A Dynamic Slack Management Technique for Real-Time Distributed Embedded System with Enhanced Fault Tolerance and Resource Constraints

Santhi Baskaran, I. Gugan, A. Aswin Kumar and D. Govindarajan

Department of Information Technology
Pondicherry Engineering College
Puducherry-605014, India

***Abstract*- This project work aims to develop a dynamic slack management technique, for real-time distributed embedded systems to reduce the total energy consumption in addition to timing, precedence and resource constraints. The Slack Distribution Technique proposed considers a modified Feedback Control Scheduling (FCS) algorithm. This algorithm schedules dependent tasks effectively with precedence and resource constraints. It further minimizes the schedule length and utilizes the available slack to increase the energy efficiency. A fault tolerant mechanism uses a deferred-active-backup scheme increases the schedulability and provides reliability to the system.**

Keywords - Real-time; slack; precedence constraints; resource constraint; resource reclaiming; fault tolerance.

I. INTRODUCTION

Many embedded command and control systems used in manufacturing, chemical processing, avionics, telemedicine, and sensor networks are mission-critical. These systems usually comprise of applications that must accomplish certain functionalities in real-time [1]. Dynamic voltage scaling (DVS) is an effective technique to reduce CPU energy. DVS takes advantage of the quadratic relationship between supply voltage and energy consumption, which can result in significant energy savings. By reducing processor clock frequency and supply voltage, it is possible to reduce the energy consumption at the cost of performance of processors [2]. Battery powered portable systems have been widely used in many applications. As the quantity and the functional complexity of battery powered portable devices continue to raise, energy-efficient design of such devices has become increasingly important. Also these systems have to concurrently perform a multitude of complex tasks under stringent time constraints. Thus minimizing power consumption and extending battery lifespan while guaranteeing the timing constraints has become a critical aspect in designing such systems. The interest in distributed systems has been growing steadily since more industrial systems rely on computer-based operations. Therefore the critical applications are being done by the computer in real-time environment must produce desired result at the correct time. The result which is not obtained in the correct time may be disastrous. As per the definition the output of real-time systems not only depends on the correctness of the result but also the time when the result is produced.

In order to make energy efficient, in the scheduling, the execution time of the tasks can be extended up to the worst case delay for each task set. 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. In energy efficient scheduling, the set of tasks will have certain deadline before which they should finish their execution and hence there is always a time gap between the actual execution time and the deadline. It is called slack time [3]. Therefore to minimize the energy consumed and to satisfy the deadline of the tasks, the processors run at variable speeds there by reducing the energy consumed by them. This scheduling will be simulated with various task sets on different set of processors using various algorithms.

Precedence and resource constraints are important factors which must be considered. The precedence relationships are represented as a Directed Acyclic Graph (DAG) consisting of nodes that represent computations and edges that represent the dependency between the nodes [4]. If resources are available there is no problem for allocation of these to various tasks, else resources should be allocate deficiently among tasks and also care should be taken to see that no deadlock occurs. Therefore it is necessary to introduce resource management mechanisms that can adapt to dynamic changes in resource availability and requirement. We proposed a modified FCS algorithm, which employs software feedback loops that dynamically control resource allocation in response to changes in input workload, task precedence and resource availability. We are improving the fault tolerance of the system using deferred-active-backup system [5].

II. RELATED WORK

The two most commonly used techniques that can be used for energy minimization in such embedded systems are Dynamic Voltage Scaling (DVS) [6] and Dynamic Power Management (DPM) [7]. 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. Various energy-efficient slack management schemes have been proposed for these real-time distributed systems. 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. Resource adaptation techniques for energy management in distributed real-time systems need to be coordinated to meet global energy and real-time requirements. This issue is addressed based on feedback-based techniques to allocate the overall slack in the entire system.

Fault-tolerant scheduling is an attractive avenue to achieving high reliability in uniprocessor and multiprocessor real-time systems. One of the first fault-tolerant scheduling mechanisms for uniprocessor real-time systems was developed by Liestman and Campbell [8]. Some researchers also investigated the power management issues in the fault-tolerant real-time systems [9]. Many fault-tolerant scheduling algorithms leverage the primary–backup scheme to tolerate processor failures. In the primary–backup approach, each task has two versions allocated to two different processors. Three variants of the primary–backup approach include: 1) the active-backup-copy-based schemes; 2) the passive-backup-copy-based schemes; and 3) the primary–backup-copy overloading techniques.

In the active-backup-copy-based schemes, the primary and backup copies of each task are executed in parallel on two processors. The active-backup-based schemes exhibit the advantages of requiring no synchronization between two copies and imposing no constraints on the execution times of tasks. However, it is recognized that processor times required by tasks in the active-backup-based approaches are doubled when compared with the passive-backup-based schemes. In contrast, the passive-backup-copy-based schemes only execute the backup copy of a task if its primary copy fails to pass the acceptance test. The backup copy of a task can be deallocated from its schedule if the task's primary copy is successfully finished. More importantly, the passive- backup-copy-based schemes can take advantage of the backup copy overloading technique. This overloading technique allows passive-backup copies assigned to different processors to be overlapped on the same process to tolerate a single processor failure. However, the passive-backup-copy-based schemes have a shortcoming of tight timing constraints. The primary–backup-copy overlapping technique allows the primary copy and backup copies to be overlapped in execution times. This technique, which can exploit the advantages of the aforementioned two schemes, is envisioned as a compromise between the other two. Nevertheless, the common drawback of the aforementioned fault-tolerant scheduling schemes is that they merely support a single type of backup copy.

III. MODELS

A. System Model

A distributed system with n homogeneous processors each with its private memory is considered for scheduling the given real-time application. The system requires the complete details of the task processing times (i.e.) the execution time and deadline before program execution. Each processing element (PE) in the system can support discrete voltage and speed changes. We assume that the energy consumption, when the processor is idle, is ignored. The real-time applications can be modeled by a task graph G = (V,E), where V is the set of vertices each of which represents one computation (task), and E is the set of directed edges that represent the data dependencies between vertices. For each directed edge (v_i, v_j), there is a significant inter-processor communication (IPC) cost when the data from vertex v_i in one PE is transmitted to vertex v_j in another PE. The data communication cost in the same processor can be ignored. Each real-time application has an end-to-end deadline D, by which it has to complete its execution and produce the result. Each local task T_i executed at node i have a specified local deadline d_i by which the task has to be processed. The local deadlines assigned must satisfy

$$\sum_{i=1}^{n} d_i \leq D \qquad (1)$$

The frequency selection is influenced by making a task more or less urgent by shifting its deadline back and forth. The range within which the local deadline at node i can be varied is bounded by [Si−, Si+]. The values for

S can be derived from the local task parameters. If $wcet_i$ represents the worst-case execution times of the local task at node i, then

$$S_i^- = wcet_i$$
$$S_i^+ = D - \sum_{j=i+1}^{n} wcet_j \quad (2)$$

The system output consists of the task numbers and how each every task is being scheduled. It also contains the details regarding the speed, energy consumed by the variable voltage processor and also the increase in the computation time for each and every task which is shown graphically. It also includes the fault tolerance ratio.

B. Resource Model

1. Basic Assumptions

To model non-CPU resources and resource requests, we make the following assumptions:

1) Resources are reusable and can be shared, but have mutual exclusion constraints. Thus, only one task can be using a resource at any given time. If multiple identical resources or multiple instances of the same resource are available, each identical instance of a resource should be considered as a distinct resource. This applies to physical resources, such as disks and network segments, as well as logical resources, such as critical code sections that are guarded by mutexes.
2) Only a single instance of a resource is present in the system. This requires that a task explicitly specify which resource it wants to access. This is exactly the same resource model as assumed in protocols such as the Priority Inheritance Protocol and Priority Ceiling Protocol [10].
3) A task can only request a single instance of a resource. If multiple resources are needed for a task to make progress, it must acquire all the resources through a set of consecutive resource requests.

2. Resource Request and Release Model

During the lifetime of a task, it may request one or more resources. In general, the requested time intervals of holding resources may be overlapped.

We assume that a task can explicitly release resources before the end of its execution. Thus, it is necessary for a task that is requesting a resource to specify the time to hold the requested resource. We refer to this time as HoldTime [11]. The scheduler uses the HoldTime information at run time to make scheduling decisions.

C. Precedence Constraints

Tasks can also have precedence constraints. For example, a thread T_i can become eligible for execution only after a task T_j has completed because T_i may require T_j's results. For implementing precedence constraints DAG is used.

Precedence constraints between tasks can also be modeled as resource dependencies. The precedence constraint that T_j precedes T_i is equivalent to the situation where T_i requires a logical resource (before it can start its execution) that is available only after T_j has completed its execution. Thus, if T_j has completed its execution before T_i arrives, then this logical resource is immediately available for T_i and T_i becomes eligible to execute upon arrival. This respects the precedence relation semantics. Furthermore, if T_j has not completed its execution when T_i arrives, then the logical resource is not available and, therefore, T_i is conceptually blocked upon arrival. Later, when T_j competes its execution, the logical resource becomes available and T_i is unblocked. This again respects the semantics of the precedence relation. This technique requires that both T_i and T_j share a binary semaphore S with an initial value zero. The first operation of T_i is to execute P(S) and the last operation of T_j is to execute V(S).

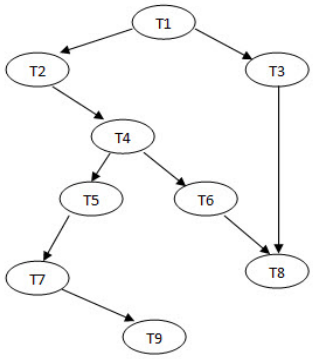

Figure1: Precedence constraints of a task set

Thus, by allowing resource dependencies in the task model, we also allow, albeit indirectly, precedence constraints between tasks.

D. Fault Tolerance Model

In this section, we describe a fault tolerance model of real-time distributed systems. This model is composed of a set of processors as well as a set of real-time primary copy tasks along with a set of corresponding backup copies of the real-time tasks running in the distributed system. In this study, we consider real-time distributed systems where processors accessing their local memory modules are connected to one another via an interconnection network. Formally, a real-time distributed system model is composed of $\tau_1, \tau_2, \tau_3 \ldots \tau_N$ of tasks in addition to $P_1, P_2, \ldots, P_M$ of processors executing the task set. The ith periodic preemptive task τ_i is characterized by two parameters: period T_i and execution time C_i. A set $\{\beta_1, \beta_2, \beta_3 \ldots \beta_N\}$ of backup copies of periodic tasks are introduced to make fault tolerance possible. d_i is the execution time of β_i.

In our system model, each backup copy may be in one of two states: passive-backup copy or active-backup copy. We assign a task's primary copy before assigning its backup copy regardless of the backup copy's status form. The status forms of backup copies are formally determined by the following:

$$\text{Status}(\beta_i) = \begin{cases} \text{passive}, & B_i > d_i \\ \text{active}, & B_i \le d_i \end{cases}$$
$$\text{where } P(\tau_i)=P_j,\ P(\beta_i)=P_k, \text{ and } B_i=T_i-R(i,j) \qquad (3)$$

$R(i, j)$ denotes the worst case response time or WCRT of τ_i on processor P_j [12]. B_i denotes the recovery time for β_i, i.e., the time left after the execution of τ_i. $P(\tau_i)$ or $P(\beta_i)$ denotes the processor on which τ_i or β_i is scheduled on. For ease of presentation, γ_i represents a primary copy or a backup copy, i.e., $\gamma_i = \tau_i$ or β_i. Equation (3) indicates that an active backup copy must be running in parallel with its primary copy, whereas a passive-backup copy only needs to be executed if its primary copy fails.

IV. PROPOSED WORK

The overall framework for the proposed system is shown in the figure

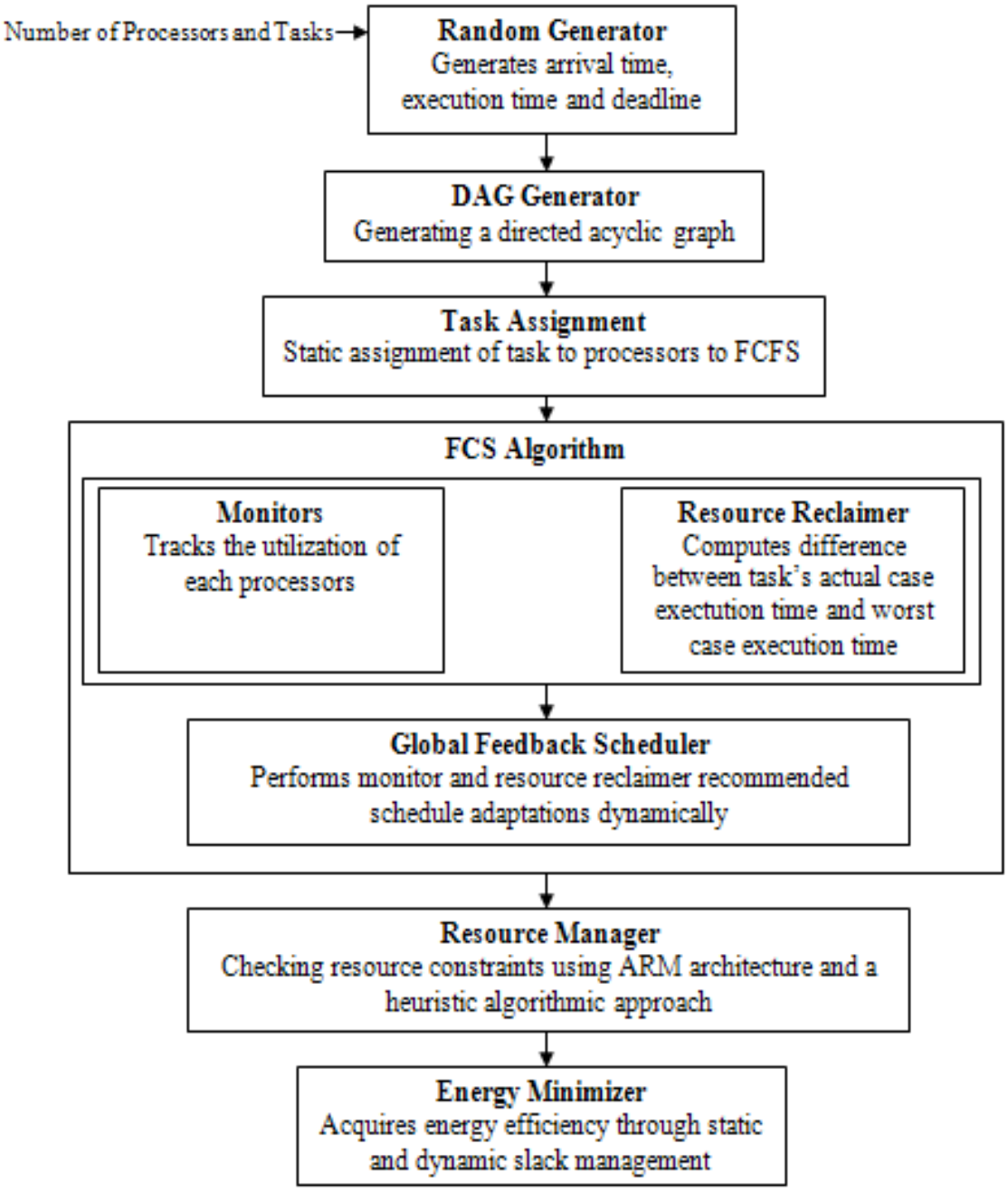

Figure 2: Architecture of Energy-Efficient FCS algorithm

A. First Come First Served (FCFS) Scheduling Scheme

In the FCFS method tasks are inserted into the queue based on the arrival time of the task. The tasks are inserted into the queue in the increasing order of the arrival time. The pseudo code for inserting task sets in FCFS is given in Figure 3.

```
Insert (queue, t) // inserts the task t into the FCFS queue
{
        for each task t_i in queue //i = 1 to n ( number of current tasks
in the queue)
                {
                   if(t.arrival_time < t_i.arrival_time)
                    {
                         queue_insert(queue, t, t_i)  // insert t before t_i
                    }
                }
        //if the task is not inserted
        if(not inserted) //add to the end of the queue
        {
                queue_append(queue, t);
        }
}
```

Figure 3: FCFS Pseudo Code

B. Modified FCS Algorithm

A modified FCS (feedback control scheduling) algorithm has been proposed for homogeneous distributed real-time systems, which include tasks supporting both precedence and resource constraints. The algorithm is based on a mixed integer predictive control approach. The algorithm is based feedback control scheduling (FCS). This algorithm can provide real-time performance guarantees efficiently, even in open environments. The feedback control scheduling algorithm framework has three components; Resource reclaimer that computes difference between task's actual execution time and worst case execution time for local and global slack adjustment; Monitors that track the CPU utilization of each processor; and Feedback scheduler that performs resource reclaimer recommended schedule adaptations dynamically. An end-to-end task model implemented by many distributed real-time applications is adopted. An application is comprised of m periodic tasks $\{T_i \mid 1 \leq i \leq m\}$ executing on n processors, and $m \geq n$ .Task T_i is composed of a chain of subtasks $\{T_{ij} | 1 \leq j \leq s_i\}$ located on different processors. The release of subtasks is subject to precedence constraints, i.e., subtask $T_{ij}(1 < j \leq s_i)$ cannot be released for execution until its predecessor subtask T_{ij}-1 is completed [13].

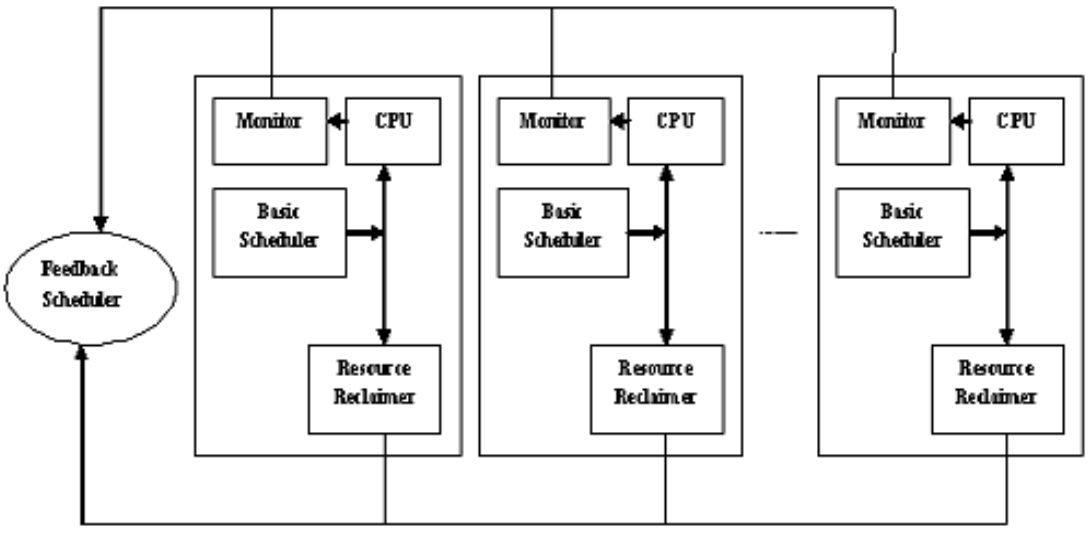

Figure 4: Feedback Control Scheduling for Embedded Distributed System

A processor in the distributed system receives subtasks of different tasks, arrived to the global Feedback scheduler. Within each processor the tasks are scheduled by FCFS method.

C. *Resource Reclaiming*

Resource reclaiming [14] refers to the problem of utilizing resources left unused by a task when it executes less than its wcet, because of data-dependent loops and conditional statements in the task code or architectural features of the system, such as cache hits and branch predictions, or both. Resource reclaiming is used to adapt dynamically to these unpredictable situations so as to improve the system's resource utilization and thereby improve its schedulability. The resource reclaiming algorithm used is a restriction vector (RV) based algorithm proposed in [15] for tasks having resource and precedence constraints. Two data structures namely restriction vector (RV) and completion bit matrix (CBM) are used in the RV algorithm. Each task T_i has an associated ncomponent vector, $RV_i[1 \ldots n]$, where n is the number of processors. $RV_i[j]$ for a task T_i contains the last task in $T<_i(j)$ that must be completed before the execution of T_i begins, where $T<_i(j)$ denotes the set of tasks assigned to processor P_j that are scheduled in feasible schedule (prerun schedule) to finish before T_i starts. CBM is an m X n Boolean matrix indicating whether a task has completed execution, where m is the number of tasks in the feasible schedule. The pseudo code for the resource reclaiming RV algorithm is given in Figure.

```
RV Algorithm()
{
   whenever a task Ti finishes execution on Processor Pj
         {
           Set CBM[I,j] to 1
           For all idle processors Pk
                 {
                   Let Tf be the first task in dispatch queue DQ[k].
                   For all the components of RVf, check CBM to see if the tasks in RVf have
                         finished execution
                   If all have finished execution
                     {
                         Start the execution of Tf on Pk.
                         Remove Tf from DQ[k].
                     }
                 }
         }
}
```

Figure 5: RV Resource Reclaiming Algorithm

D. *Resource Management*

A dynamic real-time system is composed of a variety of software components, as well as a variety of physical (hardware) components that govern the real time performance. The physical components of a real-time system can be described by a set of computational resources and an interconnection network, and other devices. The computational resources are a set of host computers $H = \{h_1 \ldots h_m\}$. It is generally assumed that the properties of the computational resources and the network resources are known. A hierarchical ARM architecture and a heuristic algorithmic approach based on a table lookup technique are proposed to solve the resource allocation problem.

- Adaptive resource management middleware (ARM) [16] optimizes the real-time performance of sets of application software. The middleware plans actions that include which software to run on which resources to achieve the maximum system level benefit.
- The ARM is responsible for the correct operation of the whole system. As input, it is given the static characteristics of both the hardware system and the software system. Based on these, it makes resource allocation decisions and has the ability to modify certain performance parameters such as service attributes.
- The resource manager consists of an allocation manager (AM) which chooses a new allocation of application software to hosts, due to major changes of extrinsic requirements, a global meta agent (GMA) which checks if reallocation of application software to hosts is necessary, and tries to optimize total benefit, meta agents (MA_i, i = 1, …, m)each being responsible for controlling an application subsystem (SS_i). One reason for including GMA is overall utility: each MA_i tries to optimize its own behavior.
- One of the main objectives is to find an optimal allocation of the applications to host computers. Both, execution time and memory usage of a task depend not only on extrinsic and service attribute parameters, but also on the machine on which the task is being executed.

E. Slack Management

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 through DVS in order to achieve the highest possible energy consumption. In energy efficient scheduling, the set of tasks will have certain deadline before which they should finish their execution and hence there is always a time gap between the actual execution time and the deadline. It is called slack time.

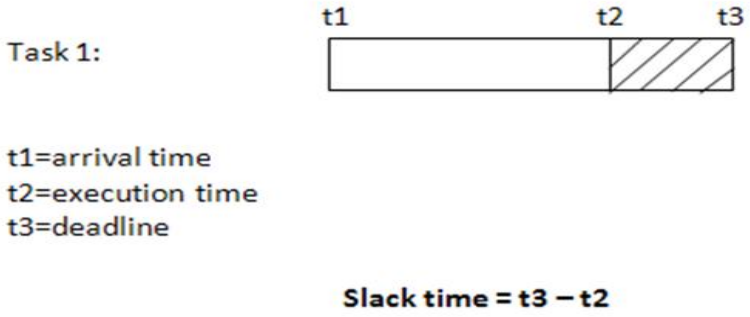

Figure 6: Slack Time

Conventional real-time systems are usually overestimated to schedule and provide resources using the wcet. In average case, real-time tasks rarely execute up to their worst case execution time (wcet). In many applications, actual case execution time (acet) is often a small fraction of their wcet. However, such slack times are only known at runtime through resource reclaimers. This slack is passed to schedulers to determine whether the next job should utilize the slack time or not.

The main challenge is to obtain and distribute the available slack in order to achieve the highest possible energy savings with minimum overhead. But most of these do not address dynamic task inputs. Only a few that attempt to handle dynamic task inputs assume no resource constraints among tasks. But in reality, few tasks need exclusive accesses to a resource. In exclusive mode no two real-time tasks are allowed to share a common resource. If a resource is accessed by a real-time task, it is not left free until the task's execution is completed. Other tasks in need of the same resource must wait until the resource gets freed. Our proposed algorithm handles this issue through the RV algorithm mentioned above.

Our slack management algorithm decides when and at which voltage should each task be executed in order to reduce the system's energy consumption while meeting the timing and other constraints. Our solution includes two phases: First we use static power management schemes based on wcet to statically assign a time slot to each task. Then we apply dynamic scheduling algorithm to further reduce energy consumption by exploiting the slack arising from the run-time execution time variation. Here a small amount of slack time called unit slack is added to all the tasks and finally we find the subset of tasks that can be allocated this slack time so that total energy consumption is minimized while the deadline constraint is also met.

F. Fault Tolerance

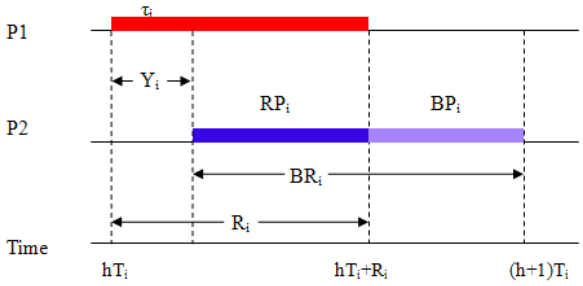

Figure 7: Illustration of deferred-active-backup copies when any failure occurs

Figure 7 shows that deferred-active-backup scheme delays the execution of β_i (backup copy) by Y_i time units. Let the WCRTs of τ_i (actual task) and β_i be R_i and BR_i, respectively. Note that BR_i is always larger than R_i, and

this fact is determined by our task assignment strategy. Divided by R_i, the execution of β_i is separated into two parts: BP and RP. RP executes in parallel with the primary copy, whereas BP is executed after its primary copy fails in producing correct results before the deadline. The implementation of deferred-active-backup scheme is challenging, because it has to precisely determine how much time the execution of active-backup copies should be delayed.
Again, we use Redundant (β_i) and Backup (β_i) to denote the execution time of RP and BP for β_i. Thus,

$$\text{Redundant}(\beta_i) = C_i(\text{execution time of } \tau_i) - \text{Backup}(\beta_i) \quad (4)$$

It should be noted that not all active-backup copies can be executed in schedules made by deferred-active-backup scheme. This is because, when a primary copy fails to produce correct results before its deadline, the recovery time of the primary copy may be occupied by copies of other tasks with higher priorities. This leads to an insufficient amount of time for the corresponding backup copy to be executed. We now introduce the deferred-active-backup copies, i.e., status (β_i) = deferred-active. The status of active-backup copies that cannot make use of the deferred-active technique are active, i.e., status (β_i) = active [5].

V. RESULTS AND DISCUSSIONS

For simulation, scheduling task sets and task graphs are generated using the following approach:

- Task sets are randomly generated with parameters such as arrival time, actual case execution time, worst case execution time and resource constraints
- Actual case execution time must be lower than the worst case execution time
- The overall deadline is generated such that it is always greater than or equal to the sum of all the worst case execution time
- Task graph is randomly generated using adjacency matrix where 0 represents the tasks that are not dependent on any other tasks and 1 represents the dependency

FCS algorithm dynamically utilizes the slack time in a most efficient way i.e., we reduce the speed of the processor and utilize the slack time completely which in turn reduces the power consumption of the system. It also considers the resource and precedence constraints with strict timing constraints. Thus this modified FCS for dynamic slack distribution has greater performance and energy efficiency against other existing slack distribution techniques.

In conventional systems the active backup starts along with the main task. If the main task is interrupted the backup task completes the execution. But it completes its execution before its deadline and leaves a certain amount of slack. In deferred-active-backup scheme the backup task is delayed and then started such that it completes the execution before its deadline but the slack time is zero. Thus the schedulability and reliability of the system is boosted when compared to the backup scheme used in conventional systems.

VI. CONCLUSION

Thus the modified Feedback Control Scheduling (FCS) algorithm with precedence and resource constraints act as a unified framework of adaptive real-time systems based on feedback control theory. The FCS framework supports fundamental resource scheduling solutions that provide robust performance guarantees for real-time systems. The modified FCS algorithm efficiently utilizes the slack dynamically. Thus the energy efficiency is increased compared with the conventional models. Our system uses deferred-active-backup scheme for fault tolerance. This scheme increases schedulability and reliability of the system when compared to the conventional backup schemes.

ACKNOWLEDGEMENT

The authors are grateful to the anonymous referees for their insightful suggestions and comments for improving the quality of this paper.

REFERENCES

[1] Rabi N. Mahapatra and Wei Zhao, "An Energy-Efficient Slack Distribution Technique for Multimode Distributed Real-Time Embedded Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 7, July 2005.
[2] Changjiu Xian, Yung-Hsiang Lu and Zhiyuan Li, "Dynamic Voltage Scaling for Multitasking Real-Time Systems with Uncertain Execution Times", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 8, August 2008.
[3] Subrata Acharya and Rabi N. Mahapatra, "A Dynamic Slack Management Technique for
Real-Time Distributed Embedded Systems", IEEE Transactions on Computers, vol. 57, no. 2, February 2008.
[4] Jaeyeon Kang and Sanjay Ranka, "DVS based Energy Minimization Algorithm for Parallel Machines", IEEE, 2008.

[5] Wei Luo, Xiao Qin, Xian-Chun Tan, Ke Qin, and Adam Manzanares, "Exploiting Redundancies to Enhance Schedulability in Fault-Tolerant and Real-Time Distributed Systems", IEEE Transactions on Systems, man, and Cybernetics—part a: systems and humans, vol. 39, no. 3, May 2009.
[6] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors", Proc. Int'l Symp. LowPower Electronics and Design, pp. 197-202, 1998.
[7] 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.
[8] L. Liestman and R. H. Campbell, "A fault-tolerant scheduling problem", IEEE Trans. Softw. Eng., vol. SE-12, no. 11, pp. 1089–1095, Nov. 1986.
[9] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems", IEEE Trans. Comput., vol. 53, no. 2, pp. 217–231, Feb. 2004.
[10] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Trans. Computers, vol. 39, no. 9, pp. 1175-1185, 1990.
[11] Peng Li, Haisang Wu, Binoy Ravindran and E. Douglas Jensen, "A Utility Accrual Scheduling Algorithm for Real-Time Activities with Mutual Exclusion Resource Constraints", IEEE TRANSACTIONS ON COMPUTERS, VOL. 55, NO. 4, APRIL 2006.
[12] A. Burchard, J. Liebeherr, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems", IEEE Trans. Comput., vol. 44, no. 12, pp. 1429–1443, Dec. 1995.
[13] Santhi Baskaran and P. Thambidurai, "Power Aware Scheduling for Resource Constrained Distributed Real-Time Systems", (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 05, 2010, 1746 1753.
[14] C. Shen, K. Ramamritham and J.A. Stankovic, " Resource reclaiming in multiprocessor real-time systems", IEEE Trans. Parallel and Distributed Systems, vol. 4, no. 4, pp. 382-397, Apr. 1993.
[15] G. Manimaran, C. Siva ram Murthy, Machiraju Vijay, and K. Ramamritham, "New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems", Journal of Parallel and Distributed Computing, vol. 44, no. 2, pp. 123-132, Aug. 1997.
[16] N. Shankaran, N. Roy, D. Schmidt, X. Koutsoukos, Y. Chen and C. Lu. "Design and Performance Evaluation of an Adaptive Resource Management Framework for Distributed Real-time and Embedded Systems", EURASIP Journal on Embedded Systems, 2008.