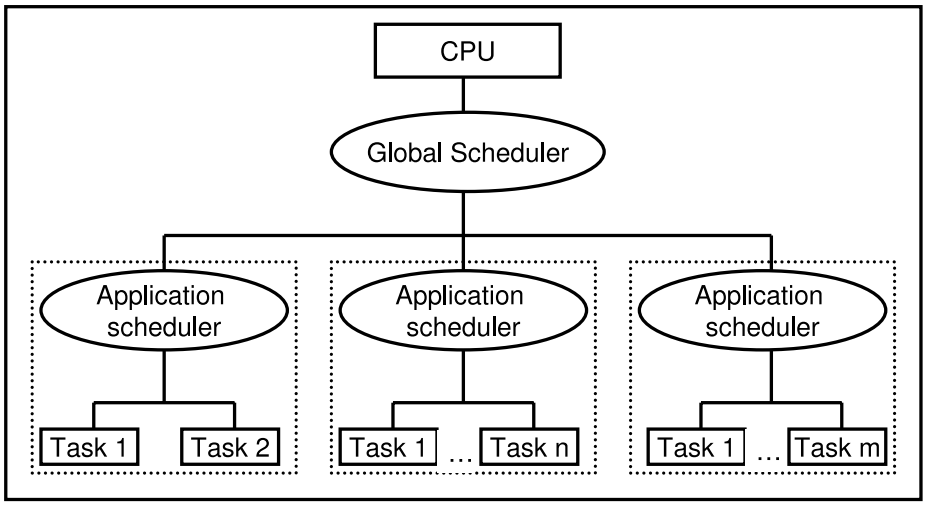

Fig. 1.1: Hierarchical scheduling model

characteristics is to preserve the application-level scheduling policy, and use a two-level scheduling paradigm: at the global level, a scheduler selects which application will be executed next and assigns a fraction of the total processor time distributed over the time line according to certain criteria. Each application possesses a local scheduler that selects which task will be scheduled next. The global scheduler must also protect applications from one another, such that an application that requires more resources does not compromise the requirement of other applications. In general, the hierarchy can be more than two level deep.

A hierarchical scheduling framework can be represented as a tree, or hierarchy of nodes, where each node represents a scheduling model and resource allocation flows from a parent node to its child nodes, as illustrated in Figure 1.1. Each application along with the application-level scheduler is considered to be a single component. The system-level scheduler allocates resources to the individual components with no knowledge of the task-level details of the components.

It is obvious that such a hierarchical framework achieves the following advantages:

- **Flexibility**: Compositionality allows for maximum flexibility in design of systems with different real-time applications. In many real-time systems, there is usually a mix of tasks with mixed constraints, i.e. some activity might be critical and treated as hard real-time task (i.e. no deadline must be missed); some other activity is less critical and nothing catastrophic happens if some constraint is not respected. However, the quality of service drops with missing more deadlines (soft real-time tasks). Different scheduling paradigms are used for hard and soft real-time activities. To compose such applications, it would be necessary to implement different classes of real-time tasks as different components, each component with its own scheduling algorithm. Thus these components could be developed independently.

- **Re usability**: Compositionality allows reuse of existing applications without changing their scheduling policy. Suppose we have a component that consists of many concurrent real-time tasks, that has already been developed assuming a fixed priority scheduler, and that component needs to be re-used in a new system with earliest deadline first scheduling algorithm, then the hierarchical composition of applications would allow the existing application to be re-used and just integrate it into the system without changing its scheduling algorithm.

- **Isolation and Ease of Analysis** Each application would just run as if it is executing on a dedicated resource. The resource sharing will happen at the higher level in the hierarchy and the application would have no clue about it. Similarly the resource level scheduler would

be totally unaware of the timing requirements of the application level scheduler and is only responsible for allocated time slices according to some pre-defined criteria to the applications under it. The schedulability at the resource level and at the application level is thus analyzed independently. A separate scheduling problem is solved at each level of the hierarchical framework and the schedulability of every component can be analyzed independently.

In real-time systems research, there has been growing attention to hierarchical scheduling owing to the flexibility and other advantages offered by such an approach. Figure 1.2 illustrates a part of the scheduling model with a three level hierarchy. Applications A_3 and A_4 in the figure are subcomponents of the component application A_1. A_3 consists of m tasks scheduled by the round-robin (RR) scheduling algorithm and A_4 consists of n tasks scheduled by the rate-monotonic (RM) scheduling policy. A_3 and A_4 are allocated resources R_3 and R_4 respectively. At the level of application A_1 that schedules A_3 and A_4 under it using the earliest-deadline-first (EDF), the scheduler views the resources requirements of the child component as its workload and schedules it using resource R_1.

The hierarchical and compositional framework thus represented, naturally gives rise to component characterization using three parameters: the resource, workload and the scheduling policy. There are different issues that need to be addressed in analyzing or constructing such a framework.

- **Component demand abstraction:** The system-level scheduler handles the individual applications as single entities without knowledge about the lower level details of the tasks within the applications. To determine how much fraction of the physical resource each application should get, the system-level scheduler should know the total timing re-

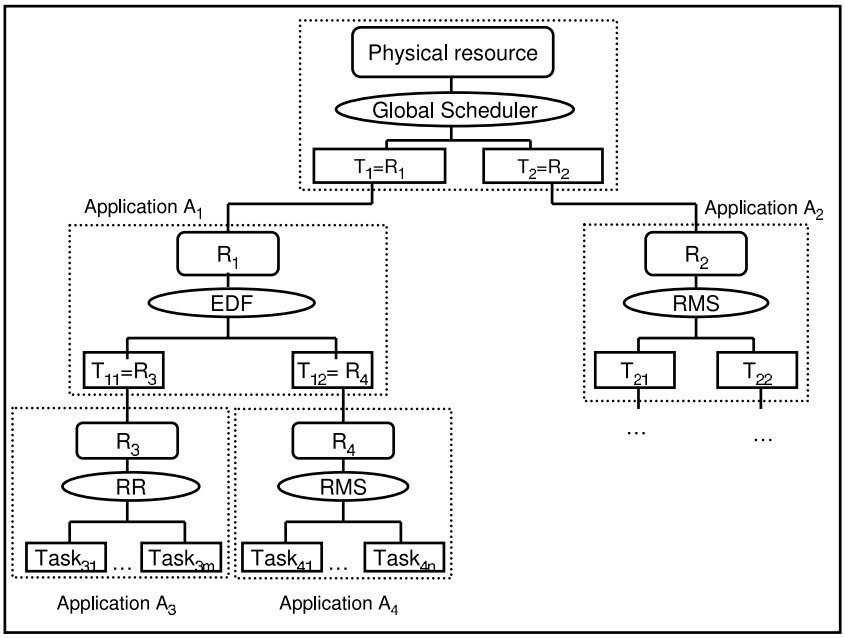

Fig. 1.2: Closer view of a hierarchical scheduling model

quirement of the individual applications. Thus within the application there should be a means of extracting the collective timing requirements of all the individual tasks. This collective timing requirement or collective demand imposed by the application is passed on the system-level scheduler and the resource supply is determined based on this demand. Extracting the temporal requirements of individual tasks and determining the collective demand of a component is called *component demand abstraction*. It is desirable to have tight abstractions so that the resource requirement is not over-estimated. Thus given the workloads and the scheduling policies, we could derive the best or minimal resource allocation for that component that will successfully schedule the given workload under the given scheduling policy with minimal wastage of resources or maximum resource utilization.

Given the set of workloads along with the scheduling policies, we could

solve the component demand abstraction at every level of the hierarchy; finally the parent scheduling model could be derived. Thus the hierarchy would be constructed from the leaf-level tasks of individual applications.

- **Schedulability analysis:** Given the set of workloads, scheduling algorithms and the resource allocations, we could determine the exact schedulability conditions for real-time guarantees. For a given workload, the component demand abstraction is solved to give the collective resource demand under a given scheduling policy for any interval of time and this demand is represented by a *demand bound function* ($dbf(t)$). From the resource allocation specified, the bound on the minimum supply of the resource within any time interval could be determined as the *supply bound function* ($sbf(t)$). A necessary and sufficient condition for scheduling the given workload under the given resource is to check for $dbf(t) \leq sbf(t),\ \forall t$. Intuitively if the minimum resource supply satisfies the maximum resource demand then the component is schedulable. Thus it is necessary to come up with tight bounding functions for the workload and resource models to determine schedulability.

- **Other temporal properties:** Given a resource and the tasks along with their triggering times given as the event model, it should be possible to compute the timing properties of the processed event stream. Thus task dependencies could be handled by using this processed event stream as the event model for the dependent task. Similarly, it should be possible to compute the computation capacity left over after servicing some workload. This reclaimed service could be used to schedule some other workload.

1.1 Resource reservation schemes

The analysis or construction of the hierarchical framework assumes that the resource allocation is parameterized using some kind of resource model. A lot of emphasis has been on developing flexible resource reservation schemes that naturally fit into the hierarchical framework. The notion of real-time virtual resources [11] proposed by Mok and Feng gives a very flexible representation of resource reservation with guaranteed output jitter. The hierarchical framework proposed by Mok and Feng was based on the bounded-delay resource partition model introduced in [20]. Shin and Lee extended this framework to give necessary and sufficient schedulability conditions for partition-level and application level scheduling with utilization bounds. In [24] the component demand abstraction problem was defined to be the specification of the collective timing requirements of the task groups to the higher level schedulers and so on. A solution to this component demand abstraction problem was proposed, that abstracts the requirements of a set of independent periodic tasks as a periodic resource so that the higher level scheduler can handle it as a single periodic task. The results presented in [24] were used in another framework [25] based on the bounded-delay resource model. The authors have also derived the utilization bounds of the bounded-delay resource under EDF and RM scheduling.

1.1.1 Drawbacks of previous approaches

The main drawback of the previous approaches is that the workload models and scheduling algorithms are restrictive. The frameworks accommodate only workloads composed of independent periodic tasks. To accommodate a more generalized workload model, it is again approximated to a periodic task model for the purpose of analysis which leads to loss of accuracy and

pessimism in timing estimates. The schedulability conditions are derived for EDF and RM algorithms although many other scheduling policies could be used. Moreover, these frameworks do not handle data dependencies between tasks within a task group or between task groups. In the hierarchical schedulability analysis, the child scheduling models abstract the resource requirement depending on their workload demands. This resource requirement would then form the workload demand of the parent resource model and the parent model has to find the resource requirement to satisfy this demand. In this manner, finally the solution to the resource model at the physical-level, which would be sufficient to schedule all the applications, is derived. Thus the problem is to find the resource model at the physical level, given the workloads and scheduling algorithms, using a bottom-up approach.

1.1.2 Our contributions

We would like to incorporate both top-down and bottom-up approaches to the construction and or analysis of the hierarchical scheduling framework. In some cases it would be desirable to determine if a given physical resource could be partitioned according to some scheduling policy to schedule a given set of workloads. Thus a top-down approach would be required for such analysis. It would also be desirable to reclaim unused computation capacity, if any within the partition to schedule other task groups. This way resource partitions and workloads could exist on the same level of the hierarchy. We propose a generalized hierarchical scheduling framework based on the real-time calculus framework presented in [27][7]. The resource model based on the real-time calculus framework gives tighter bounds on the resource supply and allows computation of unused computation capacity so that this

computation capacity could be supplied to some workload on the same level or lower level of the hierarchy. Thus the hierarchical framework allows flexible resource partitioning and resource sharing across levels in the hierarchy. It can also accommodate non-standard workload models and handle data dependencies among tasks or task groups. A more precise summary of our approach to hierarchical scheduling is presented in section 1.3.

1.2 Background

The construction of a hierarchical scheduling framework is to a large extent dependent upon the resource reservations schemes that allow the various applications to share the physical resource. A lot of effort has been into parameterizing the resource allocation such that the availability of the resource and additional parameters like the period of the resource are explicitly amenable to analysis so that resource supply bounds can be derived. Thus the properties of the resource model should be deducible from the parameterization. At every level of the hierarchy, it is desirable that there is no over-provisioning of resources so that we have maximum utilization. A lot of research has been directed toward coming up with good resource reservation schemes. In general resources are classified as:

- Dedicated resources: A resource is said to be dedicated if it is available to a scheduling component at its full capacity at all times.
- Shared resources: A resource is said to be shared if it is not a dedicated resource. Depending on the nature of resource sharing we could classify shared resource as follows:
  - Fractional resource: A fractional resource is one that available to a scheduling component at all times but at a fractional ca-

pacity. The Generalized Processor Sharing (GPS) [21] approach is an example of fractional resource model. This is a theoretical abstraction and there are some approximation methods to make this model amenable for implementation.

– Partitioned resource: A resource is said to be partitioned if it is available to a scheduling component at some times at its full capacity and unavailable at all other times. A partitioned resource is a shared resource on the basis of time-sharing. The bounded-delay [20] and periodic resource models [24] are partitioned resource models. In the hierarchical scheduling framework, we deal with partitioned resource models.

The resource allocation at every parent scheduling model acts like a server providing some amount of resource to the child scheduling models. Thus roughly speaking, the term *server abstraction* refers to some criteria for assigning resources to a set of tasks. Various server abstractions schemes have been proposed in literature. A detailed description of these server abstraction schemes and their applicability to hierarchical scheduling frameworks is presented as a part of the literature survey in Chapter 2.

1.2.1 Partitioned resource models

In our work we focus on partitioned resource models. The idea behind constructing the hierarchical scheduling framework is that each task group can be analyzed by itself for schedulability assuming the resource from the parent scheduling model is time-shared amongst the task group in question and some other task groups that share the same parent model. If the task group can be considered to be allocated a small fraction of the resource at the parent level it implies that the resource at the parent-level is time-shared by

infinite time slicing. Infinite time slicing means that the parent-level resource is split in such fine time granularity, that it is available at fractional rates to the task groups. This means the resource would be available to all task groups at a fractional rate at all times, which in other words is a GPS [21] approach. The net effect is that each task group has exclusive access to the resource that is made available at a fraction of the actual rate. But such infinite time-slicing is impractical due to resource-specific constraints and context switch overheads incurred using such an approach.

To characterize sharing of resources (without infinite time-slicing) at fractional rates, Mok and Chen [20] came up with a virtual resource model, wherein each task group can be viewed to be accessing a virtual resource that again operates at a fraction of the rate of the physical resource shared by the group but the rate varies with time during execution. The rate variation of each virtual resource is given by means of a delay bound D that specifies the maximum extra time the task group may have to wait in order to receive its fraction of the physical resource over any time interval starting at *any* point in time. In this manner, if we know that an event e will occur within x time units from another event e' assuming the virtual resource operates at a uniform rate and event occurrence depends only on resource consumption, then e and e' will be apart by at most $x + D$ time units in real-time. If infinite time-slicing is possible, the delay bound is zero and this becomes a fractional resource. In general the delay bound of a virtual resource rate variation is task-group specific. The characterization of virtual resource rate variation by means of the delay bound will allow the task models to be characterized with more general types of timing constraints like jitter. Virtual resources whose rate of operation variation is bounded are called real-time virtual resources.

The rate variation and therefore the delay bound of a real-time virtual resource is in general a function of the scheduling policy used to allocate the shared physical resource among the task groups. A simple approach to construct real-time virtual resources that is especially amenable to delay bound determination is through temporal resource partitioning. Two resource partition models were proposed in [20] and the conditions for schedulability of task groups within partitions were presented. For simplicity of schedulability analysis, a task group was assumed to consist of a set of periodic tasks, although the task period could be interpreted as the minimum inter-separation time in the sporadic task model. Although a resource could refer to the computation units of a single-processor or a communication bus, the resource considered in [20] is assumed to be a single processor that is shared by a collection of tasks. Informally, a (temporal) partition of a resource is just a collection of time intervals during which the resource is made available to the task group that is scheduled on the partition. To formalize the notion of resource partitions, first a resource partition model (static resource partition) with explicit specification of time intervals is presented below and then a more generalized representation of the virtual resource (bounded-delay resource partition) is described in section 3.1.1 of Chapter 3.

Definition 1: A Static Resource Partition Π is a tuple (Γ, P), where Γ is an array of N time pairs $\{(S_1, E_1), (S_2, E_2), \cdots, (S_N, E_N)\}$ that satisfies $(0 \leq S_1 < E_1 < S_2 < E_2 < \cdots < S_N < E_N \leq P)$ for some $N \geq 1$, and P is the partition period. The physical resource is available to a task group executing on this partition only during time intervals $(S_i + j \times P, E_i + j \times P)$, $j \geq 0$.

The time intervals where the processor is unavailable to the partition is called the *blocking time* of the partition. If we consider a dedicated resource,

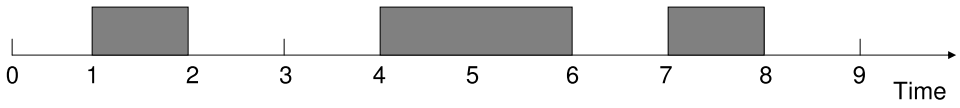

Fig. 1.3: Timing diagram of Static Partition Π_1

there is no blocking time and this would be a special case corresponding to the partition $\Pi = (\{(0, P)\}, P)$.

Definition 2: The availability factor of a static resource partition Π is $a(\Pi) = (\sum_{i=1}^{n}(E_i - S_i))/P$.

The availability factor is the cumulative sum of the time units when the resource is available within the time period of the resource taken over the resource period.

In [20] and [11], the availability factor is denoted by α but we use a, to differentiate it from the arrival curve α of real-time calculus.

Example 1: $\Pi_1 = (\{(1, 2), (4, 6)\}, 6)$ *is a resource partition whose period is 6 and the resource is available from time 1 to time 2 and from time 4 to time 6 every period as shown in Figure 1.3. The availability factor of the partition* Π *is* $a(\Pi_1) = ((2 - 1) + (6 - 4))/6 = 0.5$.

Definition 3: The Supply Function denoted by $S_\Pi(t)$ of a partition Π is the total amount of time that it is available in Π from time 0 to time t.

The definition of the supply function applies to any resource model and it is a monotonically non-decreasing function for t ($t \geq 0$). For a static resource partition the pattern of the supply function is repetitive as the partition is periodic.

Example 2: $\Pi_2 = (\{(0, 1), (2, 4)\}, 6)$ *is another resource partition as shown in Figure 1.4 whose available times are the blocking times of partition* Π_1.

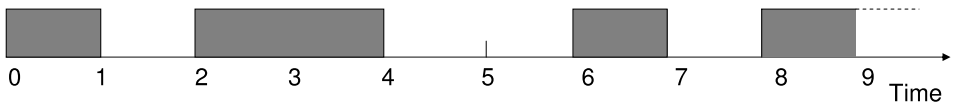

Fig. 1.4: Timing diagram of Static Partition Π_2

Thus a dedicated resource has been partitioned into two, each with availability 0.5*. There could be task groups scheduled on each of these partitions.*

The schedulability of a task group under the fixed priority scheduling policy is analyzed by considering the critical time instances. In a classic Liu and Layland model [18] the worst case scenario or the critical instant is when a task is requested simultaneously along with all higher priority tasks. For a partitioned resource model, the worst case scenario occurs when the task is requested at the start of a blocking time instance. With this observation, if the first instance of a task is schedulable at the start of every blocking time interval then it is schedulable on the partition.

If the task group has to be scheduled using a dynamic priority scheduling policy, the usual utilization bound 1.0 used for EDF scheduling no longer applies as the resource is not always available. For a static resource partition as given in the example, the supply function could be computed. Intuitively, for a task group denoted by G to be schedulable within the resource partition, the supply function of the resource should be sufficient to handle the demand imposed by the task group. The discussion pertaining to scheduling tasks within partitions is presented in section 1.2.2.

The static resource partition model is inflexible and is motivated by a scenario where the resource is already divided into a set of partitions and the goal is to schedule task groups in the given partitions. But due to its simplicity it is more amenable to timing correctness verification and it suitable for hard real-time systems. A periodic resource partition model

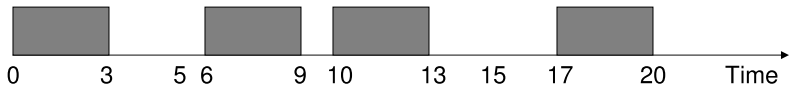

Fig. 1.5: Timing diagram of a periodic resource $\Gamma(5, 3)$

which is still periodic in nature like the static resource partition model, but does not explicitly specify time intervals, was proposed by Shin and Lee in [24].

Definition 4: The periodic resource partition Γ is characterized by (Π, Θ), and guarantees allocation of Θ time units every Π time units.

Example 3: $\Gamma(5, 3)$ *describes a partitioned resource that guarantees 3 time units every 5 time units as shown in Figure 1.5. The availability factor of this partition is* $a_\Gamma = \Theta/\Pi = 0.6$

A dedicated resource is a special case of the periodic resource model and is characterized by $\Gamma(k, k)$ for any integer k.

The advantage of using a resource model that is periodic in nature, is that most hard real-time applications are composed of periodic tasks and the demand imposed by the entire applications can be abstracted as a single periodic task.

A more generalized resource model called the bounded-delay resource model was introduced by Mok and Feng [20] and is discussed in detail in section 3.1.1 of Chapter 3.

Definition 5: The supply bound function denoted by $sbf_R(t)$ of a resource R, is the minimum guaranteed resource supply within any time interval t.

For a periodic resource the supply bound function is calculated as shown in Figure 1.6.

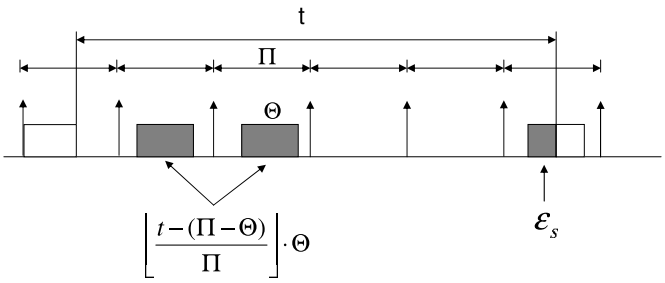

Fig. 1.6: Supply bound function of a periodic resource $\Gamma(\Pi, \Theta)$

1.2.2 Task model

The simplest task model is the classic Liu and Layland [18] periodic task model. For real-time systems, the periodic task model and its various extensions have been accepted as a workload model that accurately characterizes many traditional hard real-time applications, such as digital control and constant bit-rate voice/video transmission. Many scheduling algorithms based on this workload model have been shown to have good performance and well-understood behaviors.

Definition 6: A periodic task T is characterized as a tuple (p, e), where p is the period and e is the worst-case execution requirement of the task.

Definition 7: A task group τ is a collection of n tasks that are to be scheduled on a resource partition, $\tau = \{(e_1, p_1), (e_2, p_2), \cdots, (e_n, p_n)\}$.

Definition 8: The demand bound function $dbf_{\tau,A}(t)$ is the maximum resource demand of the task-group τ under the scheduling policy A, in any interval of time t. In other words, $dbf(t)$ indicates the maximum cumulative resource requirements (recurring instances) of tasks that have both the arrival times and deadlines within any time interval of duration t.

The demand bound function of a task-group depends on the scheduling policy used to schedule the task-group. For instance, the demand bound

function of a task-group consisting of periodic tasks under the EDF scheduling algorithm is defined as follows:

$$dbf_W(t) = \sum_{T_i \in W} \left\lfloor \frac{t}{p_i} \right\rfloor \cdot e_i \tag{1.1}$$

Similarly under a static priority scheduling algorithm like RM, the demand bound function of every single task depends on the critical instant demand, when it is released along with all its higher priority tasks as shown in Eq. (1.2). Thus if the dbf of every task in the group is met by the resource supply, then the task-group is schedulable.

$$dbf_{FP}(T_i, t) = e_i + \sum_{T_k \in V(T_i)} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \tag{1.2}$$

where $V(T_i)$ is the set of all tasks having equal or higher priority than T_i.

For a task T_i over a resource model R, the worst-case response time $r_i(R)$ of T_i can be computed as follows:

$$r_i(R) = min\{t\} : dbf_{FP}(T_i, t) \leq sbf_R(t) \tag{1.3}$$

1.2.3 Schedulability analysis

Given the task group $\tau = \{T_i = (e_i, p_i)\}_{i=1}^{n}$, the scheduling policy A and the resource partition R where the task group has to be scheduled, it is possible to derive the *demand bound function* $dbf(t)$ of the workload τ given the scheduling policy A, as discussed in the previous section. From the resource partition parameters, the *supply bound function* $sbf(t)$ of the partition, that guarantees the minimum resource supply within any t time units is calculated.

The schedulability analysis now just involves checking if the minimum resource supply can meet the maximum demand imposed by the task group at all times.

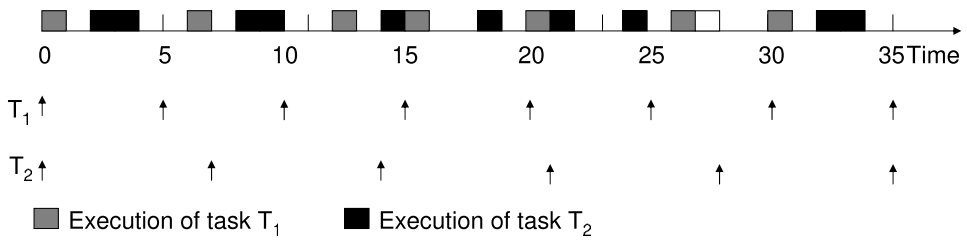

Fig. 1.7: Scheduling task group $\{(5,1),(7,2)\}$ under EDF policy on partition Π_2

For a dedicated resource, the demand bound function has to be satisfied by the resource at all times. The schedulability condition for a dedicated resource is as follows:

$$\forall t, dbf_{\tau,A}(t) \leq t \tag{1.4}$$

On a partitioned resource the schedulability condition becomes:

$$\forall t, dbf_{\tau,A}(t) \leq sbf_R(t) \tag{1.5}$$

For EDF, the schedulability needs to be checked only up to the $2 \times LCM_\tau$, where LCM_τ is the *lcm* of the periods of the tasks in the workload.

$$\forall 0 \leq t \leq 2 \cdot LCM_\tau : dbf_{\tau,A}(t) \leq sbf_R(t) \tag{1.6}$$

For fixed priority scheduling, the schedulability condition is given as follows:

$$\forall T_i \in \tau, \quad \exists 0 < t \leq p_i : \quad dbf_{FP}(\tau, T, i) \leq sbf_R(t) \tag{1.7}$$

Example 4: *Consider two periodic tasks* $W_1\{(5,1),(7,2)\}$*, scheduled on partitions* Π_2 *of Example 2 under EDF. The timing diagram of task execution would appear as shown in Figure 1.7. The task group executes on the partition as if it were executing on a dedicated resource.*

1.2.4 Hierarchical scheduling framework

One approach to constructing a hierarchical scheduling framework as discussed in the previous section is through temporal resource partitioning. In

this approach, the second-level scheduler is responsible only for assigning partitions (collection of time slices) to the task groups and does not require information on the timing parameters of the individual tasks within each task group. The schedulability analysis of tasks on a partition, depends only on the partition parameters. This enforces the desired separation of concerns; scheduling at the resource partition (task-group) level and at the task level are isolated at run time.

A structural overview of virtual resource provisioning is shown in Figure 1.8. At the top level is the physical resource, which is partitioned into several virtual resources; then, each virtual resource is partitioned recursively into several lower level virtual resource. Eventually each virtual resource will be associated with one task group which consists of one or more tasks. The mapping relation between resource and partitions is 1-to-n; that between the partition and task group is 1-to-1 and the task-group to tasks is 1-to-n again. Two classes of resource scheduling problems may be identified in this structure: one is how to schedule the tasks within a task group; the other is how to schedule virtual resources on a physical resource.

Thus the real-time virtual resource abstraction supports a universal paradigm for separating the concerns of proving the correctness of individual applications and ensuring that the aggregate resource requirements of the applications can be met as follows: First, determine the timing precision of event occurrences that is required to establish the desired timing properties of individual applications. Second, use formal methods or otherwise to prove the correctness of each individual application by pretending that it has access to a dedicated resource which operates at a lower rate but has a delay bound that is adequate for the precision requirements. Third, show that the resource partition scheme used by the run-time system satisfies

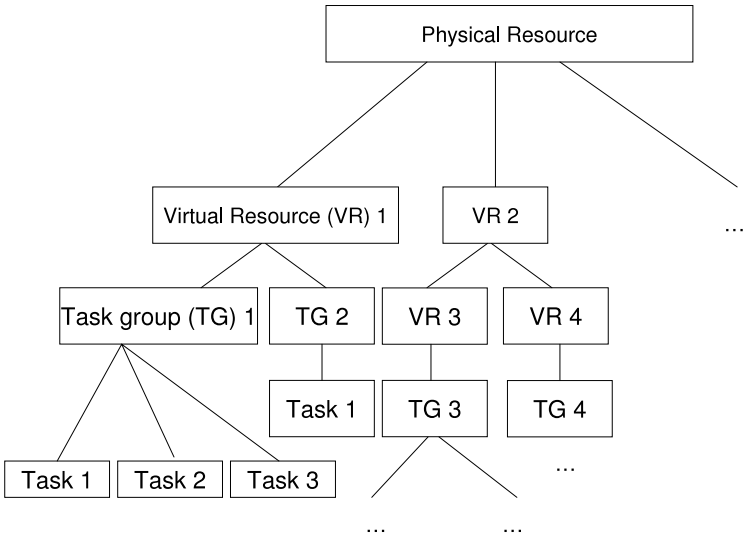

Fig. 1.8: Overview of the virtual resource structure

the delay bounds. In a nutshell, the real-time virtual resource abstraction gives us a handle on correctly composing applications with disparate timing requirements and shared resources.

The real-time virtual resource framework depicts only the structuring of the resource partitions, and hides the scheduling policy used to schedule the partitions at every level in the hierarchy. Taking into account the scheduling policy used to schedule the resources, the framework can then be explicitly modeled as in Figure 1.2.

At the leaf-levels, the task groups are scheduled on partitions as explained in the previous section. The next problem is resource-level scheduling, that is scheduling partitions within the parent partition. As seen in Figure 1.2, the hierarchical framework is structured as a hierarchy of scheduling models, where a scheduling model M is defined as (W, R, A), where W is the workload model that describes the workloads (applications) supported in the scheduling model, R is a resource model that describes the resources available to the scheduling model, and A is a scheduling algorithm that defines

how the workloads share the resources at all times. Given multiple scheduling models $M_1, \cdots, M_n$, the goal is to derive the parent scheduling model $M_P(W_P, R_P, A_P)$. Assuming A_P is given, and we consider periodic-resource model, then the resource model of the child scheduling model $\Gamma_i(\Pi_i, \Theta_i)$ is mapped to a periodic task $T_i(p_i, e_i)$ and the collection of these periodic tasks forms the workload W_P of the parent scheduling model, such that $W_P = \{T_1(\Pi_1, \Theta_1), T_2(\Pi_2, \Theta_2), \cdots, T_n(\Pi_n, \Theta_n)\}$. In short, the child partitions are treated as the individual tasks of the workload of the parent scheduling model. If the resource model is not expressed as a periodic resource, then there has to be a way of calculating the demand of the resource model, when treated as the workload.

1.3 Our approach to hierarchical scheduling

The previous approaches to hierarchical scheduling framework, have been targeted toward deriving the best resource parameters at the top-level of the hierarchy such that all the child scheduling models are schedulable, and exact schedulability conditions for scheduling task groups on every resource reservation scheme proposed. The overall approach to the construction of the hierarchical scheduling framework is to solve the component demand abstraction problem at every level of the hierarchy to derive the parent scheduling model. In this manner the construction and analysis of the hierarchical framework is bottom-up. In some cases it would also be useful to construct the hierarchical framework in a top-down manner, having fixed the physical resource. Then the resource reservation scheme and the resource-level scheduling policies would have to be derived such that the task groups are all schedulable. Our hierarchical scheduling framework is an improvement over the previous models in the following aspects:

- **Generalized resource model:** The resource models proposed in the previous frameworks, have been periodic or sporadic in nature. A more generalized resource model like the bounded-delay resource model, gives pessimistic estimates of the minimum resource supply. Our framework combines the advantages offered by the bounded-delay resource model with tighter supply bounds from the real-time calculus framework.

  In the construction and analysis of the hierarchical scheduling framework, previous approaches solve the component demand abstraction problem at every level of the hierarchy and work bottom-up. We work from the physical resource level and from the task-level, that is both bottom-up and top-down depending on the scheduling policies enforced at every level of the hierarchy. Compromising on the isolation property of hierarchical scheduling framework, we aim to maximize resource utilization by considering the demand imposed by the task groups in the schedulability analysis, instead of the abstracted resource supply.

- **Remaining computation capacity:** In schedulability analysis, the minimum resource supply and maximum resource demand are considered and in reality, this leads to over-provisioning of resources. It would be useful to calculate the remaining computation capacity left over from a resource partition and pass it on to some other resource partition in the same level, or at a lower-level in the hierarchy.

  The remaining computation capacity could also be used to schedule some other task group on the same level. We could schedule a mix of resource partitions and task groups within the same resource partition. This flexibility makes the scheduling framework very powerful since we

tend to maximize resource utilization.

- **Modeling dependencies:** Some existing concurrency models have been proposed to handle the data dependencies among the tasks scheduled on different resource partitions. We propose a solution to this problem using the real-time calculus framework and illustrate that the behavior of our model is typically similar to the existing semantics and the exact temporal properties of the producer and consumer tasks involved in a dependency are explicitly derived.

- **Generalized task models:** The task models considered in the previous approaches are very restrictive. Task groups typically consist of periodic independent tasks. In reality, not all applications are modeled as a set of periodic tasks. It would be desirable to have powerful task models, characterizing applications with precise temporal parameters. Task models that handle conditional branching and accommodate data dependencies among tasks are very powerful. Incorporating such task models into the scheduling framework, involves deriving the schedulability condition for scheduling the task group on a partitioned resource model. In our hierarchical framework, we illustrate the use of a couple of powerful task models and present the exact schedulability conditions.

1.4 Report organization

The rest of the report is organized as follows: Chapter 3 is organized into three main sections: section 3.1 discussing some generalized resource models, section 3.2 describing the workload model that consists of the event model explained in section 3.2.1 and two generalized task models in section

3.2.2. The focus of this work, that is the construction of the hierarchical framework rests on the concepts explained in sections 3.1 and 3.2, and is illustrated in section 3.3. The bounded delay resource partition model and our resource model based on real-time calculus are described in sections 3.1.1 and 3.1.2 respectively of section 3.1, with a comparative analysis of these two approaches presented in section 3.1.3. The recurring real-time task model is described in section 3.2.2, the stream-based task model in 3.2.2 and the schedulability conditions are presented in section 3.2.2. The hierarchical scheduling framework is introduced in section 3.3 and handling various cases of data dependencies is discussed in section 3.4. An illustrative case-study of a complex real-time system is presented in section 3.5 and the evaluation of the framework is presented in section 3.5.1. Finally the contribution of this work is summarized in chapter 4, and future directions are discussed in section 4.1.

2. RELATED WORK

A general methodology for temporal protection in real-time systems is the resource reservation framework. The basic idea is that each task is assigned a server that is reserved a fraction of the processor bandwidth. Thus, by using a resource reservation mechanism, the problem of schedulability analysis is reduced to the problem of estimating the computation time of the task without considering the rest of the system. The various resource reservations schemes proposed and their use in hierarchical scheduling is discussed in this chapter.

2.1 Server abstractions

Given the pair (Q, P) assigned to the server, the worst execution times, periods and deadlines of the tasks, and the internal scheduling algorithm for the task group, it is possible to perform a schedulability analysis of a group of tasks on a server. Saewong et al. [23] presented a response time analysis of such a hierarchical system. The problem of finding the server parameters (Q, P), given the group of tasks, such that the task group is schedulable on the server is answered by [14][15].

Period, sporadic and deferrable servers The simplest resource reservation policy is the periodic server abstraction [14][15]. Each application is assigned a server that is characterized by the pair (Q, P), with the meaning that the server gets Q units of execution every P units of time. Given

the parameters (Q, P), it is possible to perform a schedulability analysis of a group of tasks on the server. The global scheduling mechanism decides when to schedule the servers and the selected server, by using the local scheduling mechanism, decides which application task will be executed next. Given an application composed of periodic (or sporadic) tasks, scheduled by a fixed priority local scheduler, it is possible to find a class of parameters (Q, P) that make the application schedulable.

In *deferrable servers* that are characterized by (C_s, T_s, D_s), where C_s is the resource reservation in time units, every recurring time interval T_s before a relative deadline D, the budget is replenished (back to C_s) at the beginning of every server period. As long as the server has enough budget it could use it anytime within its period to service requests. This approach suffers from a phenomenon called *deferred execution effect* or *back-to-back execution phenomenon* (also 'jitter effect'). In particular, an aperiodic task could arrive at the latter part of the server's period and utilizes its budget just in time for a new server period. Since the budget is replenished, the aperiodic task grabs the resource (say CPU) for another C_s units of time. The net result is that $2 \times C_s$ units of time were used over two periods 'back to back'.

A *Sporadic Server* is very similar to the deferrable server, except that the budget is not replenished at the beginning of each period, but after T_s units of time elapse after the budget is consumed.

Constant Bandwidth Server (CBS) To provide efficient run-time support for continuous media (CM) applications, a constant bandwidth server [3] has been proposed. In this any task τ_i consists of a sequence of jobs $J_{i,j}$, where $r_{i,j}$ denotes the arrival time (or request time) of the j^{th} job of task τ_i. Each hard real-time task is characterized by two additional param-

eters (C_i, T_i), where C_i is the WCET of each job and T_i is the minimum inter-arrival time between successive jobs, so that $r_{i,j+1} \geq r_{i,j} + T_i$. The relative deadline of the job is given by $d_{i,j} = r_{i,j} + T_i$. For a soft real-time task, C_i denotes the average execution time of each job and T_i represents the desired activation period between successive jobs. The soft deadline is again $d_{i,j} = r_{i,j} + T_i$. The tardiness $E_{i,j}$ of a job $J_{i,j}$ is defined as $E_{i,j} = max\{0, f_{i,j} - d_{i,j}\}$, where $f_{i,j}$ is the finishing time of job $J_{i,j}$. To integrate hard and soft real-time tasks in the same system, hard tasks are scheduled by EDF algorithm based on their absolute deadlines, whereas each soft task is handled by a dedicated server, the Constant Bandwidth Server (CBS).

CBS is characterized by a budget c_s and an ordered pair, (Q_s, T_s), where Q_s is the maximum budget and T_s is the period of the server. At each instant a fixed deadline $d_{s,k}$ is associated with the server and initially $d_{s,k}$ is set to 0. Each served job $J_{i,j}$ is assigned a dynamic deadline $d_{i,j}$ equal to the current server deadline $d_{s,k}$ and the budget c_s is decreased by the amount of time the job executes. When the budget is exhausted it is recharged to the maximum value Q_s and a new server deadline is generated as $d_{s,k+1} = d_{s,k} + T_s$. When a new job arrives when the server is active the job is enqueued. When a job finishes execution, the next pending job in queue, is served using the current budget and deadline, else the server becomes idle.

Total Bandwidth Server (TBS) In order to schedule aperiodic requests whose arrival times are not known and whose WCETs are known, a fixed (maximum) percentage U_s (server size) of the processor is allocated [26] to serve aperiodic jobs. When an aperiodic job arrives, it is assigned a deadline such that the demand created by all the aperiodic jobs in any feasible interval never exceeds the maximum utilization U_s allocated to aperiodic jobs.

Proportional share (PS) schedulers Proportional share (PS) schedulers [10] are quantum-based approximations of fair schedulers. Some PS schedulers can guarantee that during any time interval of length t, a thread with a share s of the total processor bandwidth will receive at least $st - \delta$ units of processor time, where δ is an error term that depends on the particular scheduling algorithm. This guarantee is called the proportional share bound error. PS schedulers provide stronger guarantees to applications than do traditional time-sharing schedulers: they allocate a specific fraction of the CPU to each thread and have clear semantics during underload, i.e. unused and unallocated capacity is distributed across processes in proportion to their allocation.

Hierarchical extensions to the server abstractions An extension to the CBS was proposed [13] to achieve per-thread performance guarantees and inter-thread isolation in certain kinds of multi-threaded real-time computer systems. Each thread $T_j = (U_j, P_j)$ is characterized by the worst-case utilization U_j and period P_j. Each thread T_j generates a sequence of jobs $J_j^1, J_j^2, \ldots,$, with job J_j^k becoming ready for execution at time $a_j^k (a_j^k \leq a_j^{k+1}, \forall j, k)$ and having an execution requirement of e_j^k time units. If F_j^k denotes the time instant at which job J_j^k would complete execution, if all jobs of thread T_j were executed on a dedicated processor of capacity U_j and f_j^k denote the time instant at which J_j^k completes execution under CBS, it is guaranteed that $f_j^k < F_j^k + P_j$, i.e, each job of thread T_j is guaranteed to complete execution under CBS no more than P_j time units later than the time it would complete if executing on a dedicated processor.

In [9], the authors studied systems that are scheduled using fixed priority pre-emptive scheduling at both local and global scheduling levels. They gave the exact response time analysis for hard real-time tasks scheduled under

periodic, sporadic and deferrable Servers. This analysis provided a reduction in the calculated worst-case response times of tasks when compared to previous published work. A similar improvement was also apparent in the server capacity and replenishment periods deemed necessary to schedule a given task set. In a comparative study of periodic, sporadic and deferrable Servers in terms of their ability to guarantee the deadlines of hard real-time tasks, it was found that the periodic server completely dominates the other server algorithms on this metric. They extended the schedulability analysis to hierarchical systems where tasks in disparate applications are permitted to access mutually exclusive global shared resources. This work also clearly illustrated the advantages of choosing server periods that are exact divisors of the task periods, thus enabling tasks to be bound to the release of the server.

Hierarchical loadable scheduler (HLS) Hierarchical loadable schedulers (HLS) [22] is a system that supports composition of scheduling behaviors using hierarchical scheduling, as well as the ability to provide guaranteed scheduling behavior to application threads at the leaves of the scheduling hierarchy. A guarantee is provided by a scheduler to a scheduled entity (either a thread or another scheduler) using a virtual processor (VP). A guarantee is a contract between a scheduler and a scheduled entity regarding the distribution of CPU time that the VP will receive for as long as the guarantee remains in force.

A hierarchy of schedulers and threads composes correctly if and only if firstly, each scheduler in the hierarchy receives a guarantee that is acceptable to it and secondly, each application thread receives a guarantee that is acceptable to it. The set of guarantees that is acceptable to a scheduler is an inherent property of a scheduling algorithm. If acceptable guarantees

can be assigned to all application threads, the hierarchy is compositionally correct, otherwise it is not.

Reservation-based operating systems provide applications with guaranteed and timely access to system resources. They provide temporal isolation, i.e. prevent timing misbehavior of one task from interfering with other tasks. The abstraction of resource isolation should be extended to cover not just a task boundary, but also collection of tasks, applications, users, network flows or other high-level resource management entities. A hierarchical reservation model [23] was proposed where any resource management entity, such as a task, an application and a group of users, could create a reservation to obtain resource and/or timing guarantees. Resource requests will be granted only if the new request and all current allocations can be scheduled on a timely basis. Each reservation can then recursively create child reservation and become a parent reservation. Different parent reservations can specify different scheduling policies to suit the needs of their respective descendants. The resource isolation mechanism will ensure that each child reservation cannot use more resources than its allocation. However if a child reservation under uses its resource allocation, those unclaimed resources can be assigned to its siblings. The key challenge of such a system is the capability to grant throughput and latency guarantees to each node in the hierarchy based on its scheduling policy.

Hierarchical deadline-monotonic scheduler (HDM) The hierarchical deadline-monotonic scheduler, (HDM) [23], makes some assumptions about the task models to obtain analytical results. A resource kernel (RK) allows applications to specify only their resource and time-line demands leaving the OS kernel to satisfy those demands using (hidden) resource management schemes. The resource specification uses the (C, T, D) model for

the reservation of C units of resource time every recurring time interval T before a relative deadline D. To provide temporal isolation among tasks, the resource kernel monitors and enforces actual resource used by each reservation. In the HDM scheduler mode, a child reserve R_i can be created only if its parent resource $R^p(C^p, T^p, D^p)$ has been guaranteed access in a resource kernel.

The schedulability analysis of HDM involves considering a parent reserve to be a server that serves requests from its child reserves. The *deferrable server* with (C_s, T_s, D_s) allows any of its clients to use its resource any time within the current period T_s until its budget C_s is exhausted. In order to avoid back-to-back the *sporadic server* technique is used but with higher implementation complexity. Instead of providing periodic replenishment to the server budget, the sporadic server replenishes the budget based on the actual time of prior resource usage, such that in the worst-case, a child reserve behaves like a classic Liu and Layland periodic task.

2.2 Composite schedulability analysis with utilization bounds

Another way of performing composite schedulability analysis is by using utilization bounds. This is more useful for on line admission test in deciding whether an arriving event could be admitted into the system such that all other tasks already queued can still meet their deadlines. In [29], the authors derive schedulability bounds for a range of workload patterns and schedulers. By proper parameterization existing utilization bounds for different priority assignments and task releasing patterns could be derived from the results presented. The approach adopted here is based on network calculus to derive a general schedulability bound by solving a minimization problem over the entire task set population. The problem here is that the task set population

considered is potentially infinite and the solution requires deriving the worst case schedulability bound among the entire set.

2.3 Interface-based scheduling approach

Yet another approach to compositional scheduling is to define a component interface [28] that describes how the component can be used. The interface specification is given by the *input assumptions* that describes the expectations a component has about other components in the system and *output guarantees* that tell other components in the system what they can expect from this component. The major goal of a component interface is to provide enough information to decide whether two or more components can work together properly. But the interface-based real-time system design approach assumes that we are given a set of the load components and processing components and the goal is to build the interface of a complete system by composing all interfaces of the different components. This approach would be exhaustive for a system with large number of components, as it would require specifying all the parameters (load, processing) for each of the components. The components are considered to be made up of independent tasks, with periodic or periodic with jitter behavior and deadlines equal to, less than or greater than the period of invocation. Although the model guarantees tight bounds, it does not model data dependencies among tasks in the components.

Along the lines of resource partitioning schemes for hierarchical scheduling, the notion of real-time virtual resources as described in section 1.2.4 of chapter 1. The component demand abstraction problem derives the parameters for the interface between the parent and child scheduling models. The hierarchy is constructed bottom-up deriving the interface parameters at

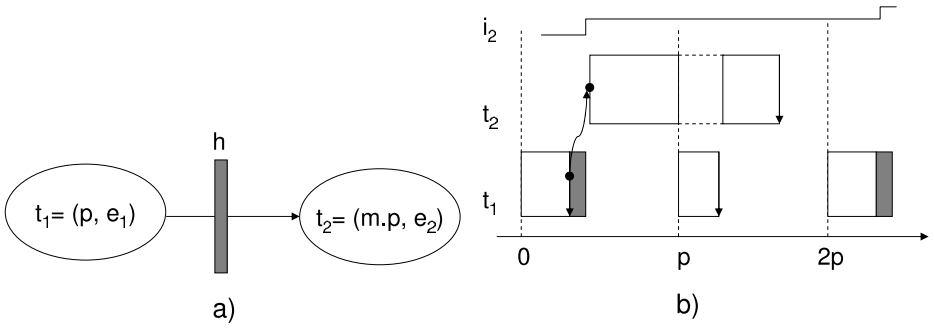

Fig. 2.1: a) Task graph with tasks $t_1 = (p, e_1)$ and $t_2 = (m \cdot p, e_2)$ b) Task and input signal line for $m = 2$

every level until the parent resource allocation is derived. The models proposed based on the virtual resources, are restrictive in the workload models and scheduling algorithms that can be incorporated. The bounded-delay resource partition model gives more flexibility on partition-level schedule. The partition delay parameter is most useful for bounding the separation between event pairs and thus a bound on the output jitter is achieved.

2.4 Handling data dependencies among tasks and task groups

Most hierarchical scheduling frameworks accommodate only task groups that consist of independent periodic tasks. There are two types of dependencies: one is between tasks in a group i.e., inter-group dependencies and the other is between tasks in different groups i.e., intra-group task dependencies. There have been various approaches suggested to handle such data dependencies. In [19] the hierarchical framework based on the periodic resource model, is analyzed in the presence of data flow constraints, with two natural semantics, namely, RTW (Real-Time Workshop) and LET (Logical Execution Time).

RTW [2] is a code generation tool for MATLAB/Simulink models and

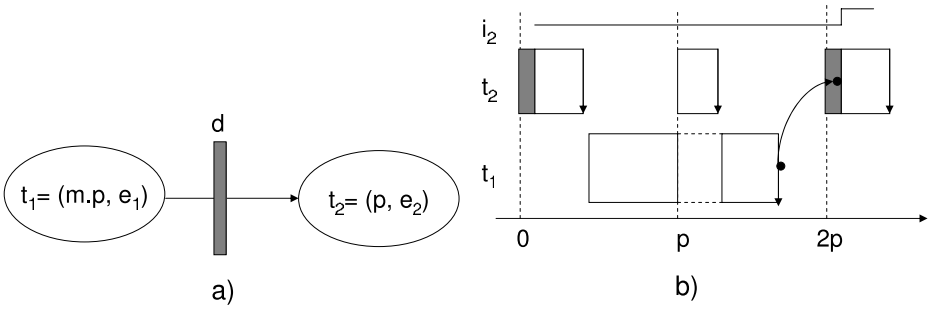

Fig. 2.2: a) Task graph with tasks $t_1 = (m \cdot p, e_1)$ and $t_2 = (p, e_2)$ b) Task and input signal line for $m = 2$

LET is the underlying concurrency of the Giotto [6] programming language based TTA. These semantics are used to handle multi-rate task programs, where the producer and consumer tasks have different sampling rates. A hard real-time system based on the sensor-actuator model is a typical system where such a scenario occurs. The sensor will be sampled at some frequency and the actuator will be sampled at a different frequency. It is necessary to ensure data integrity between the sensor and the actuator, so that a successive triggering of the sensor does not overwrite the values supplied to the actuator before the actuator finishes execution. It is also necessary guarantee determinism, so that every successive activation of the actuator, uses values from the corresponding activation of the sensor.

It is assumed that the task groups are specified with an underlying task precedence graph giving the producer-consumer relationship between tasks. RTW generates multi threaded code for a given task graph, one thread per sample time of the graph. This code runs on an RTOS that offers priority based pre-emption mechanism. Each task is assigned to a thread based on its period, and the schedule within a thread is constructed from the task dependencies. The rate-monotonic (RM) scheduler invokes the generated code, enabling pre-emption between rates. The data transfer between the

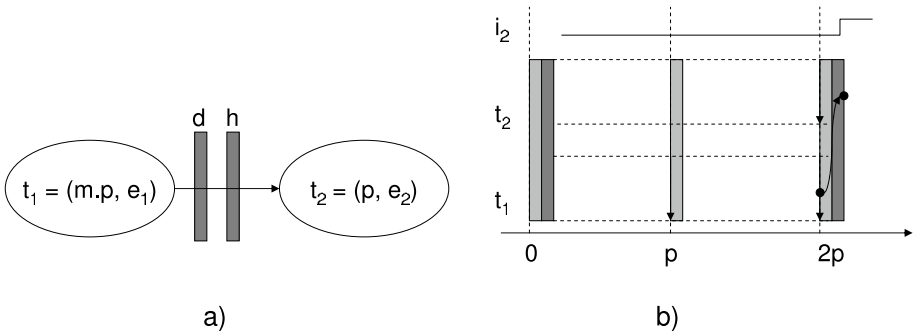

Fig. 2.3: a) Task graph with tasks $t_1 = (p, e_1)$ and $t_2 = (m.p, e_2)$ b) Task and input signal line for $m = 2$

producer and consumer is handled by introducing rate-transition blocks, which acts as a hold/delay block to retain the values written by the producer and pass it on to the consumer, thus guaranteeing data integrity and determinism. An example of such data transfers is shown in Figures 2.1 and 2.2. In RTW, another mode of handling dependencies among tasks called the asynchronous mode [1] guarantees data integrity but does not ensure determinism.

The LET semantics on the other hand, logically extends the execution time of the producer and consumer tasks and makes the exact time of data transfer predictable. The logical execution time extends from the release to termination time of tasks, data transfers between tasks happens exactly at the termination time of the producer task and release time of the consumer task. An example of data transfer using LET semantics is shown in Figure 2.3. The LET semantics does not prescribe any particular scheduling strategy. EDF or a simple round-robin scheduling policy could be applied as long as the generated code satisfies the LET assumption.

Since the LET logically extends the execution time, the end-to-end latency is higher than in the case of RTW, when there are chains of dependent

tasks. Since RTW used fixed priority scheduling of tasks to maintain the order of task execution, and LET can be implemented with EDF where precedences are ignored, benefits of better schedulability is turned into tighter abstraction in compositional scheduling. So in the trade-off between latency and composability, it becomes desirable to exploit the composability property for tighter abstraction and ease of analysis.

Another approach to modeling concurrency [17] [16] has been to adopt existing resource sharing protocols like the priority inheritance or priority ceiling protocols. The BandWidth Inheritance (BWI) protocol proposed in [17], extends the CBS algorithm [3] to real-time tasks that can access shared resources via critical sections, by using a technique derived from the Priority Inheritance Protocol (PIP).

In [4] the authors derive exact expressions for the load of a task group including the overhead of any synchronization protocol that is used to implement resource sharing. But these approaches give pessimistic bounds on the waiting time which in turn is derived from the synchronization protocol used to implement resource-sharing.

3. GENERALIZED HIERARCHICAL SCHEDULING FRAMEWORK

The hierarchical scheduling framework is constructed from the resource model used to parameterize the service allocation, and from the workload model used to describe the temporal properties of the tasks and task-groups sharing the resource. The resource and workload models are described in sections 3.1 and 3.2 respectively. These two sections serve as background to the construction of hierarchical scheduling framework described in section 3.3 and section 3.4 illustrates how techniques from real-time calculus are used to model intra-group and inter-group task dependencies. Finally in section 3.5 we present an illustrative case-study of a real-time system using hierarchical scheduling.

3.1 Resource model

The construction and analysis of the hierarchical scheduling framework is primarily dependent on the resource model used to partition resources. In section 3.1.1, we describe a generalized resource partition model called the bounded-delay resource model proposed by Mok and Feng in [20]. Section 3.1.2 illustrates the service curve representation of resources in the real-time calculus framework and section 3.1.3 draws up a comparison of the bounding functions of both models. We illustrate that the bounded-delay model can be expressed as service curves and vice versa, and also prove that the service

curves give tighter bounds on resource supply. A detailed description of the bounded-delay resource model and service curve representation follow.

3.1.1 Bounded-delay resource model

The simplest way to partition resources is to assign a fraction of the physical resource each to the task groups scheduled on the resource similar to the GPS [21] approach. But a practical implementation of such a resource reservation scheme is infeasible and some approximation techniques have been proposed to make this theoretical resource model amenable to implementation. Based on this fractional resource model, Mok and Feng [20] proposed the bounded-delay resource model where the fractional resource allocation is bounded by a delay parameter.

According to the bounded-delay resource model, if a task group has been promised a fraction a of a resource, it will receive at least $a \times t$ units of the resource's time in any interval of $t + \Delta$ for any value of t, where Δ is a design parameter, a non-negative number specified by the programmer. The parameter Δ, called the delay bound of the resource partition is enforced by the operating system. This approach, directly bounds the jitter on the rate of service provision of the partitioned resource. Thus a fractional resource specification is mapped onto a partitioned resource model by adding a parameter called the *delaybound*. Thus if we know that an event e will occur within t time units from another event e', assuming that the resource operates at a uniform rate, and event occurrence depends only on resource consumption, then e and e' will be apart by at most $t + \Delta$ time units in real-time. This permits real-time tasks to be scheduled as if they execute on dedicated CPUs of almost uniform speed modulo the delay bound Δ.

The delay bound can be specified to stay within bounds of the jitter

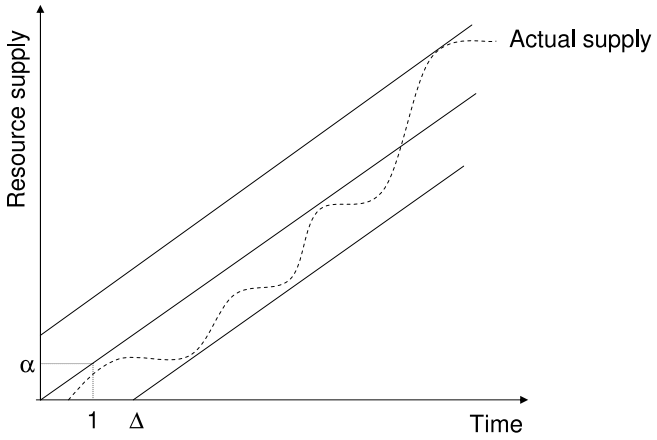

Fig. 3.1: Bounded-delay resource model $\Pi(a, \Delta)$

tolerances of the task group. The scheduler, partitions the shared resource to satisfy the delay bound requirements of the different task groups. The formal definitions of the bounded-delay resource model follow.

Definition 9: The Partition Delay Δ is defined as the smallest d so that for any t_0 and t_1, the following inequality holds:

$$(t_1 \geq t_0), (t_1 - t_0 - d)a(\Pi) \leq (S(t_1) - S(t_0)) \leq (t_1 - t_0 + d)a(\Pi) \quad (3.1)$$

where $S(t)$ is the supply function as defined in Definition 3.

Definition 10: A Bounded Delay Resource Partition Π is a tuple (a, Δ) where a is the rate of the partition and Δ is the partition delay.

The bounded-delay resource is one that is available at a rate of a, with a maximum delay of Δ time units. That is, in any interval of t time units, $a \times t$ units of resource allocation is guaranteed in a maximum of $t + \Delta$ time units.

Definition 11: The supply bound function $sbf(t)$ of a bounded-delay resource model $\Pi(a, \Delta)$ gives the minimum resource supply guaranteed in any

interval of t time units and is given by the equation:

$$sbf_{\Pi}(t) = \begin{cases} a(t-\Delta) & \text{if } (t \geq \Delta) \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

An illustration of the bounded-delay model is given in Figure 3.1.

3.1.2 Resource model using service curves

The resource model based on real-time calculus is characterized by the service function $C(t)$ analogous to the supply function $S(t)$ of Definition 3. The *service function* $C(t) \geq 0$ represents the maximum amount of computation that could be delivered up to time t. Usually processors serving task requests are considered to be able to deliver a constant amount of computation per unit time. The more general concept of a delivery curve, which is inspired by the use of service curves in the networking domain, characterizes the processor via the minimum amount of computation that is guaranteed to be delivered in an interval of length δ (actually parameterized as Δ in the real-time calculus framework, but we denote it as δ in order to differentiate it from the partition delay parameter) starting at any arbitrary point in time.

Definition 12: A computing resource can be described by a service function C, where $C(t)$ denotes the number of units of computation that could have been delivered in the interval $[0, t)$.

Definition 13: The service curve denoted as β, of a service function C is a non-decreasing, non-negative function which satisfies $C(t) - C(s) \geq \beta(t-s), \forall s \leq t$. The upper and lower service curves $\beta^u(\delta)$, $\beta^l(\delta) \in \mathbb{R}^{\geq 0}$ of a service function $C(t)$ satisfy:

$$\beta^l(t-s) \leq C(t) - C(s) \leq \beta^u(t-s), \forall s, t : 0 \leq s \leq t \tag{3.3}$$

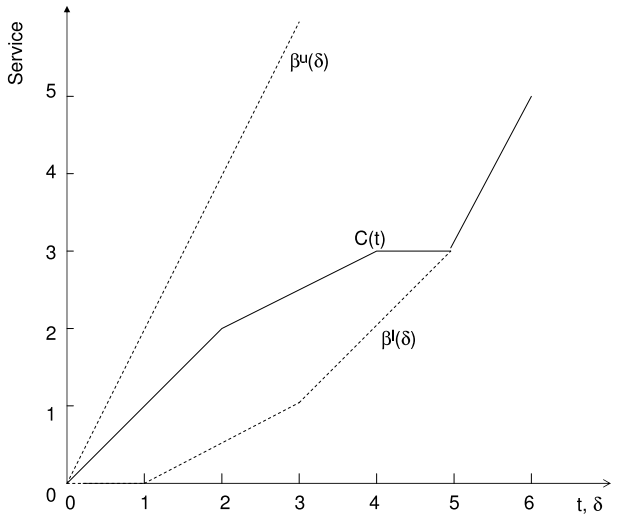

Fig. 3.2: Upper and lower service curves corresponding to a function $C(t)$ of a resource node that processes 1 event per time unit in the interval [0, 2], only 1/2 an event in [2, 4] (e.g. because of a processor share with another task), has no computing capability in [4, 5] (e.g. because of an interrupt handler), and can process 2 events per time unit starting from time 5.

The service curves $\beta^u(\delta)$ and $\beta^l(\delta)$ can be interpreted as the maximum and minimum available computing service within any time interval of length t, respectively. Therefore, given the service function C, we have $\beta^l = min_{u \geq 0}\{C(\delta+u) - C(u)\}$ and $\beta^u = max_{u \geq 0}\{C(\delta+u) - C(u)\}$. Figure 3.2 shows an example of upper and lower service curves, corresponding to a service function $C(t)$.

In case of an unloaded processor, both resource curves may be equal and represented as straight lines, i.e. $\beta^u = \beta^l = \delta$. A more complicated example is a time division multiplex bus with the period q. Within this period, a time units are available for the respective communication channel at a fixed offset and b bytes can be communicated within the available a time units. The corresponding service curves are shown in Figure 3.3.

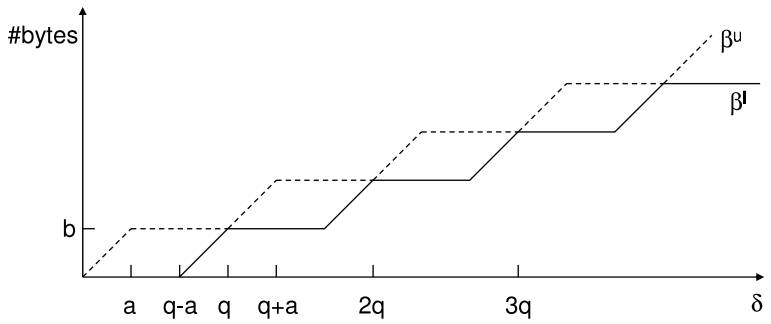

Fig. 3.3: Upper and lower service curves corresponding to a TDMA communication bus

For a processor we could apply the TDMA based scheduling policy, and partition the resource into two, assigning weights w_1 and w_2. Firstly the sum of the weights $w_1 + w_2 \leq 1$ and the scheduler works as follows: time is divided into periods of length P and within any period $w_1 \times P$ consecutive units of the processor's time is allocated to partition 1 and $w_2 \times P$ consecutive units are allocated to partition 2. Let us first assume that the period P is infinitesimally small and we neglect the effects of a finite sampling of the processor cycles. Then the service offered to the two partitions in terms of processor cycles is calculated as follows: If c is the number of processor cycles available from the processor per unit time (i.e. c is the frequency of the processor), then the total service offered by the processor is given by the service curve $\beta^l(\delta) = \beta^u(\delta) = c\delta$.

The service offered to partition 1 can be specified by the service curve $\beta_1^l(\delta) = \beta_1^u(\delta) = w_1 c\delta$. Similarly, the service curve for partition 2 is $\beta_2^l(\delta) = \beta_2^u(\delta) = w_2 c\delta$. Therefore, the lower and the upper service curves for both partitions coincide and are straight lines with slopes $w_1 c$ and $w_2 c$ for partitions 1 and 2 respectively. For instance if the number of processor cycles per period is 1.5×10^6 and the weights are assumed to be $w_1 = 0.7$, $w_2 = 0.3$, the upper and lower service curves of the partitions coincide as

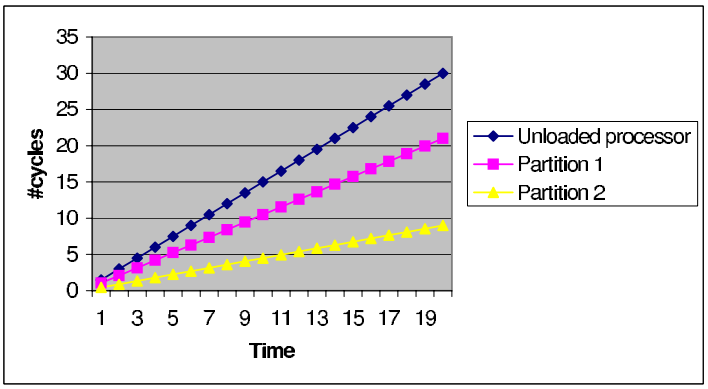

Fig. 3.4: Two resource partitions with weights 0.7 and 0.3 are created from a processor that offers 1.5×10^6 cycles every unit of time.

shown in Figure 3.4. This is fractional resource allocation also called fluid model of service allocation similar to the Proportional Share scheduler and GPS approach.

When the period P has a finite value, the service curves take the form of a *staircase function* and the lower and upper curves no longer coincide and each partition would have service curves similar to the ones shown in Figure 3.3.

3.1.3 Bounded-delay resource and service curves - A comparison

The bounded-delay resource model could be expressed in terms of service curves and vice-versa. Given the service function of a resource, it is possible to derive the delay bound Δ satisfying the inequality of Eq. (3.1). The availability rate of the service would be the fractional availability of the resource over the entire time interval considered.

Example 5: *Consider the periodic resource pattern* $\Gamma(5, 3)$ *shown in Figure 1.5. The resource pattern guarantees 3 time units of resource allocation, every 5 time units. The maximum blocking time of such a resource is* $2 \times$

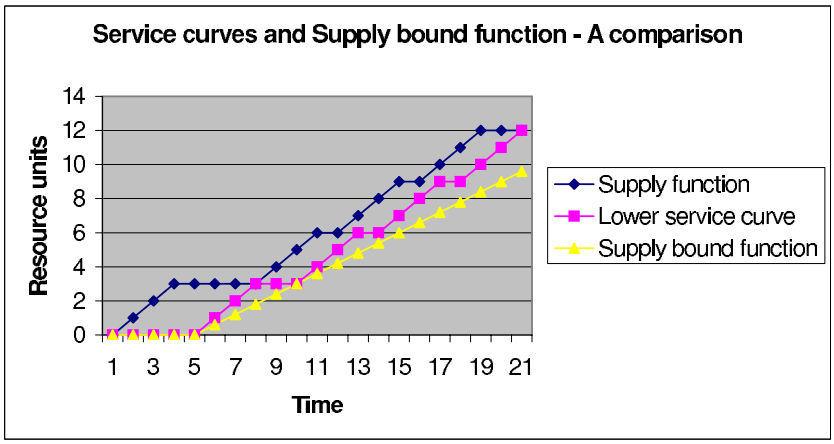

Fig. 3.5: Comparison between the supply function, supply bound function and lower service curve of the resource $\Gamma(5, 3)$

$(5-3)$ *which is 4 times units. This is evident from the interval of resource availability, in the time interval [0,10]. Thus the delay* (Δ) *of the resource is* $\Delta = 4$. *The rate (availability* a*) of such a resource is* $3/5$, *which is* 0.6. *The supply bound function of the resource can thus be calculated using Eq. (3.2). It is also possible to derive the upper and lower service curves for the same resource pattern by sliding a window of length* t *over the entire time window and recording the maximum and minimum values of service capacity, for all possible values of* t. *An illustration of the lower service curve* β^l *and the* sbf *of the resource* Γ *is shown in Figure 3.5.*

The lower bounds on resource supply as given by the supply bound function is pessimistic as can be seen in Figure 3.5.

The supply bound function (sbf) and the lower service curve (β^l) are related as follows:

$$a \times (t - \Delta) \leq \beta^l(t), \quad \forall t \geq 0 \tag{3.4}$$

In other words, $sbf(t) \leq \beta^l(t)$. By definition of sbf, we are guaranteed to receive at least $a \times (t - \Delta)$ time units of resource availability in t time units.

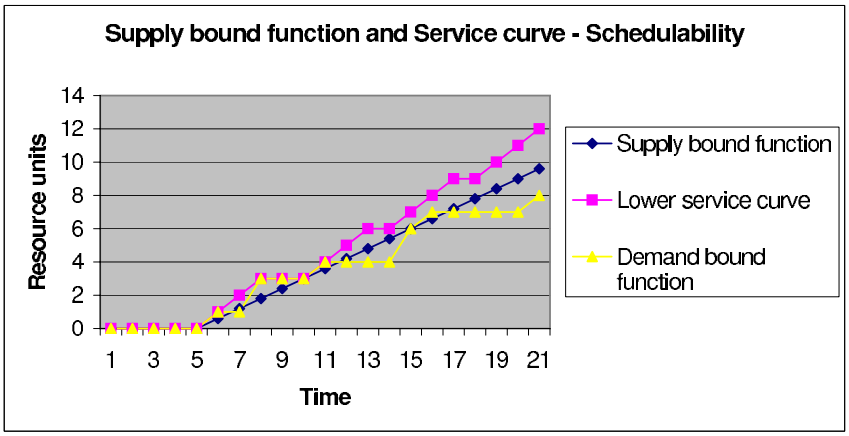

Fig. 3.6: Schedulability of $\{(5,1),(7,2)\}$ under the sbf and β^l of the resource in Example 3

If for some t_0, $\beta^l(t_0) < sbf(t_0)$, then it means the lower bound on service, i.e sbf does not hold at t_0, which contradicts the fact that sbf is the minimum service supply. Thus $sbf(t) \leq \beta^l(t),\ \forall t \geq 0$. This can also be proven by considering the delay and the longest unavailable time interval. The delay Δ is definitely greater than or equal to the longest unavailable time interval of length say t_0, $\Delta \geq t_0$. There are two cases: all time intervals t such that, $t <= t_0$ and $t > t_0$. For case 1, we can observe that $sbf(t) = 0$ by definition and $\beta^l(t) = 0$ and thus $sbf(t) = \beta^l(t)$ for all $t <= t_0$. If $t_0 <= \Delta$ then also, $sbf(t) = 0$ and thus is less than or equal to $\beta^l(t)$. But if $t > \Delta$, $sbf(t) = a \times (t - \Delta)$ and thus increases linearly with t. While sliding a window of size t to record the least supply in the range, $\beta^l(t)$, if there is a region adjacent to the longest blocking time interval of length t_0, that supplies a very low fraction of the resource, then we will see that the delay would obviously be much higher than Δ. Therefore $\beta^l(t)$ will always record a value greater than or equal to $sbf(t)$ for all values of t.

Schedulability analysis Given a partitioned resource model that can calculate its supply bound function accurately and a task group that can calculate its demand bound function accurately, it is possible to check the schedulability of the task group on the partitioned resource using the conditions presented in section 1.2.3.

Example 6: *Consider the resource given in Example 3 and a workload set given by two periodic tasks* $W = \{(5, 1), (7, 2)\}$. *The service bounds for this resource under the bounded-delay partition model and service curve representations were computed in Example 5. The demand bound function of the workload as given by Eq. (1.1) could be computed and the schedulability condition for EDF could be tested for all* t *upto* $2 \cdot LCM_W$. *As it turns out this workload is not schedulable under the bounds given by* sbf *of the bounded-delay model. But under the lower service curve* β^l, *this task group is schedulable. The demand bound function and the service bounds of both the resource models are shown in Figure 3.6.*

3.2 Workload model

The term workload model is used in [24] to describe the temporal properties of the tasks scheduled in the system. The workload model is comprised of the temporal properties of event arrivals that trigger the task in conjunction with the temporal properties of the task itself. Every task is associated with an event arrival pattern that determines the separation time between successive triggerings of the task. This event arrival pattern is described by the event model. The previous approaches to hierarchical scheduling dealt with periodic events. In our hierarchical scheduling framework, we propose the use of a generalized event model in the form of arrival curves described in section 3.2.1. Section 3.2.2 briefly describes two task models that handle

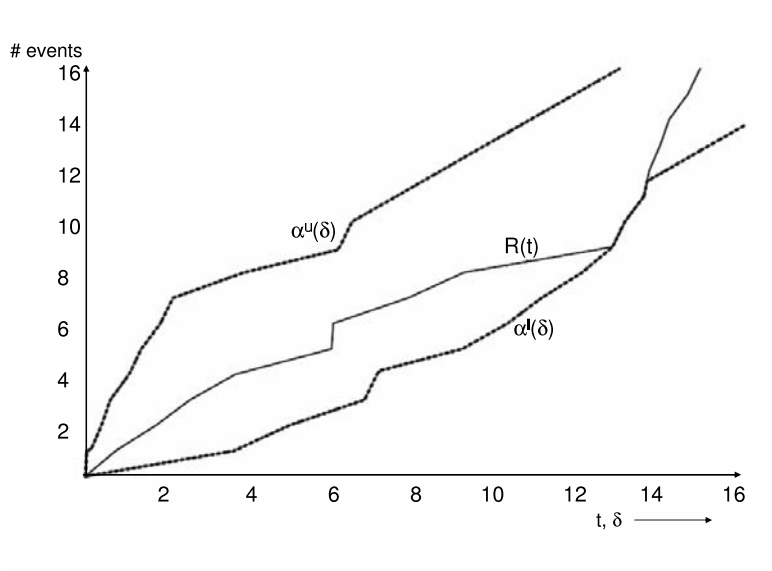

Fig. 3.7: An arrival function that has been generated from a finite size exponentially distributed event sequence with mean 1

conditional branching and precedence relationship among tasks.

3.2.1 Generalized event model - Arrival curves

A task is typically instantiated by successive timed events. Even concurrent instantiations is possible, i.e. there can be a new timed event before the previous invocation has completed execution. The timing of any event stream triggering the execution of a task is bounded by an arrival curve (event model based on real-time calculus) denoted by α. The event model given by real-time calculus framework accurately captures the characteristics of any arbitrary event stream. Given the trace of an event stream it is possible to extract a number of parameters which represent the abstract timing characteristics of the stream.

Definition 14 (Arrival function): An event stream can be described by an arrival function R where R(t) denotes the number of events that have arrived

in the interval [0, t).

Depending on the context in which these functions are used, the number of events may model the number of packets, the number of bytes or the number of instructions to be performed. Obviously, the function R(t) is nondecreasing and can be used to accurately describe the incoming events.

Definition 15 (Arrival curves): For a given event stream, let $R[s,t)$ denote the number of events that arrive in the time interval $[s,t)$. Further assume that the number of events arriving within any interval of time is bounded above by a right-continuous non-negative, sub additive function called the upper arrival curve, denoted by α^u. Similarly, a lower bound on the number of events arriving is given by a lower arrival curve α^l. R, α^u and α^l are related by the following inequality:

$$\alpha^l(t-s) \leq R[s,t) \leq \alpha^u(t-s), \quad \forall s < t \tag{3.5}$$

where $\alpha^l(0) = \alpha^u(0) = 0$.

Therefore, $\alpha^l(\delta)$ and $\alpha^u(\delta)$ can be interpreted as the minimum and maximum number of events arriving within any time interval of length δ respectively. The subadditive property of arrival curves implies that $\alpha(s+t) \leq \alpha(s)+\alpha(t)$ for all $s,t \geq 0$. Using the max-plus algebra, the maximum and minimum sub-additive functions smaller than α can be derived for upper and lower arrival curves. Therefore given the arrival function R, the corresponding curves can be computed using $\alpha^l(\delta) = \min_{u\geq 0}\{R(\delta+u)-R(u)\}$ and $\alpha^u(\delta) = \max_{u\geq 0}\{R(\delta+u)-R(u)\}$. This interpretation of the arrival curves and arrival functions should suffice for the understanding of rest of the results presented.

Standard event models like sporadic and periodic with bursts, or other event streams with known analytical behavior, are special cases of the generic

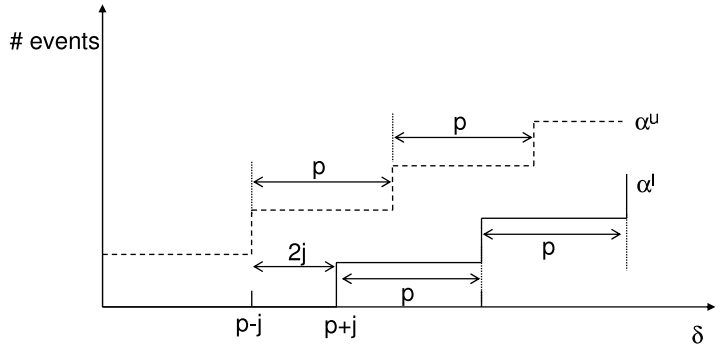

Fig. 3.8: The upper and lower arrival curves of the class of event streams with period p and jitter j

representation using arrival functions and can be represented by appropriate choice of α^l and α^u. A non-standard event model is illustrated in Figure 3.7. An example of a periodic with jitter event model expressed as arrival curves is shown in Figure 3.8.

The arrival curves can be associated with any task or task graph. For a simple task that has an execution requirement of e time units and deadline d time units, taking the execution time into account, the demand on the task becomes, $dbf(t) = e \cdot \alpha^u(t-d), \forall t > d$.

A task entering the resource node at time t for processing will experience a delay $d(t)$ bounded by the following equation

$$d(t) \leq \max_{u \geq 0} \big\{ \min\{\tau \geq 0 : dbf(u) \leq \beta^l(u+\tau)\}\big\} \tag{3.6}$$

Let us assume that the tasks execute on a resource modeled by a service curve β^l and parameterized as a bounded-delay resource model (a, Δ), where a is the availability factor of the resource. Consider a fractional resource that is uniformly available at a fractional capacity a all the time. The group of tasks each of which has a demand $dbf_i(t)$ depending on the arrival curve α_i^u associated with it, is schedulable under EDF, on β^l if the largest among the delays $d_i(t)$ for every task, when the task group is scheduled on the

fractional resource, is greater than the partition delay parameter Δ. This is similar to the virtual time scheduling of the bounded-delay model described in [20].

From the arrival curve and the processing capacity it is also possible to derive the properties of the outgoing processed stream. The upper bound on the processed event stream is denoted by $\alpha^{u'}$ and it gives the maximum number of processed events that can be expected in any interval of time.

$$\alpha^{u'}(t) = \min_{0 \leq u \leq t} \left\{ \max_{v \geq 0} \left\{ \alpha^u(u+v) - \beta^l(v) \right\} + \beta^u(t-u), \beta^u(t) \right\} \tag{3.7}$$

This processed event stream can be used to handle data dependencies among task as illustrated in section 3.4.

3.2.2 Task models

A task model depicted as a task graph giving the precedence relationships between the tasks in the group is a more accurate representation of the way applications are generally modeled. In general any task model that can accurately calculate the demand bound function can be incorporated into the hierarchical scheduling framework. We illustrate the use of two different task models that also handle conditional branching in the hierarchical framework. Most of the task models, assume that processes are implemented as straight line code, and within the infinite loop are transformed into periodic or sporadic tasks or generalized multi-frame tasks. But in many event-driven real-time applications, the timing requirements are not representable as straight-line code. Instead, the action to be taken, depends on the current state of the systems and/or the values of certain external variables which can only be determined at runtime. Hence generic task models that handle conditional branching of code have been proposed. We shall briefly introduce two task models here and illustrate their use in

the hierarchical framework in section 3.3.

Recurring real-time task model

A recurring real-time task model [5] was introduced to permit the representation of conditional real-time code. A real-time task T is represented by a *task graph* $G(T)$ and a *period* $P(T)$. The task graph $G(T)$ is a directed acyclic graph (DAG) with a unique source vertex and a unique sink vertex. Each vertex in this DAG represents a subtask, and each edge represents a possible flow of control. Each vertex u is labeled by an ordered pair of real numbers $(e(u), d(u))$ with the following interpretation: every time subtask u is triggered, a job is generated with ready time equal to the triggering-time of the subtask, an execution requirement of $e(u)$ time units and a deadline of $d(u)$ time-units after the triggering time. Each edge (u, v) is labeled by a real number $p(u, v)$ denoting the minimum amount of time that must elapse after vertex u is triggered, before vertex v can be triggered.

Some additional notations include $E(T)$ that denotes the maximum possible cumulative execution requirement on any path from the source node to the sink node of the task graph $G(T)$, and $\rho_{ave}(T)$ that denotes the quantity $E(T)/P(T)$ called the *utilization* of the recurring real-time task T. An example recurring real-time task group is shown in Figure 3.9. The task graph on the left side of the figure shows a straight-line code with three subtasks and the task graph on the right side shows conditional code with four subtasks. The recurring real-time task is instantiated sporadically with $P(T)$ being the minimum inter-separation between successive instantiations. Thus a sporadic event model is considered for the recurring real-time task.

Definition 16: [Demand bound function of a task graph] The *Demand bound function* $T.dbf(t)$ is defined as the maximum cumulative execution

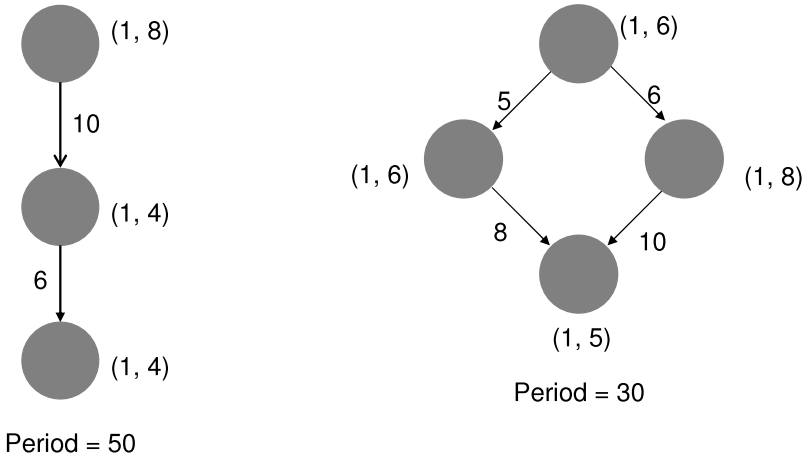

Fig. 3.9: Example recurring real-time tasks

requirement by jobs of a real-time task T that have both ready times and deadlines within any time interval of duration t.

Definition 17: [Request bound function of a task graph] The *request bound function T.rbf(t)* is defined as the maximum cumulative execution requirement by jobs of T that have their ready times within any time interval of duration t.

The algorithm to compute the *dbf* and *rbf* of a task group consisting of a recurring real-time tasks is presented in [5] along with the dynamic-priority and static priority feasibility tests. The schedulability conditions are presented in section 3.2.2.

Stream-based task model

The stream-based task model [8] based on the real-time calculus framework was proposed to accurately characterize streaming applications and is represented by a task graph triggered by an arbitrary event stream. The important difference between the recurring real-time task model and the stream-based task model is that, not all individual nodes of the task graph

have deadlines associated with them. Deadlines are associated with different paths of the graph, all of which start from the source node.

Formally, an acyclic, directed task graph $g = (V, A)$ consists of nodes $v \in V$ and edges $a = (v, w) \in A$. A node represents a task of the application and an edge represents a precedence relation between two tasks. There is a unique source node $v_0 \in V$ having no incoming edges. To each node v there are associated two quantities $c(v) \in \mathbb{R}^{\geq 0}$ and $d(v) \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, which denote the worst case resource demand and the deadline of the task associated to v, respectively.

The interpretation of the precedence relations and deadlines is defined by the dynamic model associated with a task graph. A task graph is instantiated by a timed event, for example at time t_0 and the release time of the source task v_0 is t_0. The execution follows a path (sequence of nodes) $(v_0, v_1, \cdots, v_n - 1)$ through g, where the nodes are connected by edges $(v_i, v_{i+1}) \in A$. The release time of a task v_{i+1} is the finishing time of task v_i. Each task v requires a maximal resource demand of $c(v)$, given in terms of the worst case execution time. In general the resource demand $c(v)$ can be expressed in terms of the number of required processor cycles or any other reasonable unit that describes the service of the resource (i.e., the processor on which all the tasks are executed). In addition each task has a relative deadline $d(v)$ with respect to the instantiation time t_0 of g. Therefore, its finishing time must be equal or smaller than $t_0 + d(v)$. All the events in an event stream need not be processed in the same way. Depending on the type of an event or its associated data, different paths through the task graph may be chosen. In addition, the processing of each event type might be associated with a different deadline constraint. If the deadline associated with a node is ∞ it implies that there are no deadline constraints on the

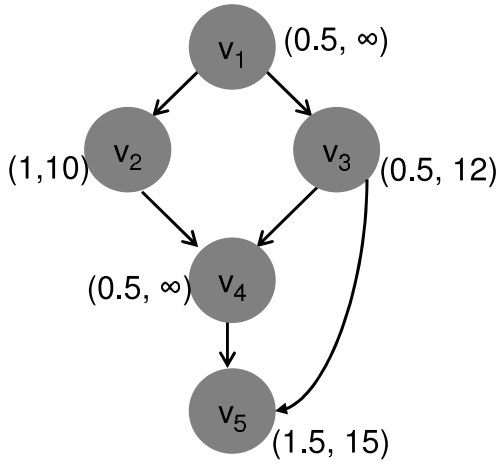

Fig. 3.10: An example task graph based on the stream-based task model

path starting from the source node to this node, although there might be deadlines associated with sub paths of this path. A sample task graph for $g_1 \in G$ is shown in Figure 3.10, where G is a set of task graphs.

Computation of dbf and rbf The *demand bound function* and *request bound function* of a task graph based on the stream-based task model are analogous to the definitions in case of the recurring real-time task model and are defined as in definitions 16 and 17.

Definition 18: The demand and resource bound sequences are defined as $\widetilde{dbf} = \langle\langle dbf_1, \delta_1\rangle, \langle dbf_2, \delta_2\rangle, \cdots\rangle$ and $\widetilde{rbf} = \langle\langle rbf_1, \delta_1\rangle, \langle rbf_2, \delta_2\rangle, \cdots\rangle$. The values of these functions can be determined using the operator $G_{\widetilde{dbf}}(t)$ defined as follows (the same definition holds for the resource bound sequence also):

$$G_{\widetilde{dbf}}(t) = max\{dbf_i | \delta_i \leq t\} \forall t \geq \delta_1 \tag{3.8}$$

and $G_{\widetilde{dbf}}(t) = 0, \ \forall t : 0 \leq t \leq \delta_1$

Every task graph is associated with an arrival function and a corresponding arrival sequence represented as an arrival curve, which gives the

triggering times of the task graph. The demand of a task graph over multiple instantiations is dependent on the arrival curve associated with the task graph.

Definition 19: [Composite demand bound function] The demand imposed by a task graph on multiple instantiations by a sequence of events whose arrival is bounded by an upper arrival curve within any time interval, is defined to be the *composite demand bound function* (dbf^C) of the task graph and is given by

$$dbf^C(t) = \sum_{i=1}^{m} (dbf_i - dbf_{i-1}) \cdot \alpha(t - \delta_i) \tag{3.9}$$

where $dbf_0 = 0$.

Definition 20: [Composite resource bound function] The load of a task graph in terms of its maximum resource requirement on multiple instantiations triggered by an arrival sequence bounded by an upper arrival curve within any time interval is defined as the *composite resource bound function* (rbf^C) of the task graph and is given by

$$rbf^C(t) = \sum_{i=1}^{m} (rbf_i - rbf_{i-1}) \cdot \alpha(t - \delta_i) \tag{3.10}$$

where $rbf_0 = 0$.

This task model is more pragmatic as it can accommodate an arbitrary event triggering sequence in the computation of the composite *dbf* and *rbf*.

Schedulability conditions

A necessary and sufficient condition for a system composed of tasks with cumulative demand given by $dbf(t)$, to be dynamic-priority feasible over a

dedicated resource is that the dbf over any interval length does not exceed the interval-length.

$$\forall t \geq 0 \sum_{T \in \Gamma} T \cdot dbf(t) \leq t \tag{3.11}$$

The schedulability conditions for recurring real-time task model and the stream-based task model on partitioned resource, with fixed-priority and dynamic priority scheduling policies are presented below.

Recurring real-time task model In case of partitioned resources that we consider in hierarchical scheduling frameworks, if we have $\beta^l(t)$ to give the supply bound of the resource partition, then the feasibility condition of Eq. (3.11) becomes

$$\forall t \geq 0 \sum_{T \in \Gamma} T \cdot dbf(t) \leq \beta^l(t) \tag{3.12}$$

Since it is infeasible to verify this condition for all $t \geq 0$, a constant lmt is derived, that is equal to

$$lmt = \frac{\sum_{T \in \Gamma} 2 \cdot E(T)}{1 - \sum_{T \in \Gamma} \rho_{ave}(T)} \tag{3.13}$$

where $E(T)$ is the maximum execution requirement of the task graph T and $\rho_{ave}(T) = E(T)/P(T)$. The schedulability condition holds for all $t \geq 0$, if and only if it holds for all $t < lmt$ where lmt is computed as shown above.

Static priority feasibility analysis on a partitioned resource can similarly be performed by testing the condition:

$$\forall t, \quad \sup_{0 \leq t' \leq t} \{\beta^l(t') - \sum_{T_j \in (\Gamma \backslash \{T_i\})} T_j \cdot rbf(t'\} \geq T_i \cdot dbf(t) \tag{3.14}$$

where $\Gamma \backslash \{T_i\}$ refers to the set of tasks with priority higher than T_i. The assignment of feasible priorities to task graphs is described in [5].

Stream-based task model For static priority scheduling, the task graphs are ordered in ascending priority and the resource (or the processor) delivers at

least $\beta(\Delta)$ resource units within any time interval of length δ. For a group of task graphs $G = \{g_1, g_2, \cdots g_k\}$ and for every $g_k \in G$, g_k has higher priority than g_{k+1}, if the schedulability condition, given by

$$dbf^C(t) \leq \sup_{0 \leq \lambda \leq t} \left\{ \beta(\lambda) - \sum_{j=1}^{k-1} rbf_j^C(\lambda) \right\} \tag{3.15}$$

holds for all graphs g_k and for all $t > 0$, then the set of task graphs is schedulable under fixed priority scheme. Intuitively, the right hand side represents the remaining resource capability after the maximum resource load due to all higher priority tasks has been taken into account. The left hand side represents the composite demand by the tasks of graph g_k.

Similarly, a set of task graphs is schedulable under preemptive dynamic priority scheduling, if and only if

$$\sum_{k=1}^{N} dbf_k^C(t) \leq \beta(t) \tag{3.16}$$

holds for all $t \geq 0$. Here too, the right hand side represents the accumulated resource demand of all task graphs g_k and the left hand side denotes the available resource capacity. While using EDF scheduling, the relative deadlines associated with the nodes of a graph might have to be adjusted (from ∞) in such a way that they are still compatible with the demand bound function. The new relative deadline of a node v_i can be assigned as the minimum of its specified deadline $d(v_i)$ and deadlines of all its direct and indirect successors in the task graph. In the case of the graph shown in Figure 3.10, the new deadlines are $d(v_1) = 10$ and $d(v_4) = 15$. The nodes of this graph can now be scheduled with these deadlines.

Example 8 illustrates scheduling recurring real-time tasks and stream-based tasks on partitioned resources.

3.3 Hierarchical scheduling framework

With sufficient background about the resource model and task models seen in the previous sections, the hierarchical scheduling framework can now be constructed. Fixing the physical resource at the system-level and the global scheduler, and given the workload groups and their local schedulers, the hierarchy can be constructed, working from both the root and the leaf levels, i.e. from the global scheduler working downwards and from the task group level working upwards.

At every level the partitioning of resources is done based on the scheduling policy enforced on that partition. For instance consider a TDMA based scheduler at the system-level. If there are four workload groups, then the demand imposed by each of these is determined, and the weights w_1, w_2, w_3 and w_4 have to be arrived at such that the task groups are schedulable under the resulting partitions.

The partitioning and partition scheduling schemes used in the bounded-delay model could be combined with the lower bounds from the real-time calculus model. Thus if we have a physical resource to be partitioned using some scheduling policy, the partitions could be parameterized as bounded-delay models (a_i, Δ_i). The schedulability of n child partitions parameterized as $(a_i, \Delta_i)_{i=1}^{n}$ within the parent partition (a, Δ), could be verified using the following conditions:

$$\sum_{i=1}^{n} a_i \leq a \quad \text{and} \quad \Delta_i > \Delta, \quad \forall i \tag{3.17}$$

For each partition the lower bound on the service would be represented by the β_i^l based on the real-time calculus framework. This resource supply bound could be used in the determining the schedulability of the workload scheduled within the partition.

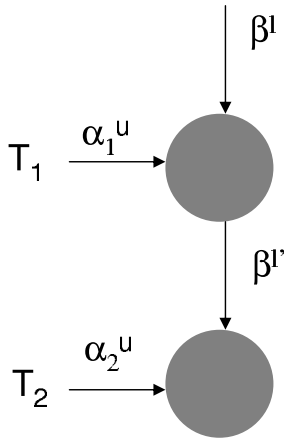

Fig. 3.11: Partitioning resources based on the scheduling policy at the virtual resource-level

Example 7: *Consider a dedicated physical resource that has to be partitioned into two, based on a TDMA scheduling policy with period P. The weights associated with the two partitions are w_1 and w_2 respectively with $w_1 + w_2 \leq 1$. The scheduler works as follows: time is divided into periods of length P and within any period, $w_1 \times P$ consecutive units of the processor's time allocated to partition 1 and $w_2 \times P$ consecutive units are allocated to partition 2. Partition 1 could be parameterized as (a_1, Δ_1) where $a_1 = w_1/(w_1 + w_2)$ and $\Delta_1 = P(1 - a_1)$ and partition 2 would be (a_2, Δ_2), where $a_2 = w_2/(w_1 + w_2)$ and $\Delta_2 = P(1 - a_2)$. So the parent partition is characterized by (a, Δ), where $a = 1$ and $\Delta = 0$, as we consider a dedicated resource at the top-level. The child partitions are schedulable on the parent if the conditions in Eq. (3.17) hold.*

In contrast to partitioning based on the scheduling policy only, there are cases when partitioning depends on the demand imposed by the workloads scheduled. In the case of a FP/RMS scheduler at the resource-level, partitioning primarily depends on the workloads it schedules. For instance, lets

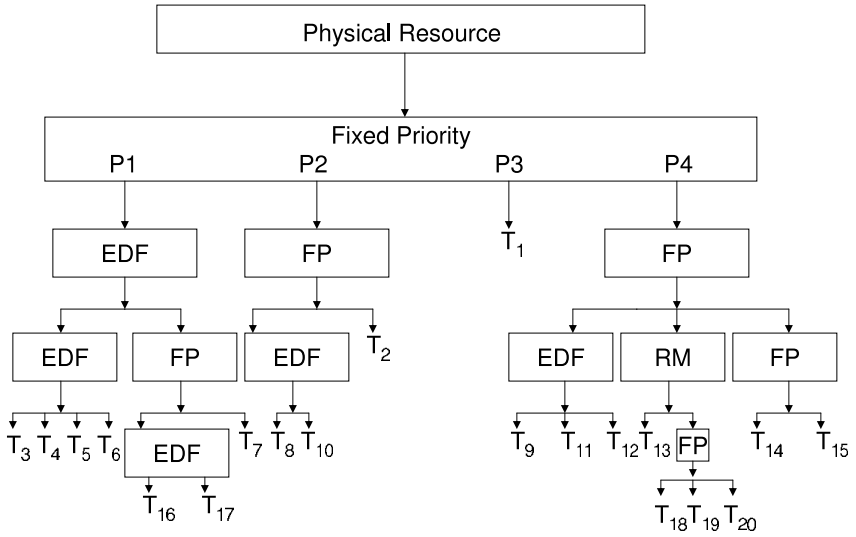

Fig. 3.12: Hierarchical scheduling framework

say the physical resource has to be partitioned into two virtual resources scheduling one task group each, and the virtual resources are scheduled using FP, with the first partition having higher priority than the second partition, then the partitioning is done as shown in Figure 3.11. As seen in the figure the resource supply to the first partition is β_1 and it schedules some taskgroup T_1 with the event triggering sequence bounded by the upper arrival curve α_1^u. The computation capacity left in first partition after processing T_1 is calculated as $\beta^{l'}$ and is used to scheduled task group T_2 triggered by α_2^u. It is evident that the remaining computation capacity is dependent on the demand of T_1. The remaining computation time is calculated as in Eq. (3.18).

$$\beta^{l'}(t) = sup_{0 \leq \lambda \leq t}\{\beta^l(\lambda) - dbf_{T_1}(\lambda)\} \tag{3.18}$$

A structural overview of an example system with hierarchical schedulers is shown in Figure 3.12. At the top level is a physical resource which uses

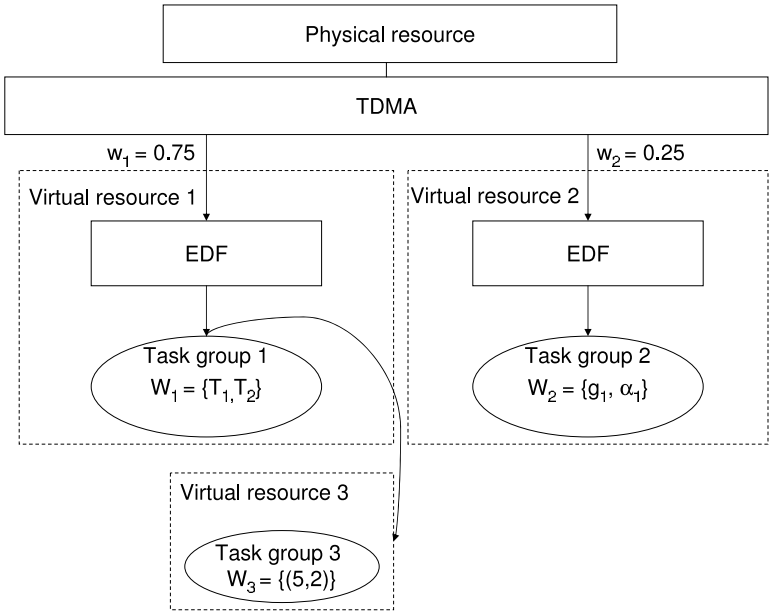

Fig. 3.13: Hierarchical scheduling framework - An example

a Fixed-Priority (FP) scheduler as the global scheduler. There are tasks T_1 to T_{20} scheduled at the various levels of the hierarchy. The tasks could be modeled as a mix of independent periodic tasks, recurring real-time tasks and tasks based on the stream-based task model, i.e. any task model that is amenable to accurate computation of the dbf. At almost every level of the hierarchy it is observed that there is a mix of tasks and resource partitions managed by local schedulers.

The system-level FP scheduler makes four partitions of the system resource and the first and second partitions are managed by EDF and FP schedulers respectively. The third partition is used to schedule task T_1, while the fourth partition is further handled by a fixed priority (FP) scheduler.

Since the FP scheduler determines the partitions based on the demand of the workloads it schedules, the system should be viewed from the leaf-level

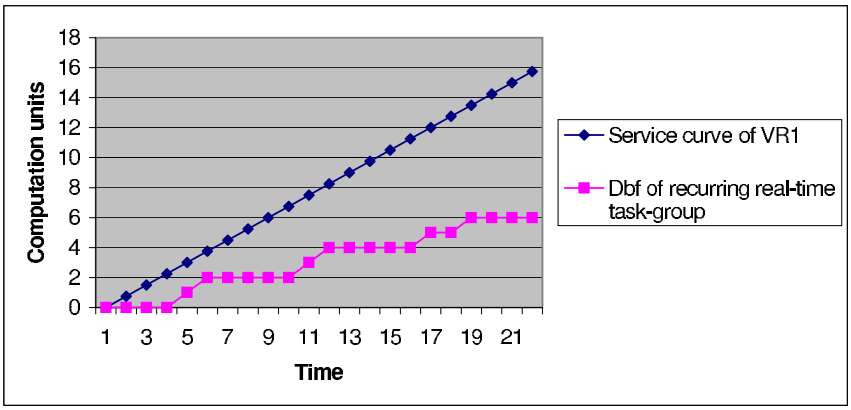

Fig. 3.14: Schedulability of W_1 under VR_1

to calculate the cumulative demand. For the first partition the demand comes from: an EDF scheduler for task group $\{T_3, T_4, T_5, T_6\}$, and a FP scheduler with EDF for task group $\{T_{16}, T_{17}\}$ with higher priority and task T_7 with lower priority. The cumulative demand imposed by this entire group denoted by dbf_1 forms the demand for the first partition. If the schedulability condition given by $dbf_1(t) \leq \beta^l(t)$ holds for all t, where $\beta^l(t)$, is the lower bound on the total resource capacity of the system, then the unused computation capacity can be determined using Eq. (3.18).

The computation capacity left over from the first partition denoted by $\beta^{l'}$ is passed over to the second partition which is managed by a FP scheduler. The demand on the second partition denoted by $dbf_2(t)$ is from the higher priority EDF scheduler for task group $\{T_8, T_{10}\}$, and lower priority task T_2. The computation capacity left over from the second partition denoted by $\beta^{l''}$ is then used to schedule task $T1$, and after that the remaining computation capacity denoted by $\beta^{l'''}$ is used for the fourth partition.

The advantage of computing the unused computation capacity can also be used to dynamically manage resources as evident in the following exam-

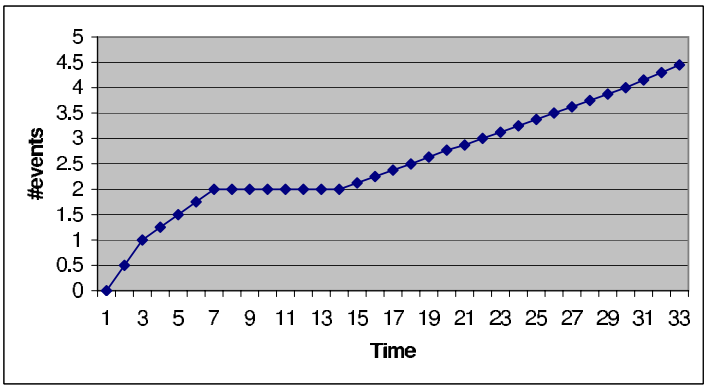

Fig. 3.15: A continuous arrival function defined using piecewise linear segments, with three pieces starting at t = 0, 2, 6, 13 with slopes 30, 15, 0 and 5 respectively, triggering the arrival of the stream-based task model in Figure 3.10

ple.

Example 8: *Consider the resource partitions* VR_1 *and* VR_2 *as in Figure 3.13. Suppose the top-level TDMA scheduler assigns weights* $w_1 = 0.75$ *and* $w_2 = 0.25$ *in a time period* P *that is infinitesimally small. We are given* $W_1 = \{T_1, T_2\}$ *and* $W_2 = \{(G_1, \alpha_1)\}$*, where* T_1 *and* T_2 *are two recurring real-time tasks represented by the task graphs in Figure 3.9 and* G_1 *is the stream-based task graph in Figure 3.10 triggered by the arrival function* α_1 *given in the Figure 3.15, and all the execution requirements and deadlines are assumed to be specified in time units. We intend to schedule* W_1 *on* VR_1 *and* W_2 *on* VR_2*. The dbf of the workloads are calculated based on algorithms for the respective task models. The demand bound functions and the supply bound functions for partitions* VR_1 *and* VR_2 *are plotted in Figures 3.14 and 3.16. As evident from the figures,* VR_1 *easily schedules the task group with some scope to reclaim the unused computation capacity, but* VR_2 *is just enough to schedule the task group* W_2*. The unused computation capacity*

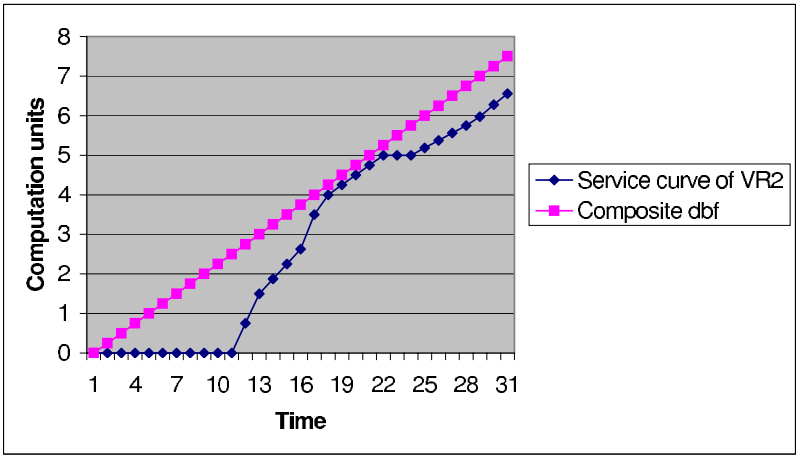

Fig. 3.16: Schedulability of W_2 under VR_2

from VR_1 can be reclaimed as in Eq. (3.18) to form partition VR_3 and a task group W_3 consisting of a single periodic task $W_3 = \{(5, 2)\}$ can be scheduled on that reclaimed resource as shown in Figure 3.17.

The important difference between Mok and Feng's hierarchical virtual resource model and our framework is that it is possible to calculate the computation capacity that remains of the partition after processing the workload scheduled under that partition. The resource characterization using service curves gives us this flexibility while the bounded-delay resource model being too generalized in the way resource is parameterized does not have this advantage. If dbf_W is the demand imposed by workload W on the resource with minimum supply of β^l, then the remaining computation capacity $\beta^{l\prime}$ is calculated by Eq. (3.18). This computation capacity could be passed on to any other resource partition or could just be used to schedule some workload within the same partition. Thus there could be workloads and partitions scheduled on the same level. While the real-time virtual resource framework, requires every task group to be scheduled on a partition cre-

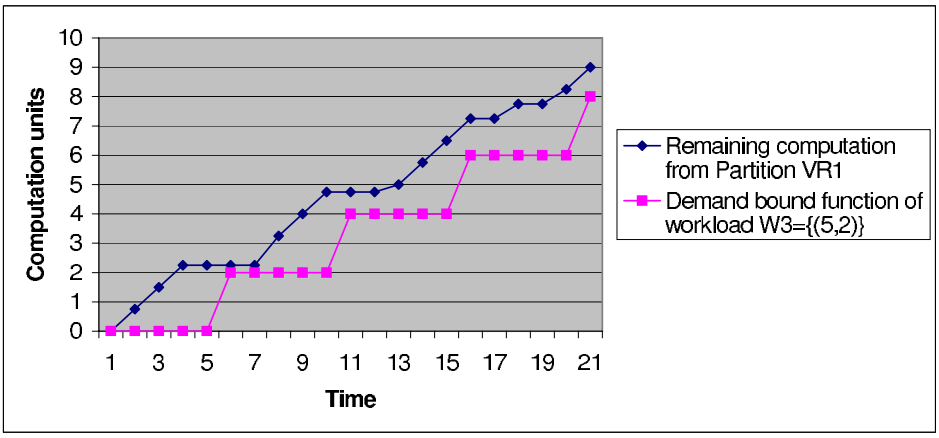

Fig. 3.17: Schedulability of W_3 under reclaimed unused computation capacity from VR_1

ated by abstracting its demand, our framework allows task-groups to be scheduled on the same level as partitions.

In general if there are n partitions of a resource β^l with task groups each with demand dbf_{W_i} scheduled on them, then the remaining computation capacity of the parent partition could be computed as

$$\beta^{l'}(t) = sup_{0 \leq \lambda \leq t}\{\beta^l(\lambda) - \sum_{i=1}^{n} dbf_{W_i}(\lambda)\} \tag{3.19}$$

This is in contrast to the real-time virtual resource framework, where the schedulability of partitions is done by considering the resource allocations to the child scheduling models, as the workload of the parent model. In this framework, we would like to consider the cumulative workload demand imposed on the child scheduling models to be the workload demand of the parent scheduling model in order to calculate the unused computation capacity. In this manner we avoid over-provisioning of resources.

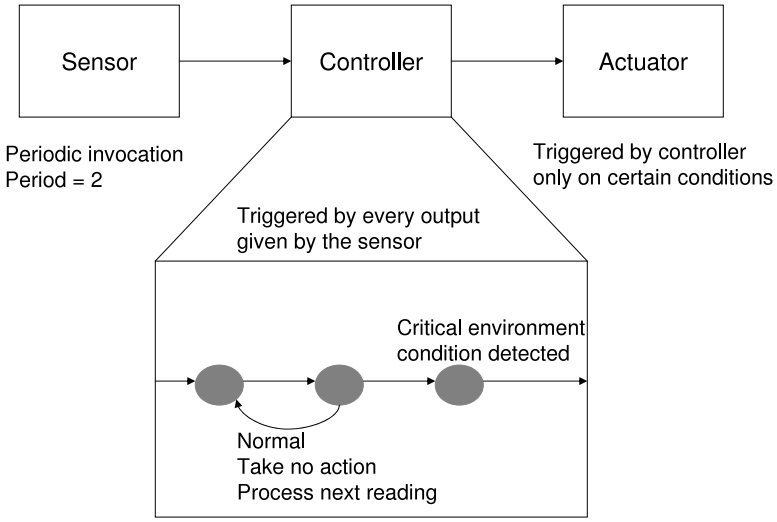

Fig. 3.18: Model of a simple real-time-system with a controller reading from a sensor and driving an actuator

3.4 Handling data dependencies

Handling data dependencies between tasks in a task group where the entire task group has a specific sampling rate is straightforward. The layout of the tasks will be in the form of a task precedence graph either based on the stream-based task model with a triggering arrival function or a recurring real-time task model with the invocation period specified. The problem is more involved if we have task groups with different triggering sequences involved in a precedence relationship.

A typical scenario is in the case of a hard real-time system that has a sensor component that takes inputs from the environment and supplies it to the controller and an actuator component that is driven by the controller to perform some action depending on the inputs received from the sensor. The model of such a system is depicted in Figure 3.18. Typically the sensor and actuator components are not always sampled at the same rate. The sensor might be sampled at a very high frequency and the controller might

not signal the actuator on all inputs from the sensor, but only on selective inputs that need to be acted upon.

Consider an air cooler-heater system, which has a sensor continuously sampling the temperature of the room and supplying it to the controller. The controller will do nothing if the temperature is within the threshold range (typically having an upper and lower threshold). If the controller finds that the temperature has dropped below the lower threshold then the heater component is turned on. On the other hand, if the controller finds that the temperature has gone beyond the upper threshold, the cooler component is turned on. Thus the actuator is activated at a different rate, while the sensor is activated at a much higher frequency. Whenever the controller reads inputs from the sensor and acts on it, the sensor might have had a subsequent invocation and care should be taken to ensure that the previous values are not erased. This ensures data-integrity. If the controller is guaranteed to produce outputs in the order corresponding to the reading of the sensor, that ensures determinism. The relationship between the controller and actuator is simply represented as a precedence relation. But the relationship between the sensor and the controller is more complicated. There has to be some way to ensure protected data-transfer from sensor to the controller in order to assure data-integrity and determinism.

To preserve data integrity, it would be desirable to have the producer (sensor) task triggered periodically and the consumer (controller) task triggered as and when the producer completes execution. The completion of a subsequent invocation of the sensor would trigger the controller with the new inputs, but that will be put in the ready queue, until the current invocation completes execution. This would ensure both data integrity and determinism. Thus the controller is triggered by output event model of the

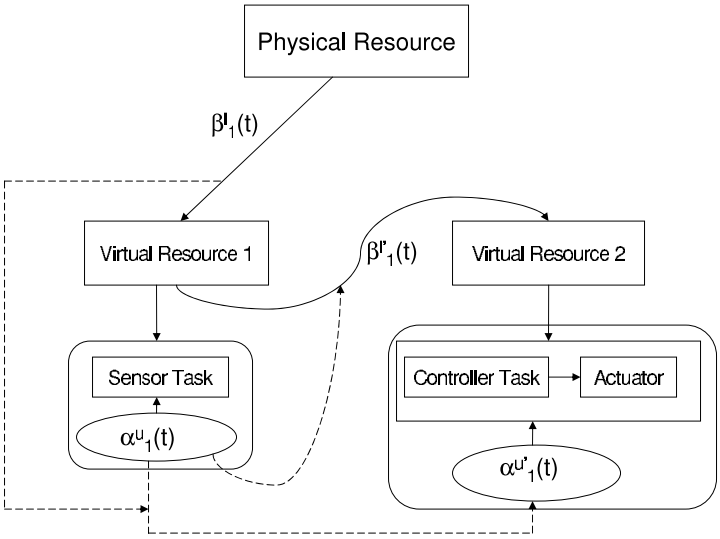

Fig. 3.19: Handling data dependencies among task groups using Real-time calculus

sensor component and the output model would not conform to any standard model as the task groups are executing on partitioned resources. The real-time calculus framework lends itself naturally these scenarios. The processed event stream can be calculated from the resource allocated to the producer task and the triggering sequence of the producer task. The processed event stream is denoted by α' and the upper and lower bounds of the processed event stream are $\alpha^{u'}$ and $\alpha^{l'}$ respectively. Informally, these denote the maximum and minimum number of output events in any time interval. Since we are only interested in the upper bound the Eq. (3.20) can be used to calculate the upper bound on the processed stream.

$$\begin{aligned}\alpha^{u'}(t) = min\{sup_{\lambda \geq 0}\{inf_{0 \leq \mu < \lambda + \mu}\{\alpha^u(\mu) + \\ \beta^u(\lambda + t - \mu)\} - \beta^l(\lambda)\}, \beta^u(t)\}\end{aligned} \tag{3.20}$$

Using this as the triggering sequence for the controller task in our example, the demand imposed by the consumer task can be determined. The schedulability of the task group consisting of the controller and the actuator tasks is verified with respect to the partition it is scheduled on.

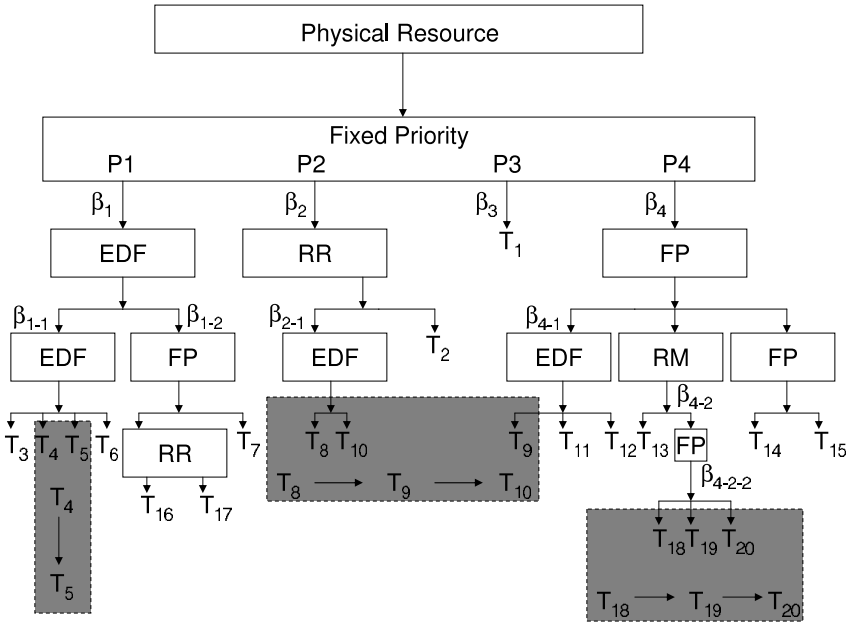

Fig. 3.20: An example of complex task dependencies

This is similar to the LET semantics in [19], in the sense that we logically extend the release time of the consumer task to coincide with the termination of the producer task. The difference here is that we can consider any arbitrary triggering sequence that need not conform to a standard event model. In addition we also explicitly specify how to derive the output event sequence. Congruent to the LET semantics, we have successfully removed the dependency between the consumer and producer task and they can be scheduled as independent tasks now either on the same partition or on different partitions.

If the sensor and the controller reside on different partitions and are scheduled by fixed priority scheduling, then the remaining processing capacity left over after processing the sensor task is modeled as the partition scheduling the controller task. The remaining computation could be computed as in Eq. (3.18).

The physical resource that schedules this kind of an application would be logically structured as shown in Figure 3.19. The physical resource would be partitioned into two virtual resources. The resource allocation to virtual resource 1 denoted by β_1^l, would virtually be the entire resource allocation of its parent. The demand imposed by the sensor task is in turn determined by the event sequence denoted by α_1^u. The resource allocated to resource partition 2 would be the computation capacity left over after processing the sensor task, i.e $\beta_1^{l'}$. The demand imposed by the task group containing the controller task and the actuator task is determined by the processed event sequence that is derived from β_1^l and α_1^u. The dotted lines in the figure indicate what values are used in the computation of parameters for the second partition. This is an example of inter-group data dependency scheduled using a static priority scheduler.

Intergroup data dependencies The example hierarchical scheduling framework shown in Figure 3.20 illustrates three types of data dependencies between tasks. The following is an example of intergroup data dependency amongst task groups on different partitions scheduled using EDF policy.

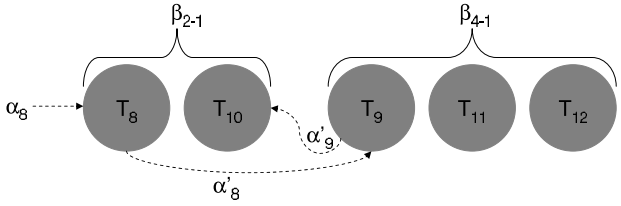

Fig. 3.21: Intergroup data dependency

Example 9: *Task T_{10} is data dependent on T_9 which in turn is data dependent on task T_8. While tasks T_8 and T_{10} are scheduled on one partition using resource β_{2-1}, task T_9 is scheduled on a different partition using resource β_{4-1} that is also shared by tasks T_{11} and T_{12}. The arrival functions*

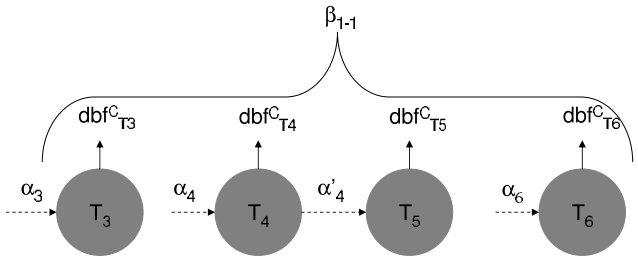

Fig. 3.22: Intragroup data dependency - scheduling with EDF

of each of these tasks and their grouping their under respective partitions is shown in Figure 3.21. All tasks are assumed to be represented as stream-based task models. Task T_8 *is triggered by an arrival function denoted by* α_8 *and the processed stream from* T_8 *denoted by* α'_8 *is the arrival function that triggers* T_9*. The processed stream from* T_9 *denoted by* α'_9 *is the arrival function for* T_{10}*. The composite dbf as in Definition 19 of tasks* T_8 *and* T_{10} *denoted by* $dbf_8^C(t)$ *and* $dbf_{10}^C(t)$ *are determined using* α_8 *and* α'_9 *respectively, and the schedulability condition is given by* $(dbf_8^C(t) + dbf_{10}^C) \leq \beta_{2-1}(t),\ \forall t$*. Similarly the composite dbf of tasks* T_9*,* T_{11} *and* T_{12} *denoted by* $dbf_9^C(t)$*,* $dbf_{11}^C(t)$ *and* $dbf_{12}^C(t)$ *are determined using* α'_8*,* α_{11} *and* α_{12} *respectively and the schedulability condition in this case is*

$$(dbf_9^C(t) + dbf_{11}^C(t) + dbf_{12}^C(t)) \leq \beta_{4-1}(t),\ \forall t$$

Intra group data dependencies The hierarchical scheduling framework in Figure 3.20 also illustrates data dependencies between task groups within the same partition, intra group data dependencies. Scheduling such tasks under dynamic and static priority schemes is illustrated below.

Example 10: *Tasks* T_4 *and* T_5 *are involved in a precedence relationship such that task* T_5 *is data dependent on* T_4*. These two tasks along with* T_3 *and* T_6 *are scheduled on an EDF scheduler with resource* β_{1-1}*. As shown in Figure*

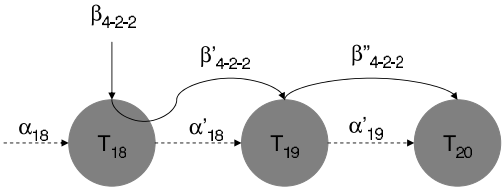

Fig. 3.23: Intragroup data dependency - scheduling with Fixed Priority scheduler

3.22, the composite demand bound function of each of these tasks denoted by $dbf_3^C(t)$, $dbf_4^C(t)$, $dbf_5^C(t)$ *and* $dbf_6^C(t)$ *are computed using* α_3, α_4, α_4' *and* α_6 *respectively, and the schedulability condition is given by*

$$\sum_{i=3}^{6} dbf_i^C \leq \beta_{1-1}, \ \ ,\forall t$$

Example 11: *Tasks* T_{18}, T_{19} *and* T_{20} *are involved in a precedence relationship such that task* T_{20} *is data dependent on* T_{19} *which in turn is data dependent on* T_{20}. *These three tasks are scheduled on a fixed priority scheduler which has lower priority than task* T_{13}. *As shown in Figure 3.20 assuming* β_{4-2} *to be the resource allocated to the RM scheduler that schedules* T_{13} *and the FP scheduler,* β_{4-2-2} *will be the processing capacity left over from* β_{4-2} *after scheduling* T_{13}, *and is passed on to the FP scheduler. The composite demand bound functions of tasks* T_{18}, T_{19} *and* T_{20} *denoted by* dbf_{18}^C, dbf_{19}^C *and* dbf_{20}^C *are determined using the arrival functions* α_{18}, α_{18}' *and* α_{19}'. *The composite resource bound function as in Definition 20 of these tasks would be denoted as* rbf_{18}^C, rbf_{19}^C *and* rbf_{20}^C. *The schedulability is determined by using the following equations:*

$$dbf_{18}^C(t) \leq \beta_{4-2-2}(t)$$

$$dbf_{19}^C(t) \leq \sup_{0 \leq t' \leq t} \{\beta_{4-2-2}(t') - rbf_{18}^C(t))\}$$

$$dbf_{20}^C(t) \leq \sup_{0 \leq t' \leq t} \{\beta_{4-2-2}(t') - (rbf_{18}^C(t) + rbf_{19}^C(t))\}$$

The actual resource flow to each of the tasks is as shown in Figure 3.23.

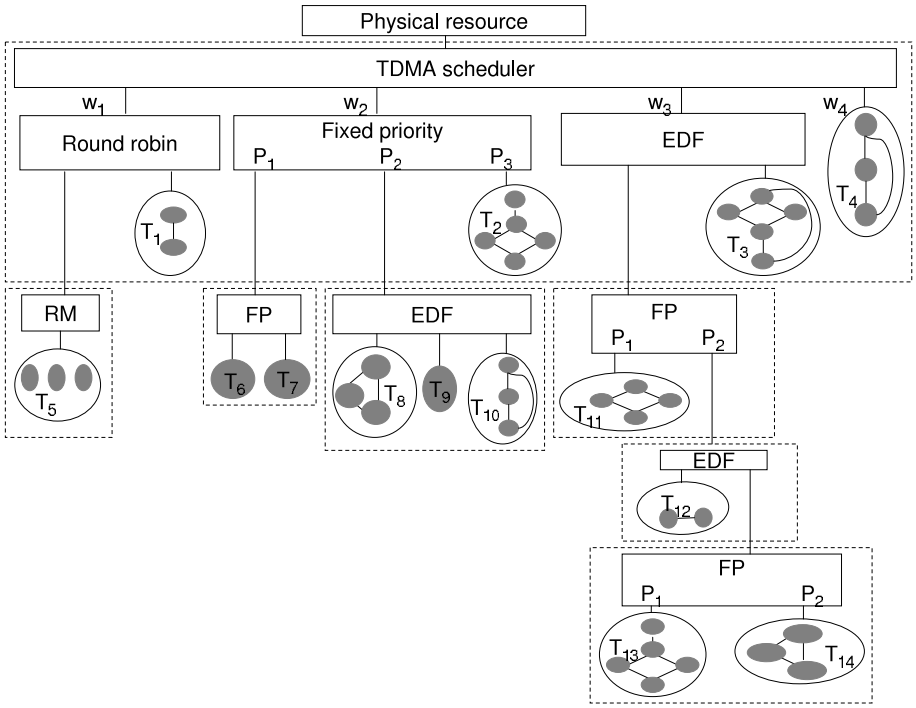

Fig. 3.24: Abstract model of a complex system based on the hierarchical scheduling framework

3.5 Illustrative case study

The case study real-time system consists of fourteen real-time tasks T_1 to T_{14}, sharing a processor under a scheduling hierarchy as shown in Figure 3.24. The scheduling hierarchy uses a mix of TDMA, Fixed priority (FP), RM, EDF and RR scheduling. The task set contains a mix of task graphs based on the recurring real-time task model and stream-based task model and independent tasks each associated with their own arrival curves. The properties of the tasks are given in Table 3.1. The task types are indicated as follows: recurring real-time tasks denoted as $R-RT$, stream-based tasks denoted by SB, independent periodic tasks given by PI, and single independent tasks associated with arrival curves given by SI. For the stream-based (SB) and recurring real-time ($R-RT$) tasks that are given by task graphs,

relative deadlines are assumed to be indicated in the task graphs and hence that column is left blank in the table.

Load	Type	Arrival	*Scheduling Algorithm*	*Exec. Demand*	*Relative Deadline*
T_1	$R-RT$	P_1	RR	dbf_{T_1}	–
T_2	SB	α_2	FP	dbf_{T_2}	–
T_3	SB	α_3	EDF	dbf_{T_3}	–
T_4	$R-RT$	P_4	TDMA	dbf_{T_4}	–
T_5	PI T_{51} T_{52} T_{53}	p_{51} p_{52} p_{53}	RM	e_{51} e_{52} e_{53}	p_{51} p_{52} p_{53}
T_6	SI	α_6	RM	e_6	d_6
T_7	SI	α_7	RM	e_7	d_7
T_8	SB	α_8	EDF	dbf_{T_8}	–
T_9	SI	α_9	EDF	e_9	d_9
T_{10}	SB	α_{10}	EDF	$dbf_{T_{10}}$	–
T_{11}	$R-RT$	P_{11}	FP	$dbf_{T_{11}}$	–
T_{12}	SB	α_{12}	EDF	$dbf_{T_{12}}$	–
T_{13}	SB	α_{13}	FP	$dbf_{T_{13}}$	–
T_{14}	SB	α_{14}	FP	$dbf_{T_{14}}$	–

Tab. 3.1: Temporal properties of tasks in the case-study real-time system of Figure 3.24

At the physical resource level, a TDMA-based scheduler partitions the resource into four with weights w_1, w_2, w_3 and w_4 respectively. Within any time period P, the processor time is divided into four partitions proportion-

ate to the weights assigned to each partition as explained in Example 7. For illustration purpose, let us assume the period to be infinitesimally small, and considering dedicated service by the processor to be $\beta^u(t) = \beta^l(t) = t$, the partitions would each have services represented as linear service curves β_1 to β_4, proportionate to the weights assigned to each partition.

The first partition having weight w_1 is utilized by a round robin scheduler scheduling a task T_1 and task-group T_5 scheduled using RM scheduler. The RR scheduler partitions the resource supply equally between the RM scheduler and task T_1. Thus β_{1-1} and β_{1-2} are the two partitions that the RR scheduler creates. Task T_1 is a recurring real-time task with sporadic arrival and its demand is calculated to be dbf_{T_1} and the schedulability condition is

$$dbf_{T_1} \leq \beta_{1-2}$$

The task group T_5 has three periodic tasks T_{5-1}, T_{5-2} and T_{5-3}, in ascending order of their periodicity. The entire task group is given a resource allocation of β_{1-1}. The highest priority task T_{5-1} utilizes β_{1-1} and passes on the remaining capacity β'_{1-1} to the other tasks. Task T_{5-2} consumes β'_{1-1} and passes on the β''_{1-1} to the third task T_{5-3}. The demand bound function of every task in the group, denoted by $dbf(T_5, t, T_{5-1})$, $dbf(T_5, t, T_{5-2})$ and $dbf(T_5, t, T_{5-3})$ is computed by taking into account the interference from its higher priority tasks as given by Eq. (1.2). Along these lines, the demands of each of the tasks are computed to be

$$dbf(T_5, t, T_{5-3}) = e_{5-3} + \left\lceil \frac{t}{p_{5-2}} \right\rceil \cdot e_{5-2} + \left\lceil \frac{t}{p_{5-1}} \right\rceil \cdot e_{5-1}$$

$$dbf(T_5, t, T_{5-2}) = e_{5-2} + \left\lceil \frac{t}{p_{5-1}} \right\rceil \cdot e_{5-1}$$

$$dbf(T_5, t, T_{5-1}) = e_{5-1}$$

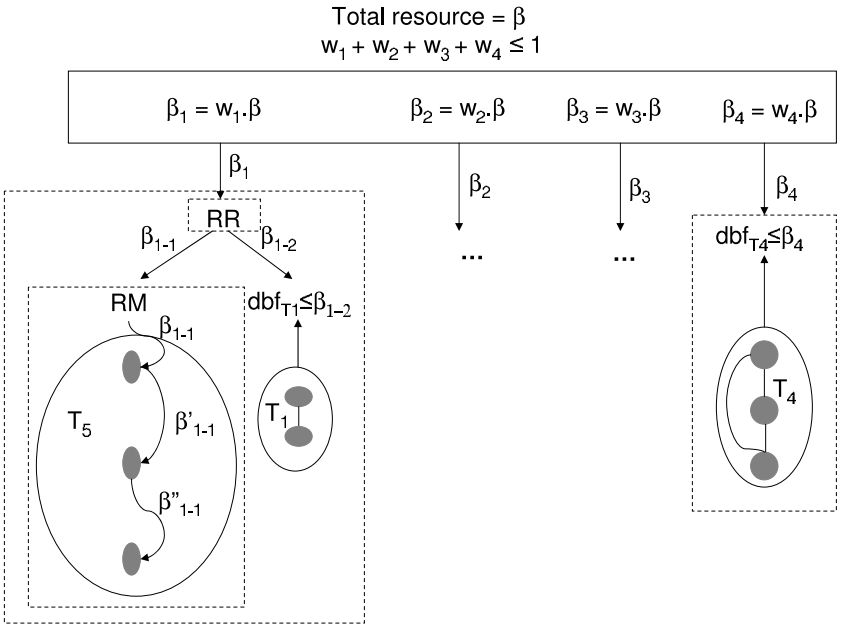

Fig. 3.25: Partitions 1 and 4 of level 1 of the system in Fig. 3.24

For every task T_i the schedulability condition given by Eq. (1.7) has to be verified.

The fourth partition of the TDMA scheduler, having weight w_4 is used to schedule a task T_4, which is a recurring real-time task having a demand dbf_{T_4}. The schedulability condition is directly given by

$$dbf_{T_4} \leq \beta_4$$

The resource flow and resource demand for partitions 1 and 4 of the TDMA scheduler of the hierarchical framework is shown in Figure 3.25. The flow of resources is given by downward arrows and the demand of the task groups is indicated by upward arrows.

The second partition of weight w_2 of the TDMA scheduler, uses a fixed priority scheduler and schedules three task groups: a FP scheduler scheduling T_6 and T_7 (T_6 having higher priority than T_7), an EDF scheduler with a task group consisting of T_8, T_9 and T_{10}, and a task T_2. The FP scheduler

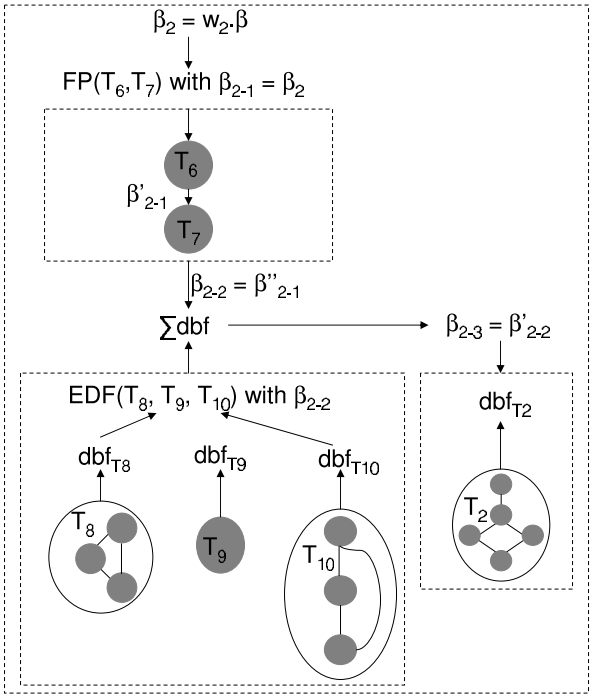

Fig. 3.26: Partition 2 of level 1 of the system in Fig. 3.24

has the highest priority, the EDF has a lesser priority than FP and higher priority than task T_2.

Tasks T_6 and T_7 are independent tasks with associated arrival curves α_6 and α_7. While T_6 has a bursty arrival pattern, T_7 is periodic with jitter and T_6 has higher priority than T_7. The execution requirements and deadlines of the tasks are given in the table. The demand bound function of T_6 and T_7 are given by

$$dbf_{T_6} = e_6 \cdot \alpha_6(t - d_6), \forall t > d_6$$

$$dbf_{T_7} = e_7 \cdot \alpha_7(t - d_7), \forall t > d_7$$

The resource allocation from β_2 to the FP scheduler is denoted by β_{2-1} and in this scenario they are equal. Task T_6 utilizes β_{2-1} and passes on β'_{2-1} to T_7.

The computation capacity remaining from the FP scheduler is denoted

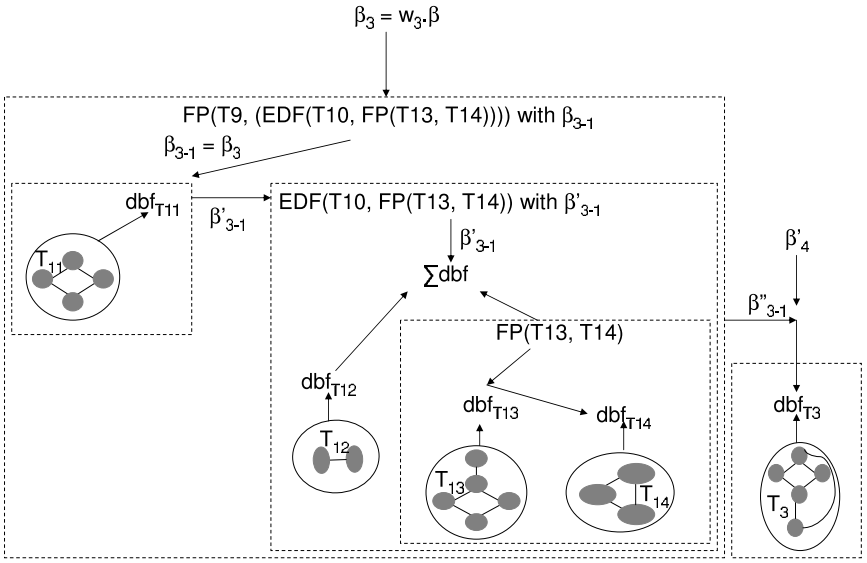

Fig. 3.27: Partition 3 of level 1 of the system in Fig. 3.24

by β''_{2-1} denoted by β_{2-2} to indicate the resource supply to the second partition at the second level of the hierarchy, and is given to the EDF scheduler. Tasks T_8 and T_{10} are based on the stream-based task model and have arrival curves α_8 and α_{10}. T_9 has a non-standard arrival pattern given by an arrival curve α_9 with piecewise linear segments. The demand imposed by T_9 is given by

$$dbf_{T_9}(t) = e_9 \cdot \alpha_9(t - d_9), \forall t > d_9$$

The demand bound functions of the other tasks are given by dbf_{T_8} and $dbf_{T_{10}}$ and computed as stated in section 3.2.2. Schedulability is checked by the inequality

$$dbf_{T_8} + dbf_{T_9} + dbf_{T_{10}} \leq \beta_{2-2}$$

The remaining computation capacity β'_{2-2} from the EDF scheduler indicated as β_{2-3} is given to the task T_2, whose demand dbf_{T_2} is computed based on its arrival curve α_2.

The resource allocation and the demand imposed by the task groups in

the second partition of the global TDMA scheduler is shown in Figure 3.26.

The third partition of the TDMA scheduler has a weight w_3 and is partitioned in a complex manner. It schedules two task groups: a fixed priority scheduler and a task T_3. The fixed priority scheduler schedules task T_{11} with higher priority and EDF scheduler with lower priority. The EDF scheduler in turn schedules task T_{12} another FP scheduler with tasks T_{13} and T_{14}. Task T_{11} has a demand $dbf_{T_{11}}$ and utilizes β_3 directly.

The remaining capacity β_3' is passed on as β_{3-2} to the EDF scheduler. Since we are dealing with and FP scheduler under the EDF scheduler, we now work from the task0-level. The cumulative demand of the tasks T_{13} and T_{14} under the FP scheduler is determined and $dbf_{T_{14}} + dbf_{T_{15}}$ is combined with $dbf_{T_{12}}$ under the EDF scheduling policy to determine the demand on the EDF scheduler. This cumulative demand should be met by β_{3-2} that comes from the FP scheduler.

Whatever remains of the capacity β_{3-2} is passed on to the task T_3. Task T_3 might also need more resources and the unused computation capacity from partition 4 might be used too. Although task T_3 is shown to be under the EDF scheduler, the resource allocation from partition 3 goes directly to the FP scheduler and the subsequent levels and the remaining computation capacity from partition 3 and partition 4 flow in for task T_3.

While having fixed the weights of the TDMA scheduler and working down, if it is found that some task or task group turns out to be infeasible to schedule, the weights could be re-adjusted and checked again. This could also be adjusted by reclaiming the left over computation capacity from some other partition to be used to schedule the infeasible task group. Thus having fixed the structure, the analysis involves adjusting the resource allocations to the various partitions such that the entire system turns out to be

schedulable.

3.5.1 Evaluation of the framework

The system model in the case-study clearly illustrates the flexibility of the hierarchical scheduling framework. The flexibility of this approach stems from the mix of various task models, scheduling policies and the ability to analyze the temporal properties of tasks and resource supplies.

- **Task models:** The framework accommodates a variety of task models with arbitrary event triggering patterns. The demand bound function of such tasks is computed with the temporal properties of the associated event model, the execution requirement and deadline. The framework also allows the use of complex task models with conditional branching code like the recurring real-time task model and the stream-based task model that closely characterize the way applications are generally modeled.

- **Data dependencies:** While the task models like stream-based task models and recurring real-time task models handle data dependencies amongst tasks that are triggered at the same rate, the ability to compute the temporal properties of the processed event stream aid us in dealing with task group dependencies with different triggering times. For instance the tasks T_6 and T_9 could have been in a precedence relationship and the arrival curve α_9 of T_9 could be the processed event stream of α_6 of task T_6.

- **Resource supply bound and utilization:** As proven in section 3.1.3, the lower service curve tightly bounds the resource supply. Using the tight bounds in analysis of the hierarchical scheduling framework

aims in maximizing utilization. The tighter supply bounds, together with the ability to reclaim the remaining computation capacity maximize the utilization of the resource. Supplying the unused capacity to a different partition, that is tightly scheduling a task group, gives a lot of flexibility to the placement of tasks in the hierarchical framework.

- **Schedulability analysis:** The schedulability analysis at different levels of the hierarchy involve using different techniques. As mentioned in the special note on arrival curves of section 3.2.2 either the virtual time scheduling introduced in [20], the partition scheduling technique introduced in [11] given by Eq. (3.17), or the task group scheduling techniques using demand bound functions dbf could be used. The flexibility to choose the technique most suitable at every level, depending on the parameters that are easily calculable renders the framework very powerful.

- **Scheduling policies:** In addition to the FP, RM and EDF scheduling policies that are usually used in the hierarchical scheduling frameworks, the use of TDMA and round-robin scheduler are introduced in our framework. The real-time calculus naturally lends itself to such scheduling policies as it draws its roots from network calculus theory.

4. CONCLUSION

We have presented a generalized approach to hierarchical scheduling based on real-time calculus. The scheduling framework also uses techniques established by the hierarchical real-time virtual resource framework. The simplicity of partition schedulability and parameterization of the real-time virtual resources based on the bounded-delay resource partition model have been combined with the tight supply bounds of the service curve representation of real-time calculus. In addition to the traditional scheduling policies like EDF and RM, we also illustrated the use of TDMA and RR scheduling policies.

We presented techniques to avoid over provisioning of resources by reclaiming remaining computation capacity left over after processing the workload. While abstracting the demands of the child partitions, it is advantageous to consider the demands of the workloads scheduled within the child partitions, to facilitate the reclamation of the unused computation capacity. In the hierarchy we allowed tasks and resource partitions to exist at the same level, as at the parent level, we anyway consider only the cumulative demand of all the children whether they happen to be partitions or tasks. This renders flexibility in the placement of tasks in the hierarchy. A single task with a specific demand bound function computed, can be placed under any partition that loosely schedules its workload with scope for reusing the remaining computation capacity, provided the cumulative demand after admitting the task is still met by the partition.

We presented the use of task models representable as task graphs, to directly handle data dependencies and precedence relationships among tasks. For task graphs that are entirely triggered at a particular rate, these task models suffice to accommodate the precedence relationships in the computation of demand imposed by the task graph. But for tasks that are sampled at different rates, we illustrated the computation of the temporal properties of tasks after they are processed, in the form of output event streams. Considering the bounds on this processed event stream as the arrival curve for the dependent task, enables the handling of data dependencies among tasks. This way, tasks involved in precedence relationships are decoupled with independent arrival curves and can be scheduled as independent tasks either on the same partition or different partitions ignoring the precedence relationships.

4.1 Future Work

In an effort to generalize the hierarchical scheduling framework further, it would be desirable to derive some utilization bounds permitting online admission test of task groups. Parallel to the approach in [29] if there is a way to derive a utilization bound, based on the temporal properties of tasks, a quick computation of the demand of the new task-group would be all that is necessary, and adding it to the pre-computed demand of all task-groups in the system, the new utilization could be checked against the pre-determined utilization bound.

Exact schedulability conditions for a wider range of scheduling algorithms could be derived. For instance the computation of the demand bound function of a task group under the Least slack-time first (LSF) or First-come first-served (FIFO) algorithms and the exact schedulability condition for a

task group under a given resource model would be beneficial when there is flexibility in the choice of scheduling policy. The response time of tasks under the various scheduling schemes could also be derived. There has been a lot of work in response time analysis of tasks under fixed priority scheduling of hierarchically scheduled systems. A similar analysis in the case of dynamic priority scheduling schemes would be very useful. This is especially beneficial in virtual-time scheduling schemes used for fractional resources.

BIBLIOGRAPHY

[1] Asynchronous support. In *Real-time Workshop User Guide*, pages 1–40. The MathWorks Inc., 2005.

[2] Models with multiple sample rates. In *Real-time Workshop User Guide*, pages 1–34. The MathWorks Inc., 2005.

[3] L. Abeni and G. Buttazzo. Integrating multi-media applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-time Systems Symposium*, pages 4–13, December 1998.

[4] Luis Almeida and Paulo Pedreiras. Scheduling with temporal partitions: response time analysis and server design. *ACM Conference on Embedded Software (EMSOFT)*, pages 95–103, 2004.

[5] Sanjoy K. Baruah. Dynamic and static priority scheduling of recurring real-time tasks. *Real-time Systems*, 24:93–128, 2003.

[6] B.Horowitz, T.A.Henzinger, and C.M.Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91:84–99, 2003.

[7] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system design. In *Proceedings of Design, Automation and Test in Europe (DATE)*, pages 190–195, March 2003.

[8] S. Chakraborty and L. Thiele. A new task model for streaming applications and its schedulability analysis. In *Proceedings of IEEE Design Automation and Test in Europe (DATE)*, March 2005.

[9] Robert Davis and Alan Burns. Hierarchical fixed priority pre-emptive scheduling. In *Proceedings of IEEE Real-time Systems Symposium (RTSS'05)*, December 2005.

[10] D. Epema and J. Jongh. Proportional-share scheduling in single-server and multiple-server computing systems. *ACM Sigmetrics Performance Evaluation Review*, 27:7–10, December 1999.

[11] X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS'02)*, pages 26–35, December 2002.

[12] Hermann Kopetz, Andreas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl, Christoph Senft, and Ralph Zainlinger. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, 9(1):25–40, 1989.

[13] G. Lipari and Sanjoy Baruah. A hierarchical extension to the constant bandwidth server framework. In *Proceedings of the IEEE International Real-Time Technology and Applications Symposium*, pages 26–35, May 2001.

[14] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS'03)*, July 2003.

[15] G. Lipari and E. Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1, April 2004.

[16] Giuseppe Lipari, Paolo Gai, Michael Trimarchi, Giacomo Guidi, and Paolo Ancilotti. A hierarchical framework for component-based real-time systems. *Lecture Notes in Computer Science (LNCS)*, 3054:209–216, 2004.

[17] Giuseppe Lipari, Gerardo Lamastra, and Luca Abeni. Task synchronization in reservation-based real-time systems. *IEEE Transactions on Computers*, 53:1591–1601, December 2004.

[18] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20:46–61, January 1973.

[19] Slobodan Matic and Thomas A. Henzinger. Trading end-to-end latency for compositionality. In *Proceedings of IEEE Real-time Systems Symposium (RTSS'05)*, 2005.

[20] A. Mok and D. Chen. Resource partition for real-time systems. In *Proceedings of Real-Time Technology and Application Symposium*, pages 75–84, 2001.

[21] A. K. Parekh and R. G. Galleger. A generalized processor sharing approach to flow control in integrated service networks: the single node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[22] J. Regehr and J. Stankovic. Hls : A framework for composing soft real-time schedulers. In *Proceedings of IEEE Real-time Systems Symposium (RTSS'01)*, pages 3–14, December 2001.

[23] S. Saewong, R. Rajkumar, J.P. Lehoczky, and M.H. Klein. Analysis of hierarchical fixed-priority scheduling. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS'02)*, June 2002.

[24] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proceedings of IEEE Real-time Systems Symposium (RTSS'03)*, pages 2–14, December 2003.

[25] I. Shin and I. Lee. Compositional real-time scheduling framework. In *Proceedings of IEEE Real-time Systems Symposium (RTSS'04)*, pages 2–14, December 2004.

[26] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings of the 15th IEEE Real-Time System Symposium (RTSS 94)*, December 1994.

[27] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of IEEE International Conference on Circuits and Systems (ISCAS)*, volume 4, pages 101–104, 2000.

[28] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. *To appear in ACM Conference on Embedded Software (EMSOFT)*, 2005.

[29] Jianjia Wu, Jyh-Charn Liu, and Wei Zhao. On schedulability bounds of static priority schedulers. In *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'05)*, pages 529–540, March 2005.