A Dynamic Slack Management Technique for Real-Time Distributed Embedded Systems

Subrata Acharya, *Member*, *IEEE*, and Rabi N. Mahapatra, *Senior Member*, *IEEE*

Abstract—This work presents a novel slack management technique, the *Service-Rate-Proportionate (SRP) Slack Distribution*, for real-time distributed embedded systems to reduce energy consumption. The proposed SRP-based Slack Distribution Technique has been considered with EDF and Rate-Based scheduling schemes that are most commonly used with embedded systems. A fault-tolerant mechanism has also been incorporated into the proposed technique in order to utilize the available dynamic slack to maintain checkpoints and provide for rollbacks on faults. Results show that, in comparison to contemporary techniques, the proposed SRP Slack Distribution Technique achieves about 29 percent more performance/overhead improvement benefits when validated with random and real-world benchmarks.

Index Terms—Real time, slack, periodic service rate, energy efficient, fault tolerance.

1 Introduction

By the turn of the century, embedded computing systems had proliferated in almost all areas of technology and applications. There has been phenomenal demand on small factor devices during the past decade. Their application spans stand-alone battery-operated devices (such as cellular phones, digital cameras, MP3 players, Personal Digital Assistances (PDAs), medical monitoring devices) to real-time distributed systems used in sensor networks, remote robotic clusters, avionics, and defense systems. Some of the well-known issues in the implementation of real-time distributed embedded systems are due to reduced power consumption and reliable data processing. These issues turn out to be significant challenges when the processing elements are heterogeneous and the distributed system handles time-varying workloads. The heterogeneity of processing elements enables a tighter bound on the flexibility of reschedules that are necessary for exploiting runtime variations than homogeneous processing elements. Although the consideration of time-varying workloads creates a realistic dynamic task input into the distributed system, it creates greater challenge for the system designers in terms of exploiting runtime slack, making reschedule decisions, and providing synchronization for reduced energy consumption. Furthermore, as reliability and dependability become important in such distributed embedded systems, there is also an important consideration to provide fault tolerance in the system. The incorporation of fault tolerance leads to an overhead on the use of slack for maintaining checkpoints and rollback in such distributed embedded systems.

Thus, the primary goal of today's distributed embedded system designers is to incorporate the above characteristics in the system model and provide energy-efficient solutions. In real-time system designs, Slack Management is increasingly applied to reduce power consumption and optimize the system with respect to its performance and time overheads. This Slack Management Technique exploits the *idle time* and *slack time* of the system schedule by frequency/voltage scaling of the processing elements in order to reduce energy consumption. The main challenge is to obtain and distribute the available slack in order to achieve the highest possible energy savings with minimum overhead. There has been a lot of attention toward the design of such an energy-efficient slack management technique, but most of these do not address time-varying inputs. Only a few that attempt to handle dynamic task inputs assume a homogeneous distributed embedded system. Also, fault tolerance has not been combined with slack management in such heterogeneous distributed embedded systems. Thus, the aim of the proposed research is to consider the design of such fault-tolerant heterogeneous distributed real-time embedded systems, which take dynamic task set inputs into account.

We propose a low-power dynamic task set input slack distribution technique, that is, the *Service-Rate-Proportionate (SRP) Slack Distribution Technique*, which fares better than contemporary techniques in providing for energy efficiency. Both the Dynamic and the Rate-Based scheduling schemes have been examined with the proposed technique. Furthermore, this work also demonstrates the impact of the proposed slack distribution technique with the class of rate-based sheduling schemes.

This paper has the following contributions:

- It introduces a dynamic slack management technique for heterogeneous distributed embedded systems to reduce power consumption.

- *S. Acharya is with the Department of Computer Science, University of Pittsburgh, 6150 Sennott Square, 210 S. Bouquet Street, Pittsburgh, PA 15232. E-mail: sacharya@cs.pitt.edu.*
- *R. Mahapatra is with the Department of Computer Science, Texas A&M University, College Station, TX 77843-3112. E-mail: rabi@cs.tamu.edu.*

Manuscript received 24 Aug. 2004; revised 24 May 2005; accepted 17 Nov. 2005; published online 24 July 2007.
Recommended for acceptance by L. Welch.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0275-0804.
Digital Object Identifier no. 10.1109/TC.2007.70789.

- It presents a static slack distribution heuristic to be used for task admittance and demonstrates how task criticality can be handled for hard real-time systems.
- It demonstrates the effectiveness of the SRP slack distribution technique with the dynamic and rate-based scheduling schemes.
- It proposes using the slack toward maintaining checkpoints at nodes for handling faults while saving energy.
- Using real-world examples and simulation, it estimates the overheads and validates its functionalities.

Furthermore, simulation and experimentation were made with contemporary techniques to provide for several results. Results show that the SRP technique improves performance/overhead by 29 percent compared to contemporary techniques.

This paper is organized as follows: Section 2 discusses background and related work. The system model and the SRP technique with the EDF scheduling scheme are introduced in Section 3. Section 4 discusses the SRP technique with the rate-based scheduling scheme. Fault tolerance with the proposed technique is discussed in Section 5. Section 6 presents the results and analysis of simulation. The conclusions and future work are stated in Section 7.

2 Background and Related Work

Embedded systems are energy-sensitive devices with specific implementations related to their applications. They are employed in many critical applications, ranging from sensor network systems, space exploration, and avionics. Reducing power consumption has emerged as a primary goal, in particular for these battery-powered embedded systems. Low-power design techniques for digital components have been intensely studied in the last decade [1], [2]. These techniques consider a single hardware component in isolation or at most a set of homogeneous processors. However, embedded systems are far more complex than these: They consist of several interacting heterogeneous components. Moreover, the input characteristics into these systems are dynamic and not static tasks, as usually assumed. This fact motivated us to design a more realistic general-purpose model for such heterogeneous distributed systems incorporating dynamic task sets.

The two most commonly used techniques that can be used for energy minimization in such embedded systems are Dynamic Voltage Scaling (DVS) [3] and Dynamic Power Management (DPM) [4], [5]. The application of these system-level energy management techniques can be exploited to the maximum if we can take advantage of almost all of the idle time and slack time in between processor busy times. Hence, the major challenge is to design an efficient slack distribution technique which can exploit the slack time and idle time of processors in the distributed heterogeneous systems to the maximum. Various energy-efficient slack management schemes have been proposed for these real-time distributed systems. Zhu et al. proposed energy-efficient scheduling algorithms using slack reclamation for shared-memory homogeneous multiprocessors that allow task migration among processors [6]. In [7], Mishra et al. statically and dynamically manage slack to slow down the scheduled tasks. The available slack on a processor is given to the next incoming task running on that processor and they assume the task graphs to have the same deadline. Luo and Jha proposed static and dynamic scheduling algorithms for periodic and aperiodic task graphs [8]. The static scheduling algorithm uses critical-path analysis and distributes the slack during the initial schedule. The dynamic scheduling algorithm provides best effort service to soft aperiodic tasks and reduces power consumption by varying voltage and frequencies. Luo and Jha [9] also propose increasing the battery life span by reducing the system discharge power profile and distribute the slack based on static scheduling. In [10], Luo and Jha present power-profile and time-constraint driven slack allocation algorithms for variable voltage processors to reduce the power consumption in distributed real-time embedded systems. Shang et al. proposed a runtime distributed mechanism to monitor power dissipation on interconnect links to maintain peak power constraints [11].

Even though there has been lot of work for providing an efficient slack management scheme, very little work caters to heterogeneous distributed systems with dynamic task inputs. Moreover, in [12], even if there is a technique that caters to dynamic task sets, it assumes that the distributed system consists of a set of homogeneous processors. There is a lack of a generalized system model and slack distribution technique, which provides for all of the above characteristics within a performance cost-effective solution. This has essentially been the driving force of the proposed research.

The preliminary studies have been performed for canonical task sets with only static scheduling schemes without a detailed optimality and cost performance study [13]. Furthermore, a detailed design, a study, and an analysis have been performed for a special class of distributed embedded systems with a rate-based scheme. A performance/overhead improved solution has been proposed in comparison with a contemporary rate-based stochastic method [14].

3 System Model

The proposed model is composed of a set of *processing elements (nodes)*, known as embedded system nodes, that will execute application(s) expressed in terms of task sets. The task sets are specified with their incoming arrival period, worst-case computation time at all the nodes executing the task set, and end-to-end deadline. The details of interaction between the processing elements in terms of computation and communication are also specified.

Let "n" be the number of active task sets in a distributed system represented by $\prod = (\prod_1, \ldots, \prod_n)$. Each task set $\prod_i$ is an acyclic graph input task which is processed by a set of nodes in order to achieve specific functions. A group of active task sets constitutes an operating mode or a configuration of the system in an application. Fig. 1 demonstrates the mapping of applications onto the processing elements. An application consists of several functions. In Fig. 1a, we have an application A consisting of five functions, as shown in the box. In order to reduce the communication overhead and to make the computation time demand balance among task nodes, the task graph is modified, as shown in Fig. 1b. The modified task graph is then directly (one-to-one) mapped onto the processing

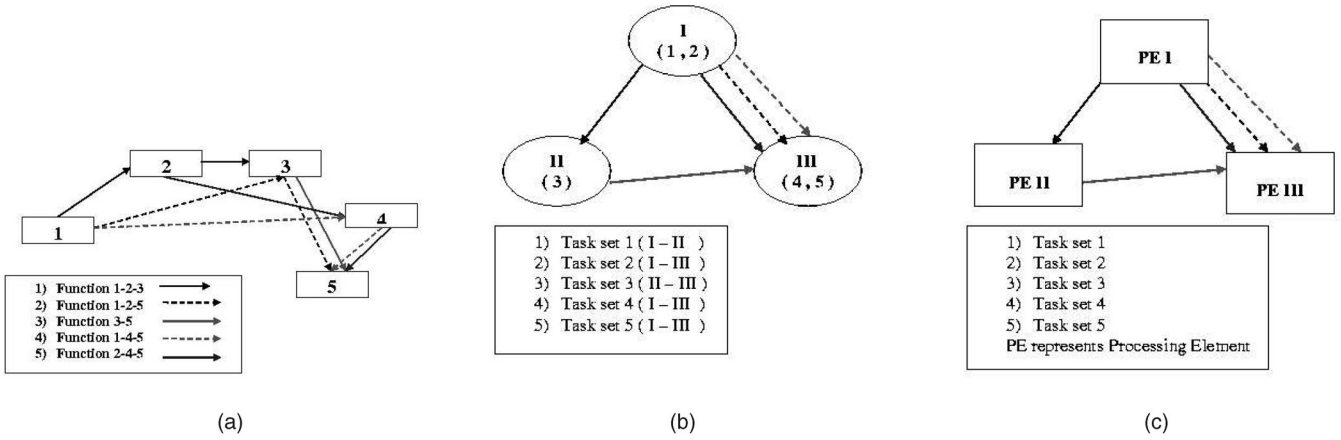


Fig. 1. Application to the task set onto processing element mapping. (a) Application A's five functions. (b) Task graph with computation/communication optimized. (c) Distributed embedded system.

elements of distributed real-time embedded system, as shown in Fig. 1c.

When multiple task sets are active in the distributed system, each node may have to process tasks from different task sets due to resource sharing. We use $(\vec{P}_i, \vec{C}_i, \vec{D}_i, c)$ to specify a task set. A task set is defined as a vector consisting of a group of tasks with specifications such as the periodicity at the source node, worst-case computation time at various processing nodes in its path, the task set's end-to-end deadline, and the criticality level of the task set. The network of nodes that are executing the task set $\prod_i$ communicates by exchanging tasks. The vector $\vec{P}_i$ represents the periodicity of tasks of a task set $\prod_i$. The periodicity is defined for a task set at the source node only. Due to variable delays in the distributed system, the periodicity is lost at subsequent nodes. Any unspecified quantity in this model is represented as ε. The vector $\vec{C}_i \equiv (C_{i1}, \ldots, C_{in})$ represents the worst-case processing time of task set $\prod_i$ at various nodes. Since we consider a heterogeneous distributed system, the tasks of the task set have different computation times at the various nodes in its path. The worst-case computation time also includes the communication overhead due to task transfers among the processing nodes that are proportional to the amount of bytes transferred. The vector $\vec{D}_i$ represents the deadline of the tasks of task set $\prod_i$. The tasks may or may not have a local deadline at the source and intermediate nodes. However, they always have an end-to-end deadline, which is at the destination or the last node serving the task set. Thus, the deadlines at various nodes are represented as $((D_{i1}|\varepsilon), \ldots, D_{in})$, where D_{in} is the hard upper bound delay, by which all of the tasks in a task set $\prod_i$ on n nodes are to be processed. The criticality value associated with a task set is denoted as c. This value is specified for all of the periodic and sporadic task sets and is a real number varying from zero to one, which are both inclusive. Aperiodic task sets have a criticality of zero. The criticality of sporadic task sets and periodic task sets helps to determine the task set's admittance and its priority of processing at a given node.

We denote $\Gamma^i(I)$ as the maximum processing time demand function for task set $\prod_i$ in a given interval of length I. This demand function quantifies the maximum amount of processing time required for a given task set $\prod_i$ due to varying *incoming* tasks and *outstanding* tasks at a node during an interval of length I. We define $\sigma_j(I)$ as the service rate at a particular node during the given interval I by taking all of the active task sets in that interval into account. The service rate defines the frequency of operation at a processing element. It is directly proportional to the computational time demand of various tasks at a given node. In a given interval, the service rate is proportional to the sum of the backlogged (incomplete) tasks carried forward from previous intervals and the upcoming computational time demanded from new tasks that require processing at the node in the given interval. The service rate of a given processing element is set depending upon these demands at the start of every interval. During a given interval, there is no change in the service rate. Each processing node in the network is assumed to be voltage/frequency scalable and has a maximum service rate specified by $\sigma_{\max}$, which is determined from the peak power constraint at a given node. The constraint is specified at the design time and is a limiting factor on the number of task sets that can be admitted into the system. Since we have a heterogeneous distributed system under consideration, we accommodate different maximum service rate values for each of the processing nodes.

3.1 Busy Interval

An important parameter in designing energy-efficient systems is the determination of busy intervals at a node in the distributed system. The busy interval of the node is the active interval of processing time, which is delimited by idle intervals. These intervals help determine the time for the application of Dynamic Voltage Scaling (DVS) or Dynamic Power Management (DPM) at a given node. DVS is applied in the busy intervals and DPM is applied during the idle intervals. The busy interval is determined from the specifications of application inputs, as given in [15].

3.2 Worst-Case Delay and Traffic Descriptor

The "*worst-case delay*" defines the upper bound on the delays experienced at nodes in the path of a task set $\prod_i$. It is represented as a vector for all of the nodes in the path of the given task set. Since we consider heterogeneous processing

elements, the experienced worst-case delays are different for the same task set at different nodes in the distributed system. The instantaneous processing time demand function Γ_t^i is the "*traffic descriptor*" at that node. The *traffic descriptor* sets the maximum rate function or the service rate at a given node and is dependent on the scheduling policy at the given node. The maximum rate function is affected by the outstanding and incoming task sets at a node in a given interval. This demand function is unique for a given processing node. The mathematical derivation for the *worst-case delay*, *traffic descriptor*, and the details associated with it at a particular node are discussed in [15]. The proposed SRP Slack Distribution technique and its analysis are based on those initial studies.

3.3 Periodic Service Rate Determination

The online slack management technique that takes advantage of the runtime variations of the executing task sets heavily depends on the service rate at a given node. The service rate is evaluated at the start of every interval. Such a *"periodic service rate" (PSR)* determination is an important characteristic of the proposed SRP model. *PSR* determination is a dynamic mechanism which operates on feedback information from the *traffic descriptor* of a given node for a given interval. The key idea in determining the minimum service rate at a node is to determine the extended processing time within delay bounds. This technique dynamically adapts the frequency/voltage scaling at the processing node by taking advantage of runtime variations in the execution time. Essentially, this new service rate will guarantee the processing of the task sets that will arrive in the upcoming interval and the task sets that arrived during previous intervals and are awaiting processing in the queue by their delay bounds. For peak service rate constrained systems, the maximum service rate will be bounded by the given peak rate. Since the service rate is a normalized service rate, the maximum service rate is set to one. The PSR determination will have different treatments, depending on the scheduling policy at a given node. The conditions to be met during new service rate determination at a given node for an upcoming interval are given as follows:

- First, the new service rate should guarantee the processing of the tasks in the upcoming interval by their delay bounds.
- Second, this service rate must guarantee the processing of the unprocessed tasks that were left in the queue (and that arrived during the previous intervals) by their worst-case delay bounds.
- Last, the new service rate must lie within the peak service rate bound $\sigma_{\max}$ of that processing element.

The analytical treatment for static scheduling schemes has been done in the previous work [13]. This work presents the analytical treatment of dynamic scheduling schemes.

3.4 Dynamic Scheduling Scheme

As the task set inputs become dynamic during system operation, employing static scheduling schemes cannot help in obtaining the maximum runtime benefits in meeting task criticality requirements and providing for optimal energy savings. Hence, for a general-purpose model design, the inclusion of dynamic scheduling schemes (for example, EDF and RBS) is essential for design completeness. Between the two, because EDF is the more efficient and widely used dynamic scheduling scheme, this has been considered for the analysis and experiments with our proposed technique. The analytical derivation of PSR for EDF is presented in the following:

In order to lay out an analytic study, we define the following notations to represent various parameters and inputs:

- $\boldsymbol{n}$: total number of incoming task sets at the node s_{ij} for a given interval of length "$\boldsymbol{I}$."
- "$\boldsymbol{I}$": represents the length or duration of each monitoring interval at a particular node. The value of "$\boldsymbol{I}$" is calculated by dividing the worst-case delay at a given node by the number of monitoring intervals "$\boldsymbol{z}$."
- $\boldsymbol{\omega_I^i}$: represents the fraction of the interval "$\boldsymbol{I}$" during which tasks from task set $\prod_i$ will be processed.
- $\boldsymbol{wc\_delay_{node}^i}$: represents the worst-case delay suffered by the tasks from input task set $\prod_i$ at a given node serving $\prod_i$.
- $\boldsymbol{T}$: gives the system start time. The queue content is zero and there is no task that is being processed at the processing elements or nodes at this time.
- $\boldsymbol{p_t^i}$: represents the actual processing time demanded by the tasks of graph input task set $\prod_i$ that have already arrived at the node before time instant "$\boldsymbol{t}$."
- $\boldsymbol{r_{backlog,t}^i}$: represents the processing time demanded by the unprocessed tasks left in the queue by graph input task set $\prod_i$, which arrived during the interval $\boldsymbol{mod}(\boldsymbol{t}-\boldsymbol{rI}-(\boldsymbol{t}-(\boldsymbol{r}-\boldsymbol{1})\boldsymbol{I}))$ at time instant "$\boldsymbol{t}$," where "$\boldsymbol{r}$" is the maximum number of task sets entering a particular node.
- $\boldsymbol{r_t^i}$: represents the required service rate at time instant "$\boldsymbol{t}$" to guarantee the processing of the tasks from graph input task set $\prod_i$ in the queue, which arrived during the interval $\boldsymbol{mod}(\boldsymbol{t}-\boldsymbol{rI}-(\boldsymbol{t}-(\boldsymbol{r}-\boldsymbol{1})\boldsymbol{I}))$ by their worst-case delay bounds. This worst-case delay bound corresponds to the processing element running at the lowest possible service rate in order to complete the task set at the deadline.

At the system start time, the service rate is given by

$$r_T^i \geq \Gamma_T^i(I)/\omega_{wc\_delay}^i. \tag{1}$$

This service rate guarantees the processing of the tasks of $\prod_i$ that will arrive in the upcoming interval by their delay bounds. This new service rate at the beginning of every interval is determined according to the following:

$$r_t^i = \sum_{r=1}^{u} r_{backlog,t}^i + \Gamma_t^i(I)/\omega_{wc\_delay,node}^i. \tag{2}$$

The corresponding queue is determined according to

$$p_t^i = \sum_{i=0}^{\partial} p_{backlog,t}^i, \tag{3}$$

where $\partial = (t - max(T, (t-(z-1)\times I)/I))$.

The service rate $\mathrm{s\_rate}_{r,t}^i$ and the corresponding processing time demanded by the outstanding tasks that arrived during the interval $(t-zI, t-(z-1)I)$ are given by

$$r^i_{backlog,t} \times (wc\_delay_i - zI) \geq p^i_{backlog,t}, \tag{4}$$

$$p^i_{backlog,t} = p^i_{t-zI} - p^i_{t-(z-1)I} \quad if\ cleared, \tag{5}$$

else

$$p^i_{backlog,t} = p^i_{t-zI} - \sum_{z=0}^{\partial} \sum_{t-zi}^{t-(z-1)i} r^i_{t-zI} + p^i_{t-zI} - p^i_{t-(z-1)I}. \tag{6}$$

Through the following theorems, we shall prove that the proposed technique fulfills the three requirements specified in the previous section.

Theorem 1. *The service rate r^i_t will guarantee the processing of the tasks of a given input task set $\prod_k$ by their worst-case delay bounds.*

Proof. To prove that r^i_t is a valid service rate to guarantee the end-to-end deadline of task in a given task set, the above-mentioned cases have to be proven satisfactorily:

Case 1. The unprocessed tasks that arrived during any outstanding previous interval $(t - zI, t - (z-1)I)$ will have to be processed within the upper bound on their delays by the service rate r^i_t, that is,

$$\begin{aligned} & r^i_t \times (wc\_delay^i_{node} - zI) \geq p^i_{backlog,t} \\ & \Rightarrow (r^i_{1,t} + \ldots\ldots + r^i_{z,t} + \ldots\ldots + r^i_{u,t}) \times (wc\_delay^i_{node} - zI) \\ & \geq p^i_{backlog,,t} \\ & \Rightarrow \{(r^i_{1,t} + \ldots + r^i_{u,t}) \times (wc\_delay^i_{node} - zI) + r^i_{backlog,t}\} \\ & \times (wc\_delay^i_{node} - zI) \geq p^i_{backlog,,t} \end{aligned} \tag{7}$$

$$\Rightarrow (r^i_{1,t} + \ldots + r^i_{u,t}) \times (wc\_delay^i_{node} - zI) + p^i_{r,t} \geq p^i_{backlog,,t}. \tag{8}$$

By substitution, from (4), which is valid, therefore $r^i_{u,t} \times (d - zI) \geq 0 \forall u$.

Case 2. The tasks that will arrive during the upcoming interval and those that are in the queue will have to be processed within their end-to-end deadlines by using this new service rate. In other words,

$$\begin{aligned} & r^i_t \times wc\_delay^i_{node} \geq \Gamma^i_t(I) + \textstyle\sum_{j=1}^{node-1} p^j_t \\ & \Rightarrow (r^1_t + \ldots + r^j_t + \ldots\ldots + r^{node-1}_t) \times wc\_delay^i_{node} \\ & \geq \sum_{j=1}^{node-1} p^j_t + \Gamma^i_t(I) \end{aligned} \tag{9}$$

$$\begin{aligned} & \Rightarrow (\sum_{g=1}^{n-1} p^i_t/(wc\_delay_i - gI)) \times wc\_delay^i_{node} + \Gamma_t(I) \\ & \geq \sum_{j=1}^{node-1} p^j_t + \Gamma^i_t(I), \end{aligned} \tag{10}$$

which is true, therefore $gI \geq 0 \forall g$.

Case 3. The third case to be considered is the preemption that occurs due to the dynamic rescheduling in the system model. The preemption should guarantee the processing of the tasks of the given task set by their worst-case delay bounds. The preemption performed with this model was both at the queue and the node. Two different treatments are discussed for queue and node preemptions of tasks meeting their worst-case delay bounds:

1. *Queue.* If there is preemption in the queue, we can meet it as we have already admitted the task set that takes its worst-case delay into account. This implies that, in the worst case, the task can be pushed to the end of the queue that corresponds to its worst-case delay bound criterion.
2. *Node.* A task of a given task set is preempted at the node if its deadline is greater than the task to be served and, hence, the completion time is less than the end-to-end deadline for the given task. Thus, preempting the subtask still guarantees the meeting of worst-case delay bounds. Moreover, let τ be the total processing time for a subtask and let the time elapsed since its processing be $(t - s)$. Then, $(\tau - s)$ is the processing time left if $(t + s) \leq I$. If this is true, $(\tau - s)$ is less than the worst-case time for performing the preempted task. On the contrary, if $(t + s) > I$, then fraction 1 of $(\tau - s)$, that is, $(\tau - s)_1$, will be less than the fraction of the worst-case time for performing such a task and the other fraction $(\tau - s)_2$ will be taken care of in the next interval with a higher service rate. This is also within bounds as the new task is admitted into the system that meets its worst-case delay bounds. Thus, for node task preemption, there should be an incoming critical task, the interval length should remain unchanged, and the queue should be full. Moreover, the fraction of computation for the new task should be bounded by t_b, where $t_b = I - elapsed\ time\ in\ the\ interval$. Since this is a conservative bound, we are able to preempt tasks and provide for dynamic scheduling while meeting the worst-case delay bounds. □

3.5 Criticality Consideration

A critical task usually occurs as a hard real-time sporadic task in the system. Admittance of such prioritized tasks and their processing by given deadlines is important to the overall system performance. In our scheme, each processing element has two queues at its input. One of them is the critical queue, which serves the periodic tasks and the dynamically occurring critical sporadic tasks. The other one is a noncritical queue with no priority that enqueues the aperiodic task sets. The critical queue is maintained based on the criticality of tasks and their age. The noncritical queue keeps the task sets in the insertion order. This consideration helps provide improved response time to the critical sporadic tasks while preventing the starvation of task sets in these queues. Task preemptions and swapping due to task criticality considerations at queues and during task processing help improve the performance of the distributed system. The scheduling at the nodes that take task criticality into account is represented as *FCFS_C*, *WRR_C*, and *EDF_C* for the *FCFS*, *WRR*, and *EDF* scheduling policies, respectively. Experimental results show much improved performance/overhead metrics with this improved technique.

3.6 Algorithmic Design

This section presents the algorithms for the design of the proposed SRP model. The algorithmic explanations are discussed based on the scheduling schemes associated with the node under consideration. When input tasks are processed in the node, there are two basic steps in which they can be handled. First, when the inputs arrive at a node, they are queued for processing. Second, the PSR determination thread takes tasks from the queues and processes them at the desired nodes.

The scheduling schemes under consideration here are two static schemes of First Come, First Served (FCFS) and Weighted Round-Robin (WRR), and a dynamic scheme of EDF. Later, the RBS scheme is applied to the model for the study of the specialized class of Rate-Based Models. In all of these scheduling schemes, the common fact is that each of them has a special queue, called the "*background queue*," for the background noncritical tasks. The critical periodic and sporadic task sets are placed in a queue called the "*critical queue*." Background queue processing at a node is done only when there are no tasks to be processed in the critical queue. For all of the scheduling schemes, the "*enqueue_task*" states the inclusion of task sets into the queues and is given as follows:

```
Algorithm 1.
1 if(tsk.criticality = 0) {
2     // add to the background queue
3     backgroundQ.add(tsk);
4 } else {
5     // the tasks get added to this queue
6     // depending on the scheduling method
7     insert (queue, tsk);
8 }
```

After the task enqueue process, the processing of the task sets in the queue starts and the corresponding algorithm is described as follows:

```
Algorithm 2.
1 if(queue.isEmpty) {
2   process(backgroundQ);
3 } else {
4   process(queue);
5 }
6
```

The background queue is processed in an FCFS manner. If the task does not have the criticality zero, it is processed based on the scheduling scheme at the node under consideration.

3.6.1 FCFS Scheduling Scheme

In the FCFS method, the task sets are inserted into the queue using the FCFS method. However, special care is taken to handle the criticality of task sets. If a task set has a higher level of criticality, then it is scheduled prior to the one that has lower criticality. However, this would lead to starvation of low-criticality task sets. Hence, an aging scheme is introduced during the design of the system model. With the aging scheme, a task with higher age is also given due weight and this weight is taken into consideration while inserting tasks into the queue. The algorithm for insertion into a queue is given as follows:

```
Algorithm 3.
1  // insert tsk into the fcfs queue.
2  insert(queue, tsk) {
3    // find the criticality of the task.
4    criticality = tsk.criticality
5
6    // process each task such that.
7    foreach tsk in queue {
8      // if there is another one with less
       criticality
9      priority = (criticality + (age(tsk)/
       avg_age(queue)))/2
10   if (criticality > priority) {
11     // insert it there.
12     queue_insert(queue, tsk);
13     }
14   }
15
16   // if the task is not inserted
17   if (not inserted) {
18     // add to the end of the queue
19     queue_append(queue, tsk);
20   }
21 } // done insert (queue, tsk)
```

Once a task set is enqueued, the "*processQueue*" is used to process it in the order in which it appears in the queue. The following procedure describes the processing of the task sets:

```
Algorithm 4.
1  // process the fcfs queue
2  processQueue(queue) {
3    // if there is some time left in interval.
4    while(intervalTimeLeft > 0) {
5      // take the first task.
6      tsk = queue.firstTask();
7
8      // if all of the first task can be
       processed
9      if(tsk.extime < intervalTimeLeft) {
10
11     // remove the first one from the queue.
12     queue.removeFirst ();
13
14     // update the time left in interval.
15     intervalTimeLeft = intervalTimeLeft -
       tsk.exttime;
16
17     // if this node is not the last one send to
        the next.
18     if(not lastNode) {
19        send(tsk, node.nextNode);
20     }
21   } else {
22     // else get some work done from the task.
23     tsk.extime = tsk.extime - intervalTimeLeft;
24    intervalTimeLeft = 0;
```

```
25    }
26  }
27 }
```

Hence, the task sets are processed in the order of the insertion into the queue and, if its total execution time is satisfied, it is sent to the next node in its path.

3.6.2 WRR Scheduling Scheme

The WRR scheme also employs the two basic procedures of "*enqueue_task*" and "*processQueue*." The major difference in the "*enqueue_task*" process for WRR is that the task sets are enqueued based on their weights and similar weighted task sets form weighted queues. This implies that if there are w different weights of input task sets, there will be w critical queues for each weight. For the sake of simplicity and considering that, most of the time, all of the task sets have different weights, we create a different queue for each task set. Thus, the total number of queues is $n + 1$ (n for weighted critical queues and one for the noncritical background queue). The criticality and aging considerations still hold here. The pseudocode for the "*enqueue_task*" is given as follows:

```
Algorithm 5.
1  // find the queue that has the task set id
2  // for current incoming message
3  wrr_sub_queue = find_queue
   (wrr_queue, tsk.identifier)
4
5  // append the current message to be processed
6  // onto the sub queue.
7  append_entry(wrr_sub_queue, tsk);
```

The processing of the task sets for each weighted queue follows the same method as discussed for the FCFS scheme, and all queues are given the processing element time based on their proportionate weights. These weighted queues are read from beginning to end and, if the task can be satisfied, it is processed and sent to the next node; else, it is executed for the remaining interval time and kept in the queue for processing in upcoming intervals.

3.6.3 EDF Scheduling Scheme

The EDF scheme includes laying out a schedule based on the deadline of task sets and allows dynamic rescheduling and preemption in the queues and processing nodes. In this method, the task sets are kept in the queue in the order of their approaching deadlines. This implies that a task set with an earlier deadline will be processed prior to others. The pseudocode for inserting task sets in EDF is given as follows:

```
Algorithm 6
1  // insert the task tsk into the queue
2  insert(queue, tsk) {
3    // for each task in the queue
4    foreach tsk1 in queue{
5      //if the deadline is less than queue
         member's deadline
6      if(tsk.deadline < tsk1.deadline) {
7        // insert it at that location.
8        queue_insert(queue, tsk);
9      }
10   }
11   // if the task is still not inserted.
12   if(not inserted) {
13     // add it to the end of the queue
14     queue_append(queue, tsk);
15   }
16 }
```

The processing of the tasks after the enqueue procedure follows the same method as discussed for the FCFS scheme. The queue is read from the beginning to end and, if the task set can be satisfied, it is processed and sent to the next node; else, it is executed for the remaining interval time and kept in the queue for processing in upcoming intervals.

4 Rate-Based Scheduling Schemes

The RBS scheme has gained popularity when dealing with task sets in a network flow environment. For example, in an application in which services packets arrive over a network, the packets' arrival may be highly flow dependent. Their arrival rate dynamically varies throughout the operation of the system. This characteristic intuitively matches the proposed service-rate-based technique presented in this work. However, this requires a separate treatment as the inputs are based on the rates of incoming task sets and are not similar to the previously mentioned real-time specifications. We have also compared the proposed SRP Slack Distribution Technique with a contemporary Slack Management System model used for RBS [14].

4.1 Service Rate-Based Slack Distribution with Rate-Based Scheduling

In the rate-based scheme, the arrival rate of an input task set is the criteria for task processing. At a node, tasks that arrive with higher rates will get a larger slice of runtime in a particular interval. In other words, the processing time is proportional to the rates of operation.

The insert procedure for this method is as shown as follows:

```
Algorithm 7
1  insert(queue, tsk) {
2    queue_append(queue, tsk);
3  }
```

In the *processQueue* procedure, the total rate of tasks is calculated every time and the processing time is divided among all of the task sets in the queue. Once the execution time of the task is satisfied, it is removed from the queue and sent to the next node for processing. The modified procedure for RBS is given as follows:

```
Algorithm 8
1  processQueue(queue) {
2   TotalRate = sum_of_rates(queue);
3   foreach tsk in queue {
4    // give the task time proportional to its
        rate.
5    tskExtime = tsk.rate * intervalTimeLeft/
        totalRate;
6
```

```
7    if(tskExTime > tsk.extime) {
8      // remove the task.
9      queue.removeTsk(tsk);
10     // get the remaining time in the interval.
11     intervalTimeLeft = intervalTimeLeft −
         tsk.exttime;
12     // send it to the next node since it's done
         for this node
13     if(not lastNode) {
14       send(tsk, node.nextNode);
15     }
16     // since the queue has changed, adjust
         the total rate.
17     totalRate = sum_of_rates(queue);
18   } else {
19     // record the time which the current task
         ran.
20     tsk.extime = tsk.extime − tskExTime;
21     // adjust time left in the interval.
22     intervalTimeLeft− = tsk.extime;
23   }
24 }
25 }
```

4.2 Stochastic System Model with the Rate-Based Scheme

This stochastic technique proposes a three-step approach for providing an energy-efficient solution [14]. The first step is the probabilistic method for determining the bounds on the task frequency of operation, the second step is an offline method for static slack management, and the third is an online mechanism for the distribution of the available slack at runtime. The model has been implemented with all ofits potentials and compared with the proposed model, taking one of the real-world benchmarks.

4.3 Parameters for Analysis

Rate-based schemes are mostly applied to control jitter and reduce miss ratio and, hence, these two parameters are compared with both models for our analysis. *Jitter control* is important as it gives a notion of the power or service rate variations in the distributed embedded system. The model helps in providing feedback to the designers on this parameter and, hence, it helps in reducing unnecessary power spikes in the system design and providing for an energy-efficient solution. Similarly, the *miss ratio* or *drop rate* of task sets is another important area of study when considering rate-based system designs. Detailed analysis and results on these parameters are provided in Section 6.

5 Fault Tolerance

With technology enhancement, there has been rapid proliferation of distributed embedded systems. As these systems evolve from very specific cases to everyday commonplace use, ensuring their reliable operation is an important criterion to be considered for the dependability of these distributed embedded systems. Many distributed embedded systems, especially those employed in safety-critical environments, should exhibit dependable operation, even in the presence of software faults. These faults are mostly transient in nature and are caused by exceptions at a node in the distributed embedded system. Monitoring the transient errors and initiating the appropriate corrective action are an important way to tolerate faults. We adopt the well-known *Checkpointing* scheme [2] for handling these software faults. Detecting a software fault in a distributed computation requires finding a (consistent) global state that violates safety properties. For synchronization, failure recovery involves rolling back the program to a previously known and safe state, followed by a reexecution of the program. We propose utilizing the available slack to maintain checkpoints and rollbacks at various nodes, in addition to reducing energy consumption.

5.1 Fault Tolerance with the Proposed Technique

The following notations describe some of the fault-tolerant attributes that have been used in the proposed dynamic slack distribution model:

- $\boldsymbol{cp}_{worstcase,node} =$ worst-case computational time overhead for maintaining checkpoint at a given node (maintaining status information).
- $\boldsymbol{cp}_{actual,node} =$ actual computational time overhead for maintaining checkpoint at a given node.
- $\boldsymbol{cp}_{w,node} =$ computational time overhead for checkpoint write.
- $\boldsymbol{cp}_{r,node} =$ computational time overhead for checkpoint read.
- $\boldsymbol{cp}_{I,node} =$ interval length of the checkpoint at that node (the frequency at which checkpoints are saved).
- $\boldsymbol{k} =$ maximum number of faults that the system can tolerate.
- $v =$ number of times that the checkpoint has to be performed at a given node. (This parameter is necessary to control the number of checkpoints required to mark the checkpoint frequency at a node. It can either be specified by the user input for a nonadaptive scheme or determined based on an adaptive scheme by looking at the fault history at the node.)
- $\boldsymbol{F} =$ feasibility condition bit (0/1) that represents the status of the distributed system whether "$\boldsymbol{k}$" faults can be handled or not).
- $\boldsymbol{ft} =$ status bit that states whether there was a fault in the previous checkpoint interval.
- $\boldsymbol{wc\_delay\_fault} =$ worst-case delay of a task set at a node (in the task set's path), with the maximum number of faults occurring at that node.
- $\boldsymbol{wc\_delay}^i_{node} =$ worst-case delay bound of a task set at a node in its path.
- $\boldsymbol{fault\_delay}^i_{node} =$ delay due to the occurrence of "$\boldsymbol{k}$" faults at a given node.
- $CP(\boldsymbol{cp}_{I,node}, \boldsymbol{cp}_{worst\ case,node}, \boldsymbol{cp}_{I,node} =$ a checkpoint which is defined as a sporadic task with the period $(cp_{I,node})$ and deadline $(cp_{I,node})$ equal to the length of the checkpoint interval. (The checkpoint overhead $(cp_{worst-case,node})$ gives the worst-case computational time/execution time of the given check point.)

For the heterogeneous distributed system under consideration, with a given task set input, the worst-case

execution delay with fault tolerance depends on the task set input specification, the checkpoint specifications, and the maximum number of faults that the distributed system can handle. Out of the several possible scenarios, two possible cases are considered here: first, when all of the "k" faults randomly occur at a node and, second, when all of the "k" faults are distributed across multiple nodes.

When faults are randomly distributed over multiple nodes for a given task set, their worst-case behavior would occur when they occur simultaneously. If such a fault occurs, the rollback will take place at all of the nodes in the task set path, which, in the worst case, would be the same as the rollback with the maximum level of faults occurring at the source node. Hence, the second case is a subset of the first case. If we can guarantee the processing of task sets admitted into the system for the first case, it would guarantee both of the cases under consideration. In order to guarantee fault tolerance, the task set's admittance has to be checked for the worst-case delay bounds, considering the maximum number of faults. Only if the task set meets these worst-case delay bounds can it be admitted into the distributed system. Based on the system under consideration, the following gives the various worst-case delay bounds:

$$wc\_delay\_fault = wc\_delay^{i}_{node} + fault\_delay^{i}_{node}. \quad (11)$$

The delay due to the occurrence of faults only "$fault\_delay^{i}_{node}$" is composed of two types of overheads: the actual checkpoint time overhead and the rollback with the maximum number of faults. The following represents the fault delay at a node of a given task set as a summation of these two time overheads:

$$fault\_delay^{i}_{node} = rb\_overhead + cp_{actual,node}. \quad (12)$$

The worst-case rollback overhead is determined with the maximum number of faults occurring at a node for the given task set. This rollback is determined as a function of the checkpoint interval $cp_{I,node}$, the overhead for reading a checkpoint $cp_{r,node}$, the overhead for writing a checkpoint $cp_{w,node}$, and the maximum number of fault at that node. Equation (13) gives the worst-case rollback overhead:

$$rb\_overhead = k \times (cp_{I,node} + cp_{w,node} + cp_{r,node}), \quad (13)$$

where $rb\_overhead$ represents the time overhead on the rollback with k faults occurring at that node.

In order that the model provides for the guaranteed level of fault tolerance, these worst-case delays have to be met at task admittance and maintained at runtime.

For the fault-tolerant slack distributed technique, the following procedures are presented:

1. The feasibility heuristic of the admittance of task sets with faults.

```
Algorithm 9.
1  for(i = 1; i ≤ (set of all subtasks of the given
     task set); i + +) {
2     for(j = 1; j ≤ k; j + +) {
3        if(wc_delay^i_node + fault_delay^i_node ≤ D_i,node) {
4           //task set admitted into model with "k"
5           //number of faults tolerance.
6           F = 1;
7        else
8           //task set not admitted into model with
              "k"
9           //number of fault tolerance.
10          F = 0;
11          }
12    }
13 }
```

2. The heuristic for the rollback technique at runtime

```
Algorithm 10.
1  for (i = cp_I,node; i ≤ v (cp_I,node); i = i + cp_I,node);
2     } if (ft == 1) {
3        fault_delay^i_node = rb_overhead + cp_actual,node {
4     else {
5        fault_delay^i_node = cp_actual,node;
6     }
7  }
```

These time constraints are provided for guaranteed fault tolerance in the proposed slack distribution model. The next section demonstrates the simulation and analysis of the results of various studies performed on such a model.

6 Experiments and Results

This section is organized into three parts. The first part presents the results for the FCFS, WRR, EDF, and RBS schemes using the proposed Service-Rate-Based Dynamic Slack Distribution Technique. Results due to fault tolerance with the proposed technique are presented in the second part. The third part gives the performance and overhead details.

The simulation was run on a dual Pentium IV hyper-threaded processor with 2 Gbyte memory running the Linux 2.6 operating system. A driver script was used to automate the simulation run for different task sets with their specifications (periodicity, worst-case computation time, and end-to-end deadline). In order to find the actual PSR overhead on a target embedded processor (Intel PXA250 Xscale Processor), a cosimulation strategy was employed, with one of the nodes being the XSacle processor and the other nodes simulated on the Pentium IV processor. There were four power levels considered in the XScale processor. Although, for heterogeneous systems, these levels will be different, for simplicity, the other simulated nodes also had four power levels.

Table 1 gives the characteristics of the benchmarks used in validating the proposed technique. The two types of benchmarks used here are the standard TGFF [16] random benchmarks (TGFF I and TGFF II) and the real-world Integrated Multimedia (IM) benchmarks (IM I and IM II).

Fig. 2 shows the task graph of the three applications for the IM I benchmark. Table 2 shows the individual task sets of each of the applications (MPEG, MP3, and ADPCM) of IM I. It can be observed that there are nine, seven, and two task sets for MPEG, MP3, and ADPCM, respectively. The total number of task sets for the IM I benchmark is the sum of all three applications, that is, 18 $(9 + 7 + 2)$.

TABLE 1
Characteristics of Benchmarks

Benchmarks	# of Nodes	# of Task Set Inputs	Benchmark Specification	Composition
TGFF1	27	180	(27, 180)	Random
TGFF2	30	192	(30, 192)	Random
IM I	10	18(9, 7, 2)	(10, 18)	MPEG MP3, ADPCM
IM II	12	34(26, 8)	(12, 34)	DSP, JPEG

6.1 Results and Analysis

6.1.1 Study of System Energy Savings at Different Normalized Peak Service Rates

Fig. 3 shows the comparison of energy savings with three scheduling schemes of FCFS, WRR, and EDF, where the peak service rate is scaled from 1 to 0.7. This study implies the impact of Dynamic Slack Management when applied to various processing elements executing IM I. There is a similar trend for the IM II benchmark. It may be observed that the EDF scheduling scheme outperforms other conventional schemes, as expected. Although it would be expected that WRR performs better than the FCFS scheduling scheme, our results show that WRR and FCFS have comparable energy savings. This is due to the fact that the improvement in energy savings due to the incorporation of a smarter scheme is masked by the increase in the queue overhead of WRR with respect to FCFS $(n + 1 : 2)$. It is also noteworthy that Dynamic Slack Management is highly effective with a dynamic scheduling scheme rather than with static schemes.

6.1.2 Study of Task Set Drop Rate at Different Normalized Peak Service Rates

When DVS is applied at a processing node to limit the peak service rate, it comes with a drop in the task admittance. In Fig. 4, the Task Set Drop Rate (in percent) is shown, with variations in the Normalized Peak Service Rate. The study has been performed on four benchmarks for three scheduling schemes. Our results show that the drop rate is higher for more service rate constrained systems. A comparison of the three schemes shows that EDF is the best scheme for handling task acceptance rate in comparison to static scheduling schemes. The random benchmarks of TGFF I and TGFF II show a smooth decent and the IM benchmarks show sharp drops. This is attributed to the fact that the random benchmarks have no regularized traffic and are less interdependent in comparison to the real-world ones.

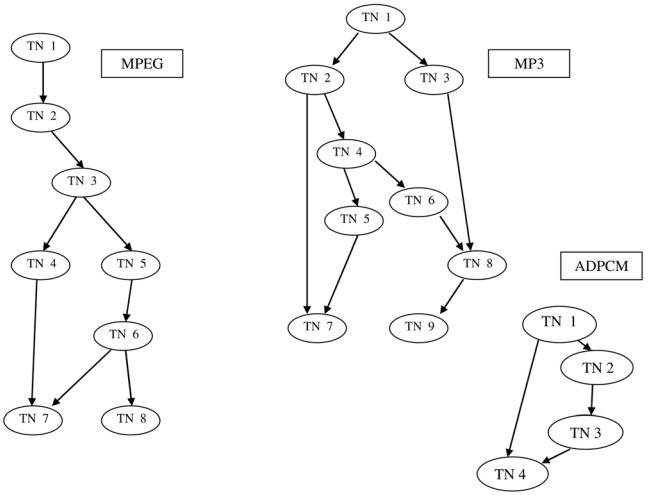


Fig. 2. Integrate Multimedia I, $\mathrm{TN} = \mathrm{Task\ Node}$.

6.1.3 Comparison of Different Slack Distribution Techniques with Different Scheduling Schemes

It is an important study to compare the performance of the proposed service rate-based slack distribution technique against other existing slack distribution techniques. For our study, we have considered the most commonly used slack management schemes of "*Worst-Case Execution Time*," "*Equal Execution Time*," and "*Greedy Slack Management*," as shown in Fig. 5. Results show significant energy savings (up to 70 percent) with the proposed technique in comparison with three prevalent techniques. It is noteworthy that EDF

TABLE 2
Task Set for Integrated Multimedia I

MPEG	MP3	ADPCM
TN 1 - TN 2 - TN 3	TN 1 - TN 2 - TN 7	TN 1 - TN 2 - TN 3 - TN 4
TN 1 - TN 2 - TN 3 - TN 5	TN 1 - TN 2 - TN 4 - TN 5	TN 1 - TN 4
TN 1 - TN 2 - TN 3 - TN 4	TN 1 - TN 2 - TN 4 - TN 5 - TN 7	
TN 3 - TN 5 - TN 6	TN 1 - TN 2 - TN 4 - TN 6	
TN 3 - TN 4 - TN 7	TN 1 - TN 3 - TN 8	
TN 3 - TN 5 - TN 6 - TN 7	TN 4 - TN 6 - TN 8	
TN 5 - TN 6 - TN 8	TN 4 - TN 5 - TN 7	
TN 6 - TN 7		
TN 6 - TN 8		

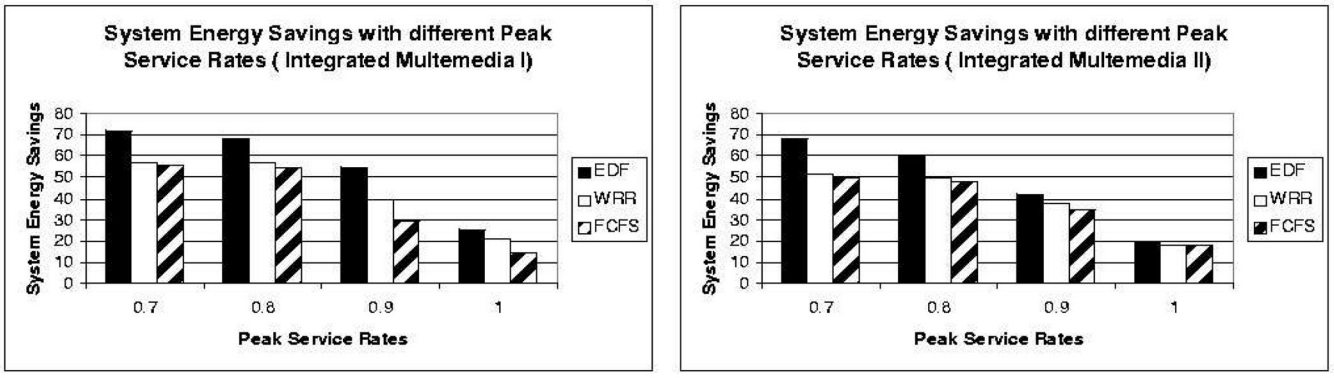
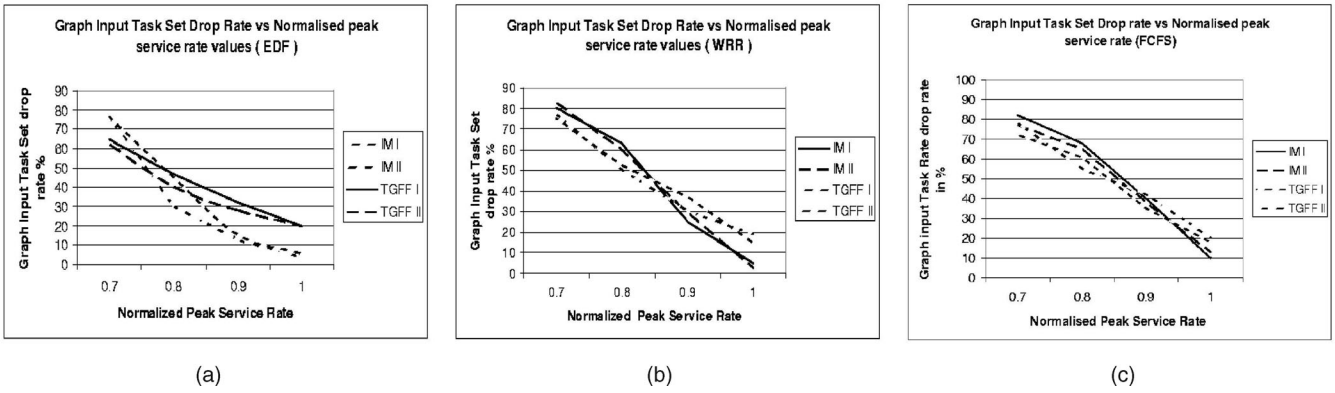
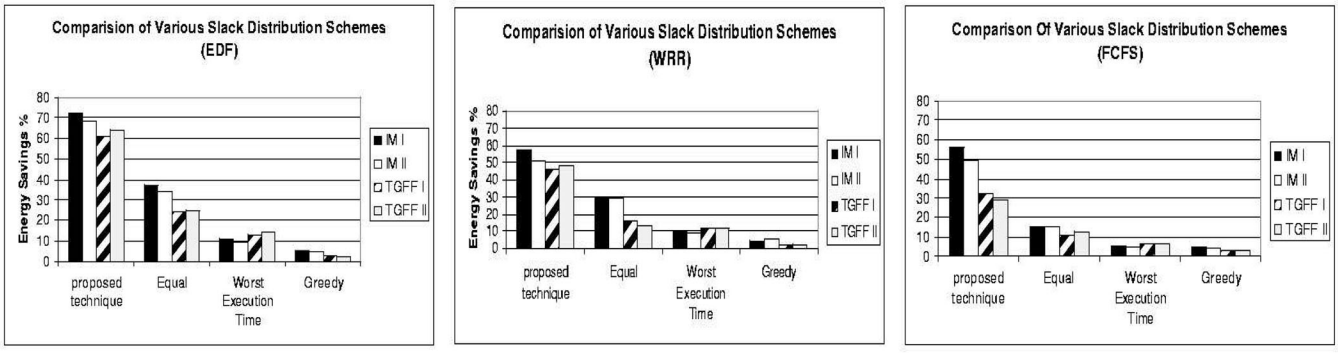


Fig. 3. Energy Savings for (a) Integrated Multimedia I and (b) Integrated Multimedia II.



(a) (b) (c)

Fig. 4. Drop rate versus service rate. (a) EDF. (b) WRR. (c) FCFS.



(a) (b) (c)

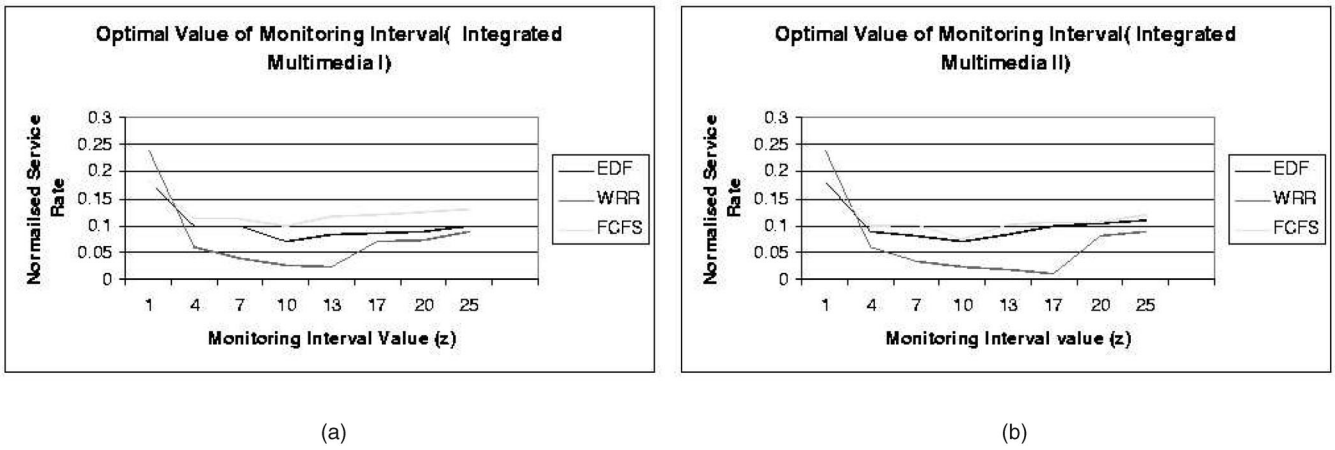
Fig. 5. Comparison of slack distribution techniques for energy savings. (a) EDF. (b) WRR. (c) FCFS.

demonstrates most energy savings when coupled with the service-rate-based dynamic slack management technique.

6.1.4 *Study of Finding the Optimal Value of Monitoring Interval*

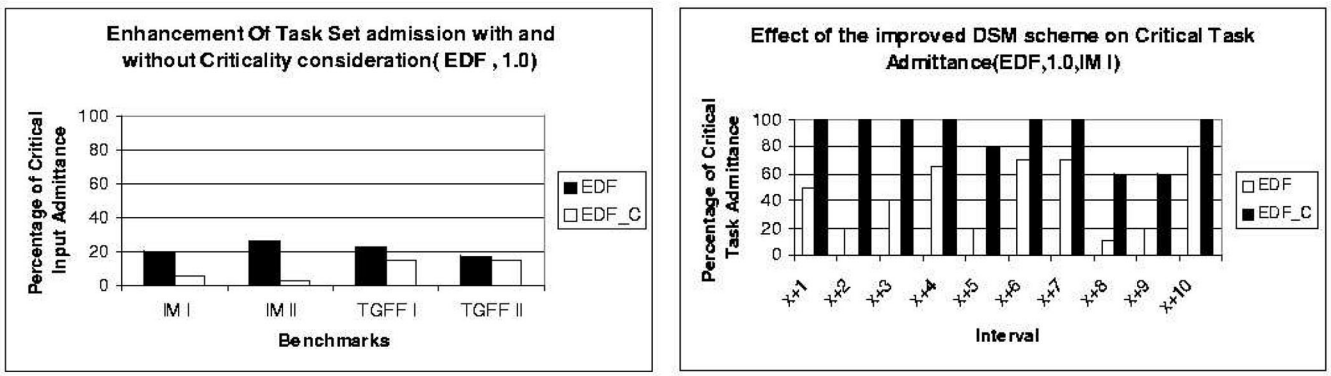
The benefits of the SRP Dynamic Slack Distribution Technique lie in exploiting the dynamic slack available at runtime. This depends on timely monitoring of these slacks and setting of the service rate. Thus, MI plays a key role in maximizing this benefit. A higher MI value indicates frequent monitoring and, hence, frequent adaptation of the runtime slacks. However, it leads to higher overheads. A lower MI value leads to the loss of available slack in a given interval to be used by the system. Thus, for a given application, a suitable MI value should be determined for maximizing the service rate that will yield higher slack. Fig. 6 demonstrates the variation in the normalized service rate for different MI values for IM I and IM II.

Our results show that there is a single optimal value of MI for a given benchmark which is best suited for each scheduling scheme. Analysis has been done for all four benchmarks and all three scheduling schemes. Due to space constraints, only the two real-world cases of MI I and MI II are included. An important observation is that the WRR scheme has a lower service rate than the FCFS and EDF schemes for a given MI value since WRR has multiple queues and the workload gets divided into each of the queues based upon their weights. Thus, inputs of different



(a) (b)

Fig. 6. Optimal monitoring interval: (a) IM I and (b) IM II.



(a) (b)

Fig. 7. Critical task set admittance enhancements. (a) At task set admittance. (b) At runtime.

weights do not interfere with each other's scheduling and processing.

6.1.5 Study of Task Criticality

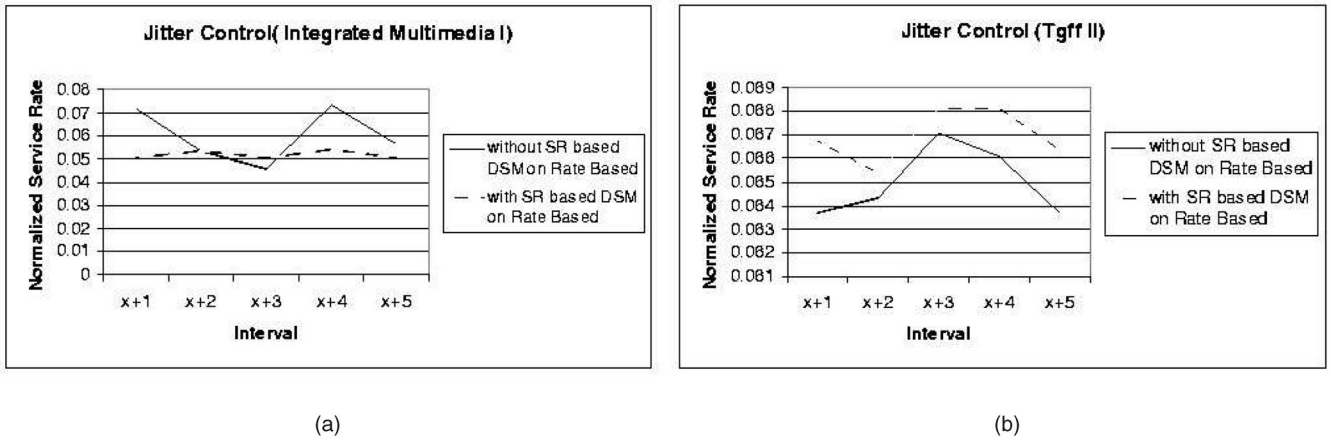
This study shows the impact of the proposed SRP Dynamic Slack Management Technique in handling critical task sets. The two important studies undertaken with this system specification are that of task criticality consideration at task set admittance and at system runtime. Fig 7a shows the improvement in the admittance of critical task sets with and without task set criticality consideration at task set admission in the proposed technique. This is important at task admittance as, for a power-constrained system, criticality plays a major role in deciding task admittance. For example, in IM I, we have three applications: MPEG, MP3, and ADPCM. If the system is power constrained, the criticality consideration can drop a less important application for the user and consider a more important application to serve in the system. Without criticality consideration in the system model design, all tasks have equal criticality. Hence, an important task having higher priority could be dropped due to the presence of less critical tasks ahead of it in the queue. An increase in the critical task set admittance is seen when task criticality is considered here.

Fig. 7b shows the improvement in the admittance of critical task sets with and without the task set criticality consideration at runtime in the proposed technique. It is noteworthy that the proposed SRP Slack Distribution Technique aids greater critical task admittance at runtime.

6.1.6 Study of Jitter Control

The next two studies are for the class of the RBS Scheme incorporated into the proposed Slack Distribution Technique. Since *jitter* is the cause of voltage fluctuations and, hence, unwanted power variations, jitter control is an important criterion for study during the design of such system models. Fig. 8 shows the effect of SRP Slack Distribution on jitter control and compares the results without the slack management for RBS. An important observation based on Fig. 8b is that the proposed technique with RBS is not suited to random workloads but is well suited to real-world multimedia applications.

We also studied the jitter control behavior of our proposed Service Rate-Based Slack Distribution Technique and compare this with a contemporary rate-based scheme, that is, the Stochastic Model [14] used in the RBS scheme. The results in Fig. 9 show that the proposed Service Rate-Based Slack Distribution Technique provides for a reduced service rate compared to the stochastic model [14] while providing comparable jitter control. This indicates that the proposed scheme is suitable for low-power consumption



(a) (b)

Fig. 8. Jitter Control: (a) IM I and (b) TGFF II.

for the same level of jitter control. The benchmark under consideration for this study was IM I.

6.1.7 Study of Miss Ratio/Drop Rate

For Rate-Based Schemes, we also compare the Miss Ratio or Drop Rate in our analysis. Drop Rate gives the number of task sets that are rejected from the total number of task sets requested for service in the distributed system.

Fig. 10 shows the drop rate for the proposed SRP Slack Distribution Technique with and without the rate-based scheme. The results are obtained for the four benchmarks under consideration. It is seen that the drop rate is reduced when the proposed scheme is applied to RBS.

The drop rate comparison has also been performed with the proposed technique and the Stochastic Model [14]. Results show that both techniques have comparable drop rates, about 26.6 percent for the IM I benchmark.

6.2 Fault-Tolerant Results

This section studies the effects of including fault tolerance in the proposed technique. The study has been made by taking the critical task admittance and the overhead due to fault tolerance (service rate calculation, checkpoint determination, and the rollback overhead) into account.

6.2.1 Study of Critical Task Admittance

This study analyzes the paid price for the critical task admittance in the presence of handling faults. This study has been done with maximum faults at one node and also for faults distributed in multiple nodes in the task set. The faults are introduced in a random manner within the busy interval of operation at a given node. For our studies, we have considered a group of random input task sets (generated from TGFF [16]) consisting of 25 input task sets employing the EDF scheduling policy, with the peak normalized service rate set to 1.0.

Our results show that the critical task set acceptance drops with the presence of fault tolerance into the model. There is also a higher drop of task sets with the increase in the maximum fault-tolerant level. The results are dependent on the benchmark taken into consideration. This type of study helps the designers in predicting, in advance, the maximum fault level that they can provide with a required amount of critical task acceptance. Fig. 11a represents the number of tasks admitted with a given maximum fault-tolerant value at a node in the task set path of the distributed system. Fig. 11b represents the same study when the fault is distributed among multiple nodes (three nodes) in the path of the task set.

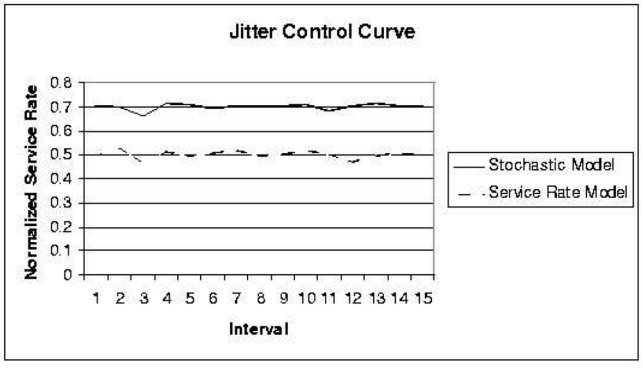


Fig. 9. Jitter control: Stochastic and Service Rate models.

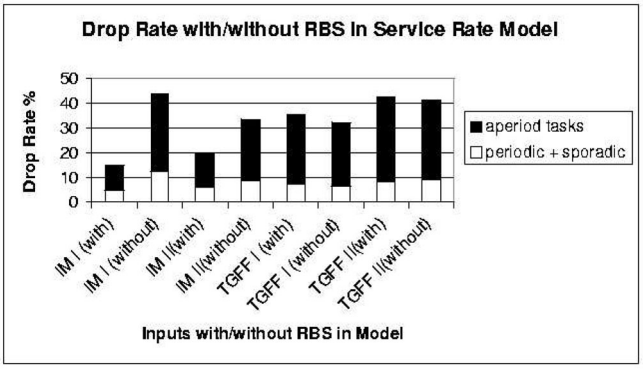
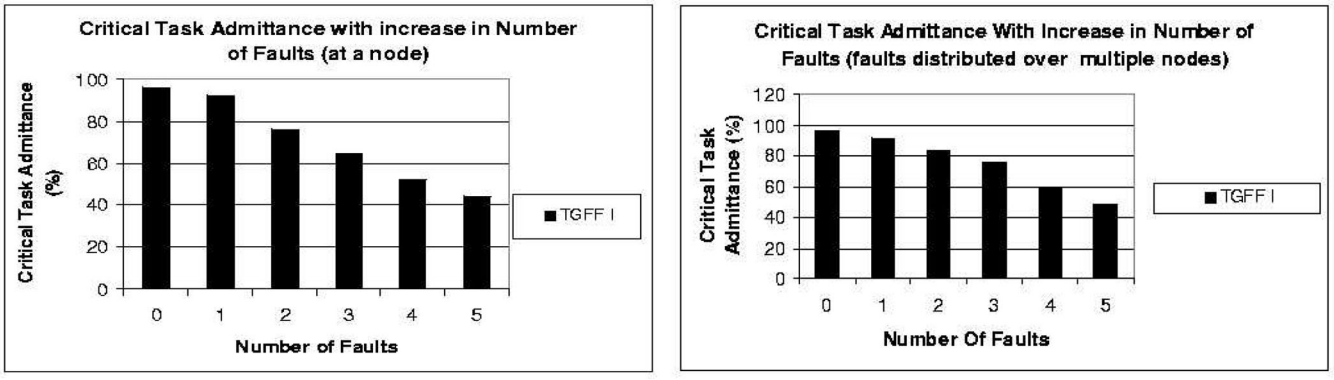
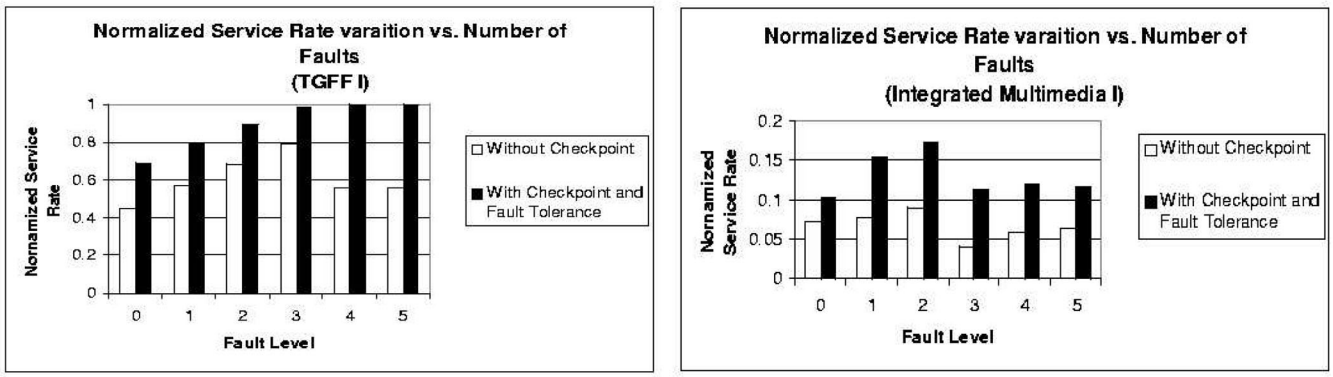


Fig. 10. Drop rate with/without RBS.



(a) (b)

Fig. 11. Critical task admittance (in percent) with variation in the number of faults. (a) Faults at one node. (b) Faults distributed at multiple nodes.



(a) (b)

Fig. 12. Service rate variation with/without fault tolerance. (a) TGFF I. (b) IM I.

6.2.2 Study of Normalized Service Rate Variation with Fault Tolerance

The next study is on the variation of the service rate with the fault tolerance into the proposed technique. The service rate will be higher compared to the model without fault tolerance as there is a requirement of a higher rate due to the checkpoint interval and the rollback overhead.

Figs. 12a and 12b represent this service rate variation with fault tolerance in the proposed technique employing the EDF scheduling policy, with the normalized peak service rate set to 1.0. The input taken for consideration is the random test case with 25 input task sets (TGFF) and the IM test case with 10 graph input task sets, respectively. The random test case has been modified to consider uniform traffic throughout in order to display the effect of service rate with an increase in the number of faults in a distributed system, keeping the traffic demand uniform. It is noteworthy that, after three faults, the service rate is limited by the peak service rate and, hence, there is a decrease in performance and a drop in the admission of task sets into the system. For the IM I application, this service rate reduces even with the increase in the fault levels as the demand on the node is reduced. A fault level of 0 represents that there is only checkpointing, but there is no occurrence of any faults and, hence, there is no rollback overhead.

6.2.3 Study of Determining Optimal Checkpoint Interval

Finding the optimal value of checkpoint is important for the development of a performance/overhead improved distributed embedded design. The checkpoint interval determines the frequency of monitoring for faults and is also proportional to the rollback distance. The checkpoint interval should be determined by taking three important aspects into account. These are the number of occurrences of faults in a given interval, the maximum number of faults, and the maximum rollback overheads in the distributed system under design. The performance/overhead gives the fault tolerance provided with the desired performance per overhead. This value is normalized for the sake of analysis.

The proposed fault-tolerant slack distribution technique can provide information to the designers that would determine a priori the optimal checkpoint interval for a given application. Fig. 13 gives the variation of the normalized service rate for different checkpoint interval values for the IM I and IM II benchmarks. The results show that a checkpoint interval value between 10 and 12 is best suited for IM I. This value is between 12 and 15 for IM II. These bounds provide designers with prior information about the optimal design for a given benchmark under consideration. The curve seems to rise after the optimal point due the fact that there is a greater rollback overhead encountered on faults due to the larger checkpoint interval length.

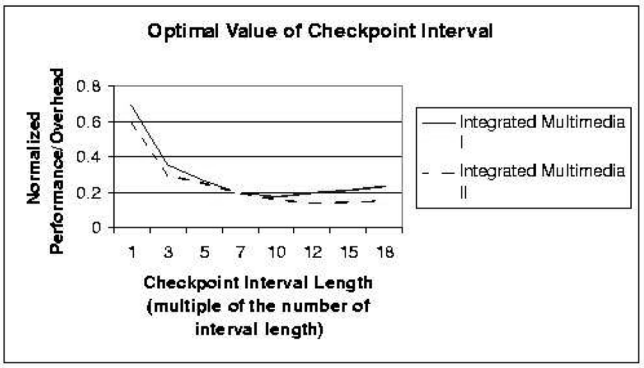


Fig. 13. Optimal value of the checkpoint interval for a given benchmark.

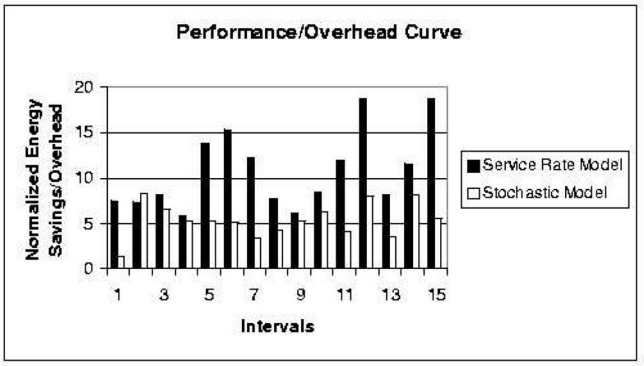


Fig. 15. Performance/overhead curve.

6.3 Cost Performance Study

6.3.1 Slack Management Technique Study

Cost performance study is of paramount importance to any model development. This is more relevant to low power systems. A valid study can be done if our model is compared to a similar model for such target systems. For our study, we have considered the slack management technique AND/OR Model given in [12]. The performance in terms of net system energy savings and the overhead in terms of space and time are determined for the IM I benchmark for both techniques. The performance/overhead metric was calculated as the net energy savings/overhead for both techniques along the interval and is given in Fig. 14. The results show higher performance/overhead by using the proposed technique.

6.3.2 Rate-Based Scheme Study

The performance/overhead study for rate-based schemes using the proposed model and the Stochastic Model [14] is examined here. This study helps us validate the quality of the proposed model in contrast to a contemporary technique [12]. The performance, that is, net system energy savings, and the cost, that is, the overhead in time and space, are calculated for both techniques and the energy savings/overhead is determined for analysis. The results demonstrate the efficiency of the proposed technique (Fig. 15).

6.3.3 Study of the Overhead of Fault Tolerance in the Proposed Technique

The overhead incurred with the proposed fault tolerance into the model is an important study for design of the system. For our study, we have considered the IM I application in Fig. 16. The overhead increases with the increase in the maximum fault-tolerant level of the system. The trend shows that the curve seems to saturate around a fault level of 4 or 5. It should also be noted that the overhead incurred is nearly 1.5 ms with the maximum number of faults as 5. This indicates that, instead of a dynamic fault-tolerant scheme as proposed, a prediction-based offline fault detection scheme can be used to reduce these high overheads. This estimate helps the designer in making a conscious decision to balance the maximum fault-tolerant level with the maximum possible sustainable overhead for a given system under study.

6.3.4 Study of Periodic Service Rate Overhead

One of the most important overheads of the proposed technique is the determination of periodic service rate. The periodic service rate overhead has been determined during simulation. The overhead can be aptly determined if a cosimulation strategy is employed, with one of the nodes being an embedded processor. The Intel Xscale processor was considered for our simulation. This embedded processor emulated one node in the distributed framework, while the rest of the nodes ran on the same 2 GHz Linux platform. The graph in Fig. 17 demonstrates the PSR overhead, with an increase in the number of task set demands at a node (Xscale

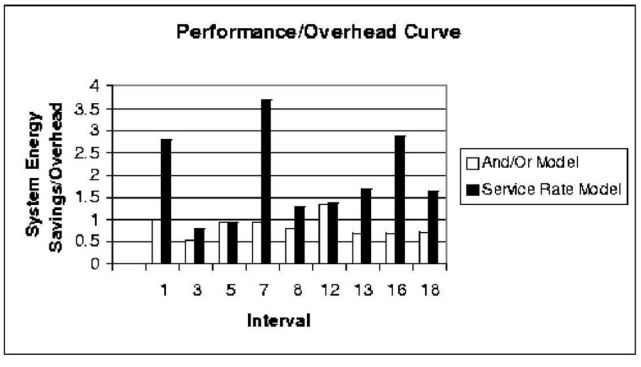


Fig. 14. Performance/overhead curve.

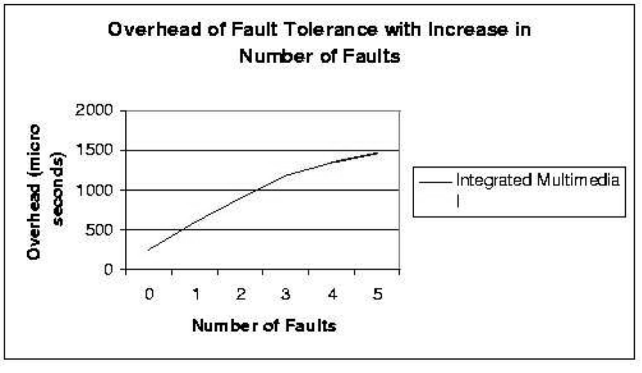


Fig. 16. Service rate variation with/without fault tolerance.

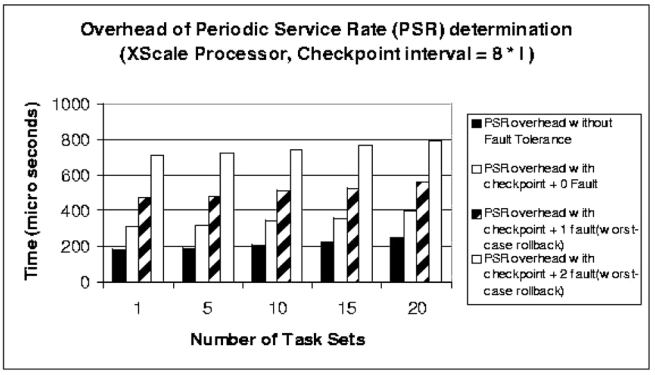


Fig. 17. Periodic service rate overhead.

processor) for a maximum of 20 task sets. The overhead has been determined for four cases: the normal case without any fault tolerance, the PSR with the checkpoint overhead and no faults, PSR with the checkpoint overhead and one fault in the system, and PSR with the checkpoint overhead and two faults in the system. It is noteworthy that an online fault-tolerant scheme is a costly affair.

7 Conclusions and Future Work

This work has introduced an improved dynamic service rate-based slack management technique for providing a power-aware energy-efficient solution for real-time distributed embedded systems. The major contribution of this work is the development of a unique slack management technique based on service rate and change in intervals for static and dynamic scheduling schemes. The results show almost 29 percent improvement in the performance overhead (time and space) over contemporary dynamic slack management techniques for such real-time distributed embedded systems. A model and simulation environment considering the heterogeneity of such real-time distributed embedded systems has been developed. Further, a class of rate-based schemes has been incorporated into the model and slack management technique. Another important contribution is the incorporation of a fault-tolerant scheme into the proposed technique for the design of a reliable stimulatory model. The results show that the fault-tolerant design inclusion helps in controlling a considerable number of faults at the cost of increased energy consumption and lowering of task admittance. Also, the optimality study on MI and checkpoint interval values helps designers in making early conscious decisions on the design of such embedded systems. For future work, there can be more work on dependability and increased reliability studies and modeling with the proposed technique. Also, security issues can be considered for such real-time distributed systems. One important future work is implementing such a technique in a real-world distributed framework, that is, sensor network applications.

References

[1] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. Design Automation of Electronic Systems,* vol. 1, no. 1, pp. 3-56, 1996.

[2] G. Cao and M. Singhal, "On Coordinated Checkpointing in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 12, pp. 1213-1225, Dec. 1998.

[3] L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. VLSI Systems,* pp. 299-316, 2000.

[4] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Proc. Int'l Symp. Low Power Electronics and Design,* pp. 197-202, 1998.

[5] M. Weiser, B. Welsh, A. Demers, and S. Shenker, "Scheduling for Reducing CPU Energy," *Proc. First Usenix Symp. Operating Systems Design and Implementation,* pp. 13-23, 1994.

[6] D. Zhu, R. Melham, and B. Childers, "Scheduling with Dynamic Voltage/Speed Adjustments Using Slack Reclamation in Multi-Processor Real-Time Systems," *Proc. 22nd IEEE Real-Time Systems Symp.,* pp. 84-94, Dec. 2001.

[7] R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melham, "Energy-Aware Scheduling for Distributed Real-Time Systems," *Proc. 17th Int'l Parallel and Distributed Processing Symp.,* 2003.

[8] J. Luo and N.K. Jha, "Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems," *Proc. Seventh Asia and South Pacific Design Automation Conf./15th Int'l Conf. VLSI Design,* p. 719, 2002.

[9] J. Luo and N.K. Jha, "Battery-Aware Static Scheduling for Distributed Real-Time Embedded Systems," *Proc. 38th Design Automation Conf.,* pp. 444-449, 2001.

[10] J. Luo and N.K. Jha, "Power-Profile Driven Variable Voltage Sealing for Heterogeneous Distributed Real-Time Embedded Systems," *Proc. 16th Int'l Conf. VLSI Design,* p. 369, 2003.

[11] L. Shang, L.-S. Peh, and N.K. Jha, "Powerherd: Dynamic Satisfaction of Peak Power Constraints in Interconnection Networks," *Proc. 17th Ann. Int'l Conf. Supercomputing,* pp. 98-108, 2003.

[12] D. Zhu, D. Mosse, and R. Melham, "Power-Aware Scheduling for AND/OR Graphs in Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 9, pp. 849-864, Sept. 2004.

[13] R.B. Prathipati, "Energy Efficient Scheduling Techniques for Real-Time Embedded Systems ," master's thesis, Texas A&M Univ., Dec. 2003.

[14] F. Gruian, "Hard Real-Time Scheduling for Low-Energy Using Stochastic Data and DVS Processors," *Proc. Int'l Symp. Low-Power Electronics and Design,* pp. 46-51, 2001.

[15] A. Raha, S. Kamat, and W. Zhao, "Admission Control for Hard Real-Time Connections in ATM LANs," *Proc. IEEE INFOCOM '01,* 2001.

[16] R. Dick, "Task Graph for Free," http://helsinki.ee.princeton.edu/~dickrp/tgff, 2004.

Subrata Acharya received the MS degree in computer science from Texas A&M University in December 2004. She is currently working toward the PhD degree at the University of Pittsburgh. Her research interests include security, networking, and embedded systems. She is a member of the IEEE.

Rabi N. Mahapatra received the PhD degree in computer engineering from the Indian Institute of Technology (IIT), Kharagpur, India, in 1992. He was an assistant professor in the Electronics and Communication Engineering Department, IIT, Kharagpur, until 1995. He is currently an associate professor with the Department of Computer Science and the director of the Embedded Systems Research at Texas A&M University. His current research interests include embedded systems codesign, system-on-chip, VLSI design, and computer architectures. He has published more than 85 papers in reputable journals and conference proceedings. His recent publications can be found at http://faculty.cs.tamu.edu/rabi/. He is a senior member of the IEEE and a member of the IEEE Computer Socety.