Power Aware Scheduling for Resource Constrained Distributed Real-Time Systems

Santhi Baskaran
Department of information Technology, Pondicherry Engineering College
Puducherry – 605 008, India

P. Thambidurai
Department of Computer Science and Engineering,
Pondicherry Engineering College
Puducherry – 605 008, India

***Abstract*—Power management has become popular in mobile computing as well as in server farms. Although a lot of work has been done to manage the energy consumption on uniprocessor real-time systems, there is less work done on their multicomputer counterparts. For a set of real-time tasks with precedence and resource constraints executing on a distributed system, we propose a dynamic slack management technique for feedback control scheduling (FCS) algorithm known as modified FCS algorithm. This algorithm schedules dependant periodic real-time task sets by effectively managing exclusive access resources with strict timing constraints along with energy efficiency. Simulation results show that, in comparison to commonly used greedy technique, the proposed technique achieves 28 percent less power consumption when validated with random task graphs.**

Keywords - Real-time; resource reclaiming; slack; precedence constraints; resource constraints.

I. INTRODUCTION

Energy consumption is an important design issue for battery-operated systems. In these systems, processor is the major energy consumer of energy. 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 [1], 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. Processors with the ability of dynamic voltage scaling are currently available [2] [3]. Traditional real-time scheduling theories depend on accurate a priori knowledge of the system workload, or worst-case execution times of tasks. Any variations in those values are undetected and unmanaged and they may lead to either under-utilization of system resources, which increases excessive system cost; or over-utilization of system resources when unpredictable events occur, which impairs greatly real-time performance of the system [4]. Therefore, it is necessary to introduce resource management mechanisms that can adapt to dynamic changes in resource availability and requirements. A promising solution is feedback control scheduling (FCS), which employs software feedback loops that dynamically control resource allocation in response to changes in input workload and resource availability.

A distributed real-time system consists of two or more devices that are collectively responsible for the execution of an application with specified requirements on performance and Quality of Service. The ready availability of inexpensive processors, large memory capacities and high bandwidth communication networks attracted the use of distributed systems for many of the real-time applications. Distributed real-time systems have emerged as a popular platform for applications such as multimedia, mobile computing and information appliances. An important problem that arises from using distributed systems is how to assign tasks and resources to the processors so as to fully utilize the processors, while ensuring that the timing constraints of the tasks are met along with energy efficiency. Hence in this paper we address energy efficiency in the context of homogeneous distributed real-time systems, targeting variable speed processor architectures.

The primary contributions of this work are four-fold:

1) We derive a dynamic model that describes distributed real-time systems with precedence and resource constraints,
2) We develop an modified feedback control scheduling algorithm with efficient resource reclaimer and resource manager,
3) We propose a dynamic distributed energy management technique which adapts and maintains end-to-end, latencies within specified timeliness requirements (deadlines) and enhances energy savings at the nodes, and
4) We present simulation results which demonstrate that the proposed scheduling algorithm can provide real-time performance guarantees efficiently along with energy savings.

This paper is organized in the following way. The related work is addressed in Section II. The application, system, power

and resource models are described in Section III. Proposed algorithm is described in Section IV. Section V discusses the simulation and analysis of results. Finally Section VI concludes this paper with future work.

II. RELATED WORK

Many hardware and software techniques have been proposed to reduce the energy consumption of such systems. The two most commonly used techniques are DVS [5] and Dynamic Power Management (DPM) [6]. The application of these energy management techniques can be exploited to the maximum if we can take advantage of all the idle time and slack time in between processor busy times. In a distributed system, if each node's DVS mechanism aggressively reduces the node's CPU speed, the induced latencies can cause an end-to-end deadline to be missed, or if DVS mechanisms are uncoordinated, they can prevent nodes where energy savings are most required from utilizing DVS, thereby limiting their energy-savings. Hence 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. The slack allocated to a node is based on the end-to-end latencies, its pay-off factor which represents the relative benefits or significance of conserving its energy, and the energy management objective of the system. The allocated slack is then used by each node's DVS mechanism to scale down its CPU frequency accordingly [7]. There has been an extensive study on low power scheduling of periodic tasks, aperiodic tasks, and their combinations on single processors [8], [9], [10], [11]. However, relatively little work has been done in the area of low-power scheduling for multiprocessors and distributed systems, assuming the tasks to be independent and known a priori. Power conscious algorithms for joint scheduling of periodic task graphs and aperiodic tasks are proposed [12], [13]. Recently, several DVS based algorithms for slack allocation have been proposed for tasks with precedence relationships in a multiprocessor real time system [14, 15, 16]. A linear programming (LP) based formulation is to used to solve the continuous voltage DVS [17], and extended to incorporate the communication time between tasks [18].

III. MODELS

In this section, we briefly discuss the system, application, power and resource models that we have used in our work.

A. System and Application Model

A distributed system with n homogeneous processors each with its private memory is considered for scheduling the given real-time application. 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 vertex v_i, we associate it a worst case execution time (wcet) at the reference (highest) voltage that can be obtained a priori by profiling. 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. We consider a distributed real-time application T with an end-to-end deadline is executed as a series of local tasks $\{T_1, ..., T_n\}$ on these PEs. 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

$$\sum_{i=1}^{n} d_i \le D$$

The latency at a node (source, sink, or intermediate) is expressed by L_{node} and includes delays such as for processing and communicating. Each L_{node} depends on the speed of the PE as determined by the DVS algorithm. A message from a parent node to child node experiences a transmission delay over each communication link represented by L_{link} that consists of propagation delays and delays for retransmissions or other error correcting measures, etc. The term slack (S) specifies by how much the sink node finishes processing before the end-to-end deadline (positive slack) or by how much the deadline is missed (negative slack). It can be expressed as:

$$S = D - L_{total}, \text{ where}$$

$$L_{total} = \sum_{i=1}^{n} Li_{node} + \sum_{j=1}^{n-1} L_{j\,link}$$

The proposed mechanism in this work adapts the local deadlines based on the allocated slack to affect the decisions made by the DVS algorithm. 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 $[S_i^-, S_i^+]$. 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

$$S_i^- = wcet_i$$

$$S_i^+ = D - \sum_{j=i+1}^{n} wcet_j$$

We also consider preemptive scheduling, but we consider no migration. For simplicity, we ignore overheads of speed adjustments and preemptions.

B. Power Model

The DVS technique reduces the dynamic power dissipation by dynamically scaling the supply voltage and the clock frequency of processors. The relationship between power dissipation P_d, supply voltage V_{dd}, and frequency f is represented by

$$P_d = C_{ef} \text{ X } V^2_{dd} \text{ X } f \text{ and}$$

$$f = k \text{ X } (V_{dd} - V_t{}^2)/V_{dd},$$

where C_{ef} is the switched capacitance, k is the constant of circuit, and V_t is the threshold voltage [19]. The energy consumed to execute task T_i, E_i, is expressed by $E_i = C_{ef}$ X V^2_{dd} X c_i, where c_i is the number of cycles to execute the task. The supply voltage can be reduced by decreasing the processor speed. It also reduces energy consumption of task. Here we use the task's execution time at the maximum supply voltage during assignment to guarantee deadline constraints.

C. Resource Model

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 [20].
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.

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. The scheduler uses the HoldTime information at run time to make scheduling decisions.

IV. PROPOSED ALGORITHM

The overall framework for the proposed algorithm is shown in Figure 1.

A. 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 feedback control scheduling. This algorithm can provide real-time performance guarantees efficiently, even in open environments. The feedback control scheduling algorithm framework has three components; *Monitors* that track the CPU utilization of each processor; *Resource reclaimer* that computes difference between task's actual execution time and worst case execution time for local and global slack adjustment; and *Feedback scheduler* that performs resource reclaimer recommended schedule adaptations dynamically. For this, an end-to-end task model implemented by many distributed real-time applications is adopted. An application is comprised of m periodic tasks $\{T_i \mid 1 \le i \le m\}$ executing on n processors, and $m \ge n$. Task T_i is composed of a chain of subtasks $\{T_{ij} \mid 1 \le j \le s_i\}$ located on different processors. The release of subtasks is subject to precedence constraints, i.e., subtask T_{ij} $(1 < j \le s_i)$ cannot be released for execution until its predecessor subtask T_{ij-1} is completed. The FCS architecture for distributed system is shown in Figure 2.

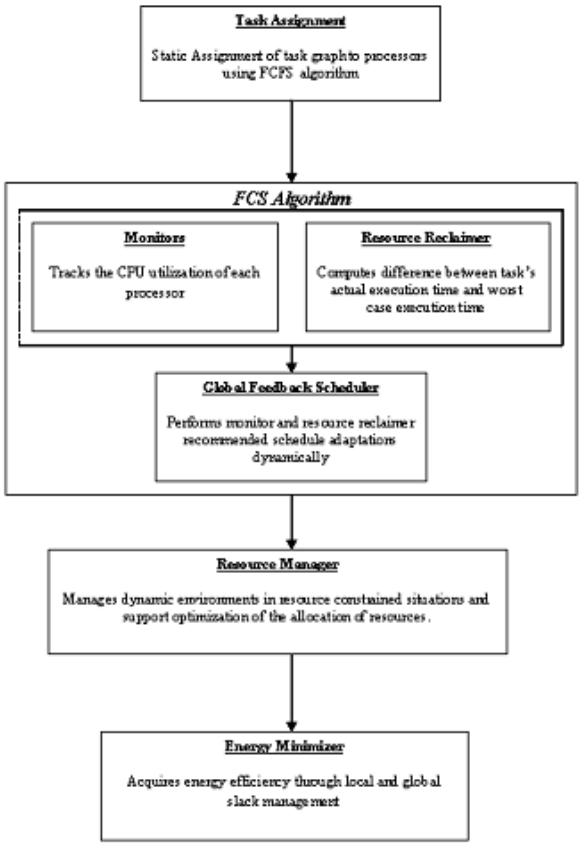

Figure 1 Overall Architecture of Energy-efficient FCS algorithm

Each task T_i is subject to an end-to-end relative deadline equal to its period. 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 the Basic scheduler using Earliest Deadline First algorithm (EDF), since EDF has been proven to be an optimal uniprocessor scheduling algorithm [21]. A task set is schedulable under EDF, if and only if it satisfies the condition that the total processor utilization U is less than 1. For a set of periodic real-time tasks $\{T_1, T_2, \ldots, T_m\}$, EDF schedulability criterion is expressed as:

$$U = \sum_{i=1}^{m} e_i / p_i \leq 1$$

where e_i is the worst case execution time and p_i is the period of the task T_i and m is the total number of tasks in the set. EDF algorithm allows dynamic rescheduling and preemption in the queues and processing nodes. In the scheme, the task sets are kept in the queue in the order of their approaching deadlines. The pseudo code for inserting task sets in EDF is given in Figure 3.

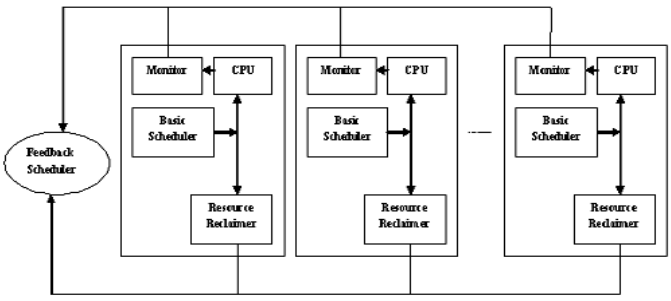

Figure 2. Feedback Control Scheduling for Distributed System

The Monitor module periodically checks the processor utilization and sends message to the global feedback scheduler, which on arrival of a new task decides to assign the task to a processor where the utilization criterion for the local EDF scheduler is still satisfied.

```
Insert (queue, t)   //inserts the task t into the EDF queue
 {
        foreach task ti in queue  //i = 1 to n (number of current tasks in the
queue)
            {
                If (t.deadline < ti.deadline)
                  {
                    queue_insert ( queue, t, ti)   //insert t before ti
                  }
            }
                If (not inserted) //all tasks in the queue have less deadline than
task t
                  {
                    queue_append ( queue, t)  //add t to the end of queue
                  }
 }
```

Figure 3. EDF Pseudo Code

Resource reclaiming [22] 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 [23] 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 n-component 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 4.

```
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 4. RV Resource Reclaiming Algorithm

B. 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. It is generally assumed that the properties of the computational resources, the network resources and other resources are known.

The resource manager plans actions that include which software to use which resources to achieve the maximum system level benefit and to optimizes the real-time performance of sets of application software. 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.

C. *Slack Management through DVS*

Dynamic voltage scaling and dynamic frequency scaling (DFS) allow adjusting processor voltage and frequency at runtime. Usually, higher processor voltage and frequency leads to higher system throughput while energy reduction can be obtained using lower voltage and frequency. Recent trends in modern processor architecture provide support for these two mechanisms. For example, Intel Mobile Processors with SpeedStep technology [24] and AMD Mobile Processors with PowerNow! Technology [25], etc. Instead of lowering processor voltage and frequency as much as possible, energy-efficient real-time scheduling adjusts voltage and frequency according to some optimization criteria, such as low energy consumption or high throughput, while still meeting the timing constraints of real-time tasks. However, reduction of processor voltage and frequency increases the circuit delay, causing slowdown in the execution of programs. Hence, power-aware real-time scheduling makes a trade off between energy saving and system performance. It is required to schedule properly where to change the processor voltage and frequency to have the best energy-efficient performance.

The real-time scheduling problem with power optimization constraints is NP-hard [26]. It is time consuming to find an optimal schedule where energy consumption is minimized and all timing constraints along with precedence and resource constraints are met. Many previous works either proposed offline scheduling for large energy reduction, or used heuristic methods to reduce scheduling overhead. However, while the former approaches are inflexible and too costly to store in memory, the latter ones may not realize the full potential of energy savings.

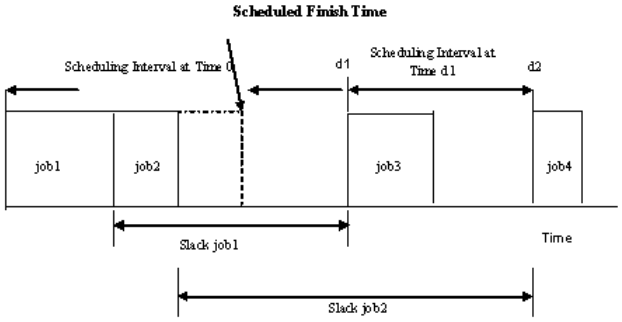

Figure 5. System Slack Times

As shown in Figure 5, the slack of a task with deadline at d_i at any time $t < d_i$ is equal to $(d_i - t)$ minus the time required to complete the remaining portion of the task. 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 wcet. In many applications, actual execution time (aet) 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.

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. The main challenge is to obtain and distribute the available slack in order to achieve the highest possible energy savings with minimum overhead. Many energy-efficient slack management techniques were designed in the literature. 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.

Hence, given a periodic task graph, with all the tasks assigned to processors, 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.

V. SIMULATION AND ANALYSIS OF RESULTS

A simulator was developed to simulate voltage scalable processor which dynamically adjusts the processor speed according to the proposed algorithm, given input of a hardware profile and workload. The hardware profile describes the multiple operating points of the processor, each with a separate frequency and power consumption. Unless otherwise specified, the following hardware profile {(1.6, 1.484), (1.4, 1.420), (1.2, 1.276), (1.0, 1.164), (800, 1.036), (600, 0.956)}, is used where 1.6, 1.4, 1.2, 1.0, 800, 600 denote the processor frequency in GHz and MHz, and 1.484, 1.420, 1.276, 1.164, 1.036, 0.956 denote the corresponding voltage.

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 execution time, worst case execution time and resource constraints.
- The actual execution time is taken randomly and worst case execution time is also randomly generated such that it is always greater than the actual execution time.
- The overall deadline is generated such that it is always greater than or equal to the sum of all the actual 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, with varying breadth and depth.

It is necessary to compare the performance of the proposed slack distribution technique mentioned as modified FCS against other existing slack distribution techniques. For our study, we have considered the most commonly used slack management scheme Greedy slack management with other components of the algorithm remaining the same. This algorithm is denoted as FCS. The performance metric used for comparison is percentage of power consumption. For each set of tasks the number of processors is kept constant and the energy consumption for a minimum of ten DAGs are noted. The average values of all those DAGs were calculated. Each point in the above graphs is the average value of such DAGs. This method was repeated by changing the number of processors (varied between 2 to 10) and comparisons were made between the existing and proposed algorithms. A few sample results of those comparisons are shown in the graphs below. It can be seen from the results that the proposed algorithm had less power consumption than the existing one, thus leading to more energy efficiency.

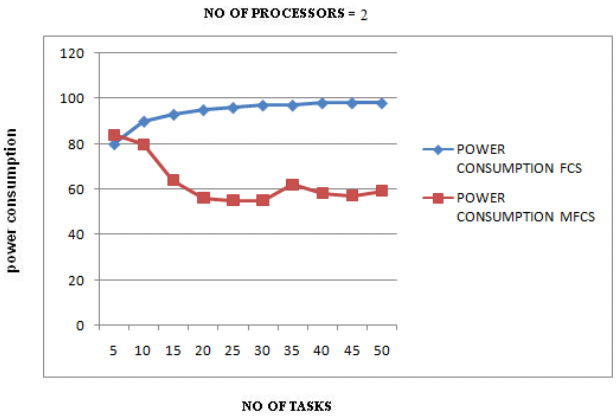

Figure 6. Power Cosumption of FCS and Modified FCS for n = 2

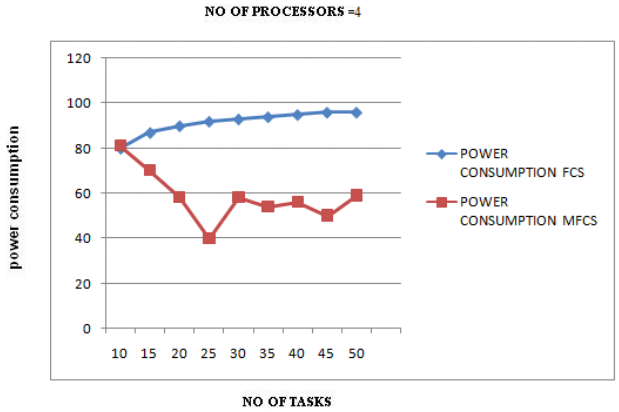

Figure 7. Power Cosumption of FCS and Modified FCS for n = 4

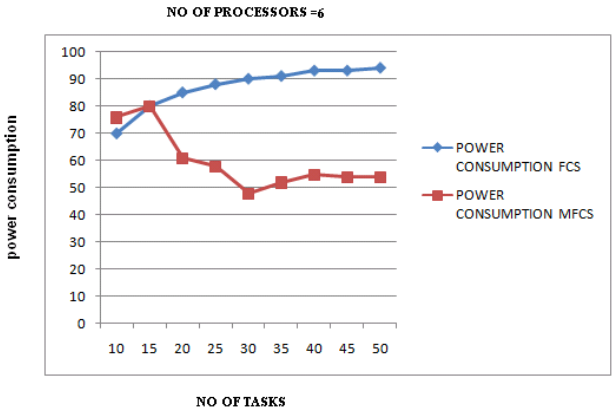

Figure 8. Power Cosumption of FCS and Modified FCS for n = 6

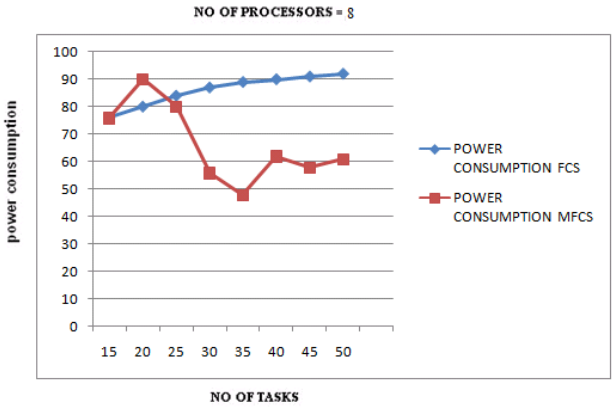

Figure 9. Power Cosumption of FCS and Modified FCS for n = 8

From the graphs it is inferred that the proposed algorithm yield more energy savings than the existing algorithm. The

modified algorithm outperform existing one by reducing on an average 28 percent of power consumption. This is due to the addition of resource reclaiming technique in the proposed algorithm.

TABLE I. AVERAGE POWER CONSUMPTION OF ALGORITHMS

No. of Processors	FCS	Modified FCS
2	94	63
3	91	57
4	91	58
5	89	61
6	87	59
7	87	65
8	86	66

From Table 1, it is noted that the proposed algorithm consumes 28 percent less power on an average than the existing one. This is due to the concept of unit slack concept introduced dynamically.

VI. CONCLUSION AND FUTURE WORK

In this work, an energy efficient real-time scheduling algorithm for distributed systems is presented. This scheduling algorithm is capable of handling task graphs with precedence and resource constraints in addition to timing constraints. The major contribution of this work is the development of modified feedback control scheduling algorithm with efficient resource manager, resource reclaimer and dynamic slack distribution technique. The static power reduction component determines the minimum voltage at which each tasks can be run. It also exploits the idle intervals by putting the processor in power down modes to reduce the power consumption. Simulation results show almost 28 percent less power is consumed by the proposed algorithm compared with the existing commonly used greedy algorithm.

For future work, security and fault tolerant issues can be considered for such real-time distributed systems. Another important future work is extending this technique for a heterogeneous distributed real-time system.

REFERENCES

[1] T. D. Burd, T. Pering, A. Stratakos, and R. Brodersen, A dynamic voltage scaled microprocessor system", IEEE J. Solid-State Circuits, Vol. 35, pp. 1571-1580, 2000.
[2] http://developer.intel.com/design/intelxscale
[3] http://www.transmeta.com
[4] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, Feedback Control Real- Time Scheduling: Framework, Modeling, and Algorithms. Real-Time Systems Journal, Special Issue on Control-theoretical Approaches to Real-Time Computing, 23(1/2): 85-126, 2002.
[5] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," In Proc. International Symposium on Low Power Electronics and Design, pp. 197-202, 1998.
[6] 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.
[7] Dinesh Rajan, Christian Poellabauer, Andrew Blanford, and Bren Mochocki, "Cooperative Dynamic Voltage Scaling using Selective Slack Distribution in Distributed Real-Time Systems,"
[8] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," Proc. Design Automation Conf., pp. 134-139, 1999.
[9] Y. Shin, K. Choi, and T. Sakurai, "Power Optimization of Real- Time Embedded Systems on Variable Speed Processors," Proc. Int'l Conf. Computer-Aided Design, pp. 365-368, 2000.
[10] G. Quan and X. Hu, "Energy Efficient Fixed Priority Scheduling for Real-Time Systems on Variable Voltage Processors," Proc. Design Automation Conf., pp. 828-833, June 2001.
[11] P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," Proc. ACM Symp. Operating Systems Principles, Oct. 2001.
[12] J. Luo and N.K. Jha, "Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems," Proc. Asia and South Pacific Design Automation Conf., Jan. 2002.
[13] J. Luo and N.K. Jha, "Power-Profile Driven Variable Voltage Scaling for Heterogeneous Distributed Real-Time Embedded Systems," Proc. VLSI Design Conf., Jan. 2003.
[14] J. Luo and N. K. Jha, Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems, Int. Conf. on Computer-Aided Design, Nov. 2000, pp. 357-364.
[15] J. Luo and N. K. Jha, Power-profile Driven Variable Voltage Scaling for Heterogeneous Distributed Real-time Embedded Systems, Int. Conf. on VLSI Design, Jan. 2003, pp. 369-375.
[16] M. T. Schmitz and B. M. Al-Hashimi, Considering Power Variations of DVS Processing Elements for Energy Minimization in Distributed Systems, Int. Sym. on System Synthesis, Oct. 2001, pp. 250-255.
[17] Y. Zhang, X. (Sharon) Hu, and D. Z. Chen, Task Scheduling and Voltage Selection for Energy Minimization, Design Automation Conf., June 2002, pp. 183-188.
[18] Jaeyeon Kang and Sanjay Ranka, "DVS based Energy Minimization Algorithm for Parallel Machines,"
[19] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, Low-Power CMOS Digital Design, IEEE Journal of Solid- State Circuits, 27(4), Apr. 1992, pp. 473-484.
[20] 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.
[21] C.M. Krishna and Shin K. G., Real-Time Systems, Tata McGraw-Hill, 1997.
[22] 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.
[23] 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.
[24] Intel internet homepage, http://www.intel.com/index.htm, 2004.
[25] AMD internet homepage, http://www.amd.com/us-en, 2004.
[26] K. Chen and P. Muhlethaler, A scheduling algorithm for tasks described by time value function, Journal of Real-Time Systems, 10(3):293–312, May 1996.

AUTHORS PROFILE

Mrs. Santhi Baskaran received her B.E. degree in Computer Science and Engineering from University of Madras, Chennai, India in 1989 and M.Tech. degree in Computer Science and Engineering from Pondicherry University, Puducherry, India in 1998. She served as Senior Lecturer and Head of the Computer Technology Department, in the Polytechnic

Colleges, Puducherry. India, for eleven years, since 1989. She joined Pondicherry Engineering College, Puducherry, India in 2000 and currently working as Associate Professor in the Department of information Technology. Now she is pursuing her PhD degree in Computer Science and Engineering. Her areas of interest include Real-time systems, embedded systems and operating systems. She has published research papers in International and National Conferences. She is a Life member of Indian Society for Technical Education and Computer Society of India.

Prof. Dr. P. Thambidurai is a Member of IEEE Computer Society. He received his PhD degree in Computer science from the Alagappa University, Karaikudi, India in 1995. From 1999, he served as Professor and Head of the Department of Computer Science & Engineering and Information Technology, Pondicherry Engineering College, Puducherry, India, till August 2006. Now he is the Principal for Perunthalaivar Kamarajar Institute of Engineering and Technology (PKIET) an Government institute at Karaikal, India. His areas of interest include Natural Language Processing, Data Compression and Real-time systems. He has published over 50 research papers in International Journals and Conferences. He is a Fellow of Institution of Engineers (India). He is a Life member of Indian Society for Technical Education and Computer Society of India. He served as Chairman of Computer Society of India, Pondicherry Chapter for two years. Prof. P.Thambidurai is serving as an Expert member to All India Council for Technical Education (AICTE) and an Adviser to Union Public Service Commission (UPSC), Govt. of India. He is also an Expert Member of IT Task Force and Implementation of e-Governance in the UT of Puducherry.