

Cooperative Dynamic Voltage Scaling using Selective Slack Distribution in Distributed Real-Time Systems

Dinesh Rajan, Christian Poellabauer, Andrew Blanford, and Bren Mochocki

Department of Computer Science and Engineering

University of Notre Dame

Notre Dame, IN 46556

{dpandiar,cpoellab,ablanfor,bmochock}@cse.nd.edu

Abstract—This work is based on the observation that existing energy management techniques for mobile devices, such as dynamic voltage scaling (DVS), are non-cooperative in the sense that they reduce the energy consumption of a single device, disregarding potential consequences for other constraints (e.g., end-to-end deadlines) and/or other devices (e.g., energy consumption on neighboring devices). This paper argues that energy management in distributed real-time systems has to be end-to-end in nature, requiring a coordinated approach among communicating devices. A cooperative distributed energy management technique (Co-DVS) is proposed that i) adapts and maintains end-to-end latencies within specified timeliness requirements (deadlines) and ii) enhances energy savings at the nodes with the highest *pay-off factors* that represent the relative benefits or significance of conserving energy at a node. The proposed technique employs a *feedback-based approach* to dynamically distribute end-to-end slack among the devices based on their pay-off factors.

I. INTRODUCTION

The increased capabilities of mobile and embedded devices and the trend toward communication-intensive distributed applications have led to increased energy demands and shorter battery life times. As a consequence, energy-awareness has become a critical factor in system design and application development. In addition, timeliness of task execution and inter-task communication is essential to distributed applications such as video conferencing, mobile robot teams, or sensor networks. Energy management techniques for real-time systems exploit periods of inactivity in the corresponding resources/devices to either switch to lower operating levels (e.g., processor frequency) or power them down (e.g., network interface). The term ‘*slack*’ is typically used to refer to such idle or inactive periods in a resource. For a processor, the slack summarizes the idle intervals arising from tasks completing execution before their specified deadlines. A number of efficient energy management techniques that utilize slack for maximizing energy savings in real-time systems have been studied and explored. A popular approach to energy conservation by eliminating slack is to reduce speed and core voltage of embedded processors whenever the system is not used to capacity. This approach, called *DVS (Dynamic Voltage Scaling)*, has been explored in previous work [9], [11], typically with the goal to meet deadlines of real-time tasks while minimizing the energy consumption of embedded or mobile systems [14]. DVS takes advantage of the quadratic relationship between supply voltage and energy consumption [1], which can result in significant energy savings. Other successfully deployed

energy management techniques in mobile systems include using low-power modes for resources such as disk drives during idle periods [3] and reducing energy overheads of wireless communications [6].

While such techniques successfully reduce the energy costs of an individual mobile or embedded device, these techniques are highly non-cooperative (or ‘selfish’) in a distributed system with energy and real-time constraints. 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 devices that comprise such a distributed system can be homogeneous and powered by a common energy source such as in a mobile robot with multiple independent computing devices. A distributed system can also consist of several heterogeneous devices that are equipped with individual energy sources, e.g., a sensor network consisting of several nodes that are each provided with separate battery resources. Existing energy management techniques when applied to the individual resources in any such distributed system conserve energy aggressively at those resources irrespective of the costs incurred. In a distributed setting, ‘cost’ refers to the violation of other constraints (e.g., end-to-end deadline misses) and/or adverse effects on the energy consumption of other devices.

A. Effects on timeliness

As a generalized example of a distributed real-time system, consider two communicating devices - a source device streaming media content to a sink. The video and audio streams exchanged between these devices are typical examples of *soft real-time communications*, i.e., while latencies and jitter need to be controlled to allow for an acceptable video replay and understandable voice playback, infrequent deadline misses can be allowed. Each device, if equipped with an energy saving technique such as DVS, will try to maximize its own battery life time. However, it does so unaware of the potential adverse effects this may have on end-to-end real-time constraints.

This is visualized in Figure 1, where the horizontal bars show the execution times (or latencies) of the media stream at both the source and the sink devices. The y-axis represents the CPU clock frequency used by the DVS algorithm on the source side and each bar is annotated by the clock frequency of the sink. Assume these two devices support six discrete frequency levels (ranging from 0.6 to 1.6GHz). The use of the DVS

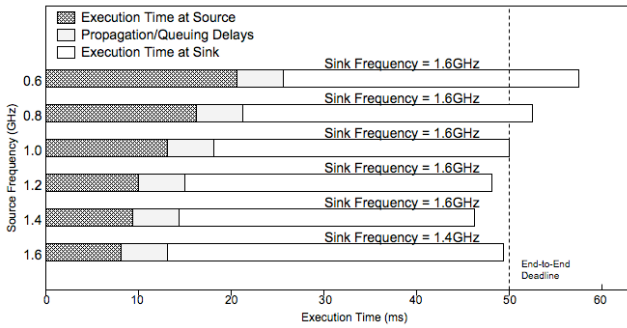


Fig. 1. DVS induced latencies.

algorithm at the source introduces latencies into computation and communication, e.g., the processing steps to capture and compress an image and the communication steps to traverse the protocol stack, and to fragment and transmit the image are prolonged at lower clock frequencies. The images arrive at the sink with increased latencies, putting more stringent constraints on the replay requirements of the sink device, i.e., the remaining time until the sink’s replay deadline is shorter compared to the case where the source device would not have used DVS. For example, the top bar shows the scenario where the source selects (based on its workload and DVS algorithm) the lowest possible speed supported by the device (0.6GHz). This results in increased latencies at the source, causing the end-to-end deadline to be missed at the sink (indicated by the vertical dashed line), even though the sink executes at its highest speed (1.6GHz). This leads to an important insight: *a resource adaptation on one device can cause end-to-end real-time constraints to be violated on other devices* [7].

In this scenario, the source needs to execute at 1.0GHz or higher to satisfy the end-to-end deadline for the stream.

B. Effects on energy constraints of other devices

If the two devices are part of a distributed system supported by a common energy source (e.g., a mobile robot), the DVS algorithms on both devices need to be coordinated to maximize global energy savings and to meet the end-to-end deadlines. In such a case, the source needs to execute at 1.0GHz to reduce the total energy drawn from the common energy-source compared to the other higher frequency settings. However, to account for the dynamic variations in latencies (e.g., bus-contention latencies can significantly affect transmission times [8]) and workload in these devices, it is beneficial to execute the upstream devices at a higher frequency. For example, in Figure 1, this translates to the setting in the bottom bar, where the source executes at its highest frequency thereby allowing flexibility for frequency scaling at the sink. Such a setting allows the sink’s DVS algorithm to quickly adapt to higher frequencies to meet end-to-end deadlines whenever the latencies incurred by the stream dynamically vary and increase prior to their arrival at the sink. The downside to this approach (i.e., executing the upstream devices at higher frequencies) is that the total energy savings in the system can be sub-optimal.

Distributed environments with heterogeneous devices supported by their own energy sources such as sensor networks

need to be addressed differently. In such systems, the energy needs of the devices can vary significantly based on their available or remaining energy levels. For the same example studied previously, assume the two communicating devices as part of a sensor network. Consider these devices to be equipped with independent energy resources (e.g., separate batteries). For the scenario shown in Figure 1, assume that the sink node has a battery charge level of about 10%, while the source has a full battery charge. As discussed earlier, the use of DVS at the source node introduces additional latencies which may result in the sink node not being able to (fully) exploit its own DVS capabilities, therefore running at a higher speed than required to satisfy the end-to-end deadline. Moreover, executing the sink at a higher speed would be disadvantageous given its low energy level relative to the source. In such a case, the source node would need to execute at a higher frequency allowing the sink to reduce its own speed, thereby decreasing its energy consumption, while still meeting the end-to-end deadline. The bottom bar in Figure 1 visualizes this setting. This leads to an important insight in distributed systems with individual energy resources: *aggressive resource adaptation on one device can limit the benefits achievable by adaptation mechanisms on other devices*.

To summarize, in systems where the source is the more critical node (in terms of energy), the third bar from the top would be the most preferable, i.e., the source runs at the lowest possible speed (1.0GHz) that ensures that the end-to-end deadline is met, while the sink selects its highest speed. In a system where the sink is more critical, the source needs to execute at its highest speed allowing the sink to run at a lower speed (1.4GHz).

C. Cooperative energy management goals

Based on these observations, resource adaptation techniques for energy management in distributed real-time systems need to be coordinated to meet global energy and real-time requirements. For the rest of this work, the focus is on the commonly adapted energy management technique of DVS.

Summarizing the insights from the scenario given in Figure 1, the need for cooperative DVS mechanisms is twofold:

- 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;
- if DVS mechanisms are uncoordinated, they can prevent nodes where energy savings are most required from utilizing DVS, thereby limiting their energy-savings.

This paper introduces an adaptive mechanism for distributed settings where a source and a sink (and any nodes in between) coordinate their DVS adaptations to achieve these goals. The presented approach is 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.

II. SYSTEM MODEL

This section introduces the notations used in this work and describes the system under consideration. We consider a distributed setting of n nodes where a global task T with an end-to-end deadline is executed as a series of local tasks $\{T_1, \dots, T_n\}$ on these nodes. The driving scenarios for such settings include streaming applications such as mobile video-conferencing tools, wireless video sensors, or any other scenario where periodic tasks on two or more devices exchange streams of data. A communication stream can be considered as a sequence of packets, where each packet traverses a number of steps, from generation at the source, to transmission, to queuing (e.g., at forwarders), to final processing at the sink. The forwarders are intermediate nodes that route traffic between two communicating nodes in a distributed environment.

Each packet of the stream has an *end-to-end deadline* D , by which the packet has to be received and processed at the *sink* node. Each local task T_i executed at node i has a specified local deadline d_i by which the packet has to be processed or transmitted. The local deadlines assigned must satisfy

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

To simplify discussion, we refer to the deadline d_1 of the (stream-generating) task at the source as d_{source} , the deadline d_n of the (processing) task at the sink as d_{sink} and any intermediate deadlines d_j (where $1 < j < n$) at the forwarding nodes as $d_{forward}$. Initially, local deadlines for each of the nodes are assigned by equally splitting the end-to-end deadline D , i.e., $d_{source} = d_{forward} = d_{sink} = D/n$.

The overall latencies incurred by the stream of packets as they traverse through the various nodes before arriving at their final destination (sink) can be summarized by their end-to-end latency L (or end-to-end delay) formulated as

$$L_{total} = \sum_{i=1}^n L_{node}^i + \sum_{j=1}^{n-1} L_{link}^j \quad (2)$$

where n indicates the number of nodes traversed by the packet.

The latency at a node (source, sink, or forwarder) is expressed by L_{node} and includes delays such as for processing and queuing. Each L_{node} depends on the CPU speed of the node, as determined by the DVS algorithm. A packet experiences a transmission delay over each link represented by L_{link} that consists of propagation delays and delays for retransmissions or other error correcting measures, etc.

For the rest of this paper, the term *slack* (σ) specifies by how much the sink finishes packet processing before the end-to-end deadline (positive slack) or by how much the deadline is missed (negative slack). It can be expressed as:

$$\sigma = D - L_{total} \quad (3)$$

The proposed mechanism in this work adapts the local deadlines based on the allocated slack to affect the decisions made by the DVS algorithm. It is important to note that our mechanisms influence the CPU frequencies selected by the

DVS algorithm without introducing any changes to the DVS algorithm. That is, 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 $[\delta_i^-, \delta_i^+]$. The values for δ 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

$$\delta_i^- = wcet_i \quad (4)$$

$$\delta_i^+ = D - \sum_{j=i+1}^n wcet_j \quad (5)$$

The DVS algorithm employed at each node in this work is based on the *look-ahead EDF (laEDF)* algorithm [9]. This algorithm combines EDF scheduling with an aggressive voltage scaling approach. It employs a look-ahead technique to defer as much work as possible and the CPU frequency is recalculated whenever a task finishes before its worst-case execution time. The proposed mechanism in this work could be applied to other scheduling and DVS algorithms as well.

We assume that the processor on each node in the system has a finite set of operating speeds denoted by a tuple $\{f_i, V_i\}$, where f_i is the operating frequency and V_i is the operating voltage. We ignore the *transition overheads*, i.e., time and energy costs for switching between speed levels, in this work.

Pay-off factors. As described earlier, the proposed mechanism in this work allocates slack (σ) to the nodes in a heterogeneous system based on their pay-off factors. The pay-off factor of a node represents the importance or benefits of allocating σ (or a proportion) to it. The pay-off factors can be chosen based on a single parameter or a combination of factors that represent the energy levels, architecture, or the energy management goals of the system. Previous work has used pay-off factors to represent the requirements of an application in a networked environment [5]. The pay-off factors in this work were assigned based on the remaining battery charge levels of the respective nodes. In this case, the node with the lowest battery level, referred as the *critical node*, was assigned the highest pay-off factor. This results in the critical node receiving the highest priority during the slack allocation procedure thereby enhancing the energy savings achieved at this node. It is important to note that the proposed mechanism can easily adapt to varying pay-off factors and any resulting changes in the node priorities. The pay-off factors can also be used to represent the significance of a node in the system (e.g., a forwarder node can be assigned the highest pay-off factor to indicate its significance to communications in the entire system) or the power profile and workload of a node.

III. COOPERATIVE DYNAMIC VOLTAGE SCALING

This section describes the Cooperative DVS mechanism (Co-DVS) for end-to-end energy and latency management and introduces the concept of energy management directives.

A. Motivation

In streaming wireless applications (e.g., mobile multimedia), both energy conservation and timeliness are essential. To motivate our approach for achieving these goals, consider the settings presented in Figure 1. As described earlier, if the source-based DVS can be coordinated such that it increases its speed, the sink will be able to meet the end-to-end deadline for the packet stream. Such coordination among the individual DVS algorithms at each of the devices is achieved by influencing the local deadline of the real-time packet handling tasks executed at those devices. The modified local deadline on each device affects the scheduling/DVS decisions of laEDF and thereby the speed of the device and the induced latencies. As an example, if the local deadline d_{source} for the packet generating task at the source can be reduced such that it scales the source frequency to 1.0GHz, then the end-to-end deadline for the packet is satisfied. This is visualized by the third bar from the top in Figure 1 where the local deadline at the source is reduced from its initial value (i.e., $D/2$) to increase its speed thereby meeting the end-to-end deadline.

Further, in distributed systems involving multiple heterogeneous nodes with individual energy sources, the energy needs of the nodes vary significantly with their available battery charge levels. As seen earlier, for the case when the sink battery charge is much lower, it needs to execute at a lower speed and conserve more energy while meeting the deadline constraints. This can be achieved by further lowering d_{source} which causes the packet generating task to be executed at a higher speed (shown by the bottom bar in Figure 1) and reduces the latencies incurred at the source.

The proposed mechanism adapts the deadline of the local task i of the distributed application based on the slack allocated (σ_i) to that node. That is, the deadline of local task i is recomputed as $d_i = d_i + \sigma_i$. A positive allotted slack results in the local deadline being increased while a deadline miss (or negative slack) leads to the deadline being lowered. The local deadlines at each node are always adjusted in compliance with the specified bounds given by Equations 4 & 5.

B. Energy Management Directives

Given a single mobile device, the goal of energy management techniques such as DVS is to reduce the energy consumption of the device. However, in a distributed system, different goals can be desired depending on the system framework (e.g., homogeneous vs. heterogeneous devices):

- **System-wide energy conservation.** The distributed components of common embedded systems such as mobile robots, mobile data-logging units, etc., are supported by a common energy source. Here, the goal is to delay the depletion of this energy source. This is achieved by conserving energy on a system-wide level that can be expressed by the directive: *reduce the sum of the energies consumed at all nodes.*
- **Node-specific energy conservation.** Other distributed systems, such as mobile ad-hoc networks and sensor networks, depend on individual, limited energy sources

on each node. Here, the goal is to focus energy conservation on specific *critical* nodes, e.g., nodes that serve as forwarders or gateways for other nodes or nodes whose energy sources are almost depleted. This goal can be expressed by the directive: *limit the DVS capabilities of non-critical nodes in favor of critical nodes.*

Therefore, *energy management directives* need to be specified as part of a distributed energy management algorithm. The following sections describe the proposed mechanism based on **selective slack allocation** that adapts the local deadlines in accordance with the specified energy management directives. The slack is allocated to each of the nodes based on *feedback* of end-to-end latencies and the pay-off factors.

IV. SELECTIVE SLACK DISTRIBUTION

This section introduces the selective slack distribution algorithm for achieving cooperative dynamic voltage scaling in a distributed environment. The proposed selective distribution algorithm distributes the slack σ in the system to the nodes in order of their position in the communication path (system-wide energy conservation) or their pay-off factors (node-specific energy conservation). For example, if the slack is positive, the algorithm would distribute this slack to the downstream node (or node with the highest pay-off factor) and any remaining slack is then distributed to the next upstream node (or the node with the next highest pay-off factor). This contrasts with the approach employed in [7] where the slack is distributed to every node as a proportion of their current operating frequencies. Since only a limited number of discrete CPU frequencies are supported in common mobile processors, distributing larger amounts of slack to individual nodes is more likely to result in frequency adaptations as compared to such proportional slack distribution mechanisms.

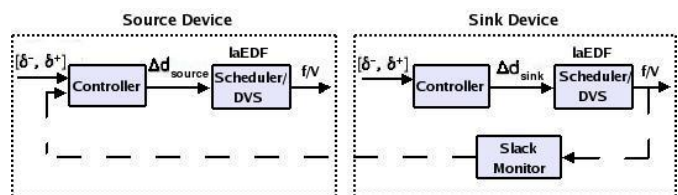


Fig. 2. Feedback loop for selective slack distribution in $Co-DVS_{system}$.

A. System-wide energy conservation mechanism

For a distributed embedded system consisting of homogeneous devices powered with a common energy source, the directive is to achieve system-wide energy savings while satisfying end-to-end timeliness requirements. To achieve this goal, a simple slack distribution algorithm ($Co-DVS_{system}$) is presented in this section.

The $Co-DVS_{system}$ slack distribution algorithm (Algorithm 1) *resides on each node* as part of the controller in the model illustrated in Figure 2. The controller is used to receive the feedback on slack and adapt the local deadline accordingly at that corresponding node. Any positive slack (σ) is utilized in the system starting from the downstream nodes

Algorithm 1 Pseudocode for $Co-DVS_{system}$

Require: Measured slack σ

```
1: upon end-to-end deadline miss or expiry of feedback interval  $\tau$ :
2:   measure and update  $\sigma$ 
3:   distribute( $n, \sigma$ )

4: distribute(node  $i, \sigma$ ):
5:   if  $\sigma > 0$  and  $freq_i > freq_{min}$  then
6:     if  $(\sigma - \Delta\delta_i^+) > 0$  then
7:        $d_i = d_i + \Delta\delta_i^+$ 
8:       distribute( $i - 1, \sigma - \Delta\delta_i^+$ )
9:     else
10:       $d_i = d_i + \sigma$ 
11:    end if
12:  else if  $\sigma < 0$  and  $freq_i < freq_{max}$  then
13:    if  $(\sigma + \Delta\delta_i^-) < 0$  then
14:       $d_i = d_i - \Delta\delta_i^-$ 
15:      distribute( $i - 1, \sigma + \Delta\delta_i^-$ )
16:    else
17:       $d_i = d_i + \sigma$ 
18:    end if
19:  else
20:    distribute( $i - 1, \sigma$ )
21: end if
```

(i.e., starting from the sink) by increasing their local deadlines thereby allowing them to run at lower frequencies (lines 5 to 11). The amount of positive slack utilized at each node is bounded by the extent to which it can increase its deadline. This is given by the parameter,

$$\Delta\delta_i^+ = \delta_i^+ - d_i \quad (6)$$

Similarly, deadline misses are handled by distributing the negative slack among the nodes starting from the downstream nodes (lines 12 to 18). In such a case, the downstream nodes adapt to the deadline misses by lowering their local deadlines causing them to execute at higher frequencies. The amount of slack distributed to a node in this case is bounded by:

$$\Delta\delta_i^- = d_i - \delta_i^- \quad (7)$$

After a downstream node has changed its operating frequency to the lowest (highest) available frequency in the event of positive (negative) slack, any remaining slack is fed to the next preceding upstream node (lines 8, 15, 20). Such an arrangement of the downstream nodes lowering their frequencies before the upstream nodes allows the system to react faster to dynamic variations in latencies and system load. For instance, if the bus contention delays increase significantly, the downstream nodes can scale their frequencies accordingly to keep the incurred latencies within the specified constraints.

B. Node-specific energy conservation mechanism

In systems such as sensor networks that consist of nodes powered by individual energy sources, the available energy levels and needs can vary drastically between the various nodes. Thus it is imperative to identify the energy-critical nodes and favor higher energy conservation at such nodes to achieve prolonged life-times. In this work, the pay-off factors are assigned and continually updated based on the remaining battery levels of the respective nodes.

Algorithm 2 Pseudocode for $Co-DVS_{node}$

Require: Slack σ , ordered list of nodes Q arranged from critical (highest pay-off) to least critical (lowest pay-off).

```
1: Initialize():
2:    $node_{critical} = Q[1]$  and  $node_{non-critical} = Q[n]$ 
3:    $\alpha = \Delta\delta^+(node_{critical})$  and  $\beta = \Delta\delta^-(node_{non-critical})$ 

4: remove-update(List  $Q$ , Node  $i$ ):
5:    $\{Q\} = \{Q\} - i$ 
6:   Re-order  $Q$ . sizeof( $Q$ ),  $n = 1$ .
7:    $node_{critical} = Q[1]$  and  $node_{non-critical} = Q[n]$ 
8:    $\alpha = \Delta\delta^+(node_{critical})$  and  $\beta = \Delta\delta^-(node_{non-critical})$ 

9: feedback(Node  $i$ ,  $\theta$ ):
10:   $d_i = d_i + \theta$ 

11: rebalance():
12: if  $\alpha > 0$  or  $f(node_{critical}) < \text{min-freq}(node_{critical})$  then
13:   if  $\beta = 0$  then
14:     remove-update( $Q, node_{non-critical}$ )
15:   else
16:     feedback( $node_{non-critical}, -(k\beta)$ )
17:     remove-update( $Q, node_{non-critical}$ )
18:   end if
19: else
20:   remove-update( $Q, node_{critical}$ )
21: end if
22: rebalance()

23: upon end-to-end deadline miss or expiry of feedback interval  $\tau$ :
24:   measure and update  $\sigma$ 
25:   distribute()

26: distribute():
27: if  $\sigma > 0$  then
28:   if  $(\sigma - \alpha) \leq 0$  then
29:     feedback( $node_{critical}, \sigma$ )
30:   else
31:     feedback( $node_{critical}, \alpha$ )
32:      $\sigma -= \alpha$ 
33:     remove-update( $Q, node_{critical}$ )
34:     distribute()
35:   end if
36: else if  $\sigma < 0$  then
37:   if  $\sigma + \beta \geq 0$  then
38:     feedback( $node_{non-critical}, \sigma$ )
39:   else
40:     feedback( $node_{non-critical}, -\beta$ )
41:      $\sigma += \beta$ 
42:     remove-update( $Q, node_{non-critical}$ )
43:     distribute()
44:   end if
45: else if  $\sigma = 0$  or  $\sigma$  has settled then
46:   rebalance()
47: end if
```

The $Co-DVS_{node}$ mechanism in Algorithm 2 explains the proposed selective slack-distribution algorithm for a heterogeneous distributed system with individual energy sources. The outline of the mechanism is similar to Figure 2 except that a single central controller algorithm is employed and the battery states of each of the devices is also monitored and shared as feedback. The algorithm which is executed *at the sink node*, uses *selective feedback* of slack to achieve a selective distribution. If there is positive slack (σ) in the system, the slack is distributed to the critical node with the highest pay-off factor allowing it to lower its frequency and conserve higher energy. Any remaining slack is then distributed to the next critical node as described in lines 27-35 of Algorithm 2. The

algorithm reacts to deadline misses ($\sigma < 0$) by distributing the deficient slack to the non-critical nodes causing them to lower their deadlines and increase their speeds (lines 36-44). In both cases, the amount of slack distributed to a node is bounded by $[\Delta\delta_i^+, \Delta\delta_i^-]$ given in Equations 6 & 7.

A selective approach is employed in either case to distribute the slack in the system. That is, for positive slack, the most critical node is favored first and any remaining slack is then fed back to the next most critical node and so on. Similarly when slack is negative, the more non-critical nodes are forced to speed-up by allocating the deficient slack to these nodes (starting from the least critical ones) in order to reduce the incurred latencies within the specified constraints.

Rebalancing. The rebalancing procedure is used to ensure that the critical node runs at its lowest possible frequency (based on its current workload) while meeting the end-to-end deadlines. The rebalancing procedure is invoked after any positive slack in the system has been exhausted (i.e., $\sigma = 0$) or has settled (e.g., to within 5% of the average slack in the previous five feedback intervals) after distribution to the nodes based on the conditions described in lines 27-44. The rebalancing routine (lines 11-22) speeds up the non-critical nodes by lowering their deadlines by a proportion k of their respective $\Delta\delta^-$. The proportion k was assigned to each node based on the remaining battery level (pay-off factor) of the node relative to the other nodes in the system. Such a proportional rebalancing mechanism is used to avoid heavily penalizing the non-critical node by indiscriminately increasing its frequency when the same effect can be achieved by proportionally increasing the frequencies of the other nodes based on their pay-off factors.

V. EXPERIMENTAL ANALYSIS

This section describes the performance of the Co-DVS mechanisms in different scenarios with the help of simulations from a discrete-event-driven simulator built in Java. The evaluations were performed on a 3 node system where a source node transmits a packet stream to the sink node through an intermediate forwarder node. The experiments were run over a stream of 500 packets each with an end-to-end deadline of 70ms. The end-to-end slack measured at the sink was averaged over every 5 received packets.

A. Results

1) *System-wide energy conservation:* Figure 3 illustrates the adaptations performed by $Co-DVS_{system}$ for a distributed system with 3 nodes. The figure also illustrates the adaptations in response to dynamic variations in the system workload. A load disturbance in the form of a period task is introduced and varied at the sink to simulate changes in the system load.

The adaptation starts with the downstream node (sink) which utilizes the available positive slack in the system to lower its frequency. Any remaining slack is then fed back to the forwarder node which allows it to slow down to its lowest available frequency. Finally, the remaining slack is fed to the source node which uses it to scale down its frequency to 1.2GHz. When the load disturbance (background task with

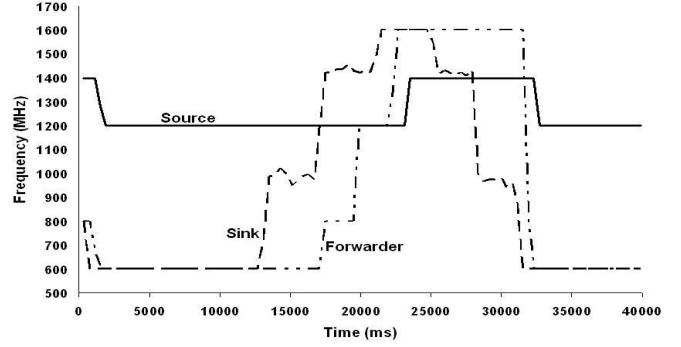


Fig. 3. Average frequencies of the nodes in $Co-DVS_{system}$.

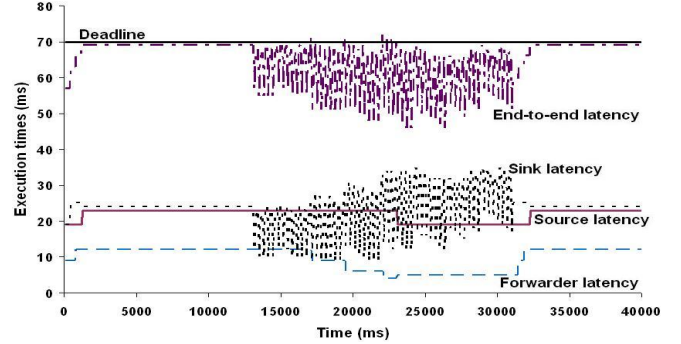


Fig. 4. Latencies incurred at different nodes in $Co-DVS_{system}$.

period = 29ms) is introduced at 13 seconds, the sink node reacts to it by scaling its frequency accordingly (note that the average frequencies at the sink shown in Figure 3 also include the frequencies at which the disturbance task is executed). The execution time of this disturbance is increased by about 60% at 17 seconds causing the sink to further increase its frequency. Since deadline misses occur even with the sink frequency at its highest setting, the forwarder is now forced to speed up its frequency. The disturbance is enhanced further at 21 seconds by increasing its execution time by another 15%. As a result of this disturbance, the forwarder is forced to further increase its frequency to bring down the end-to-end latencies within the specified timeliness constraint (70ms). When deadlines are missed further with the sink and forwarder executing at the highest speed, the source node is forced to scale its frequency to reduce the DVS-induced latencies and satisfy the end-to-end deadline. Figure 4 shows the end-to-end latencies to be maintained within the specified deadline constraint at this setting. The disturbances are then slowly removed in the order of their introduction at times 25, 28 and 31 seconds which results in the nodes lowering their frequencies accordingly.

2) *Node-specific energy conservation:* Figures 5 and 6 present the adaptations of $Co-DVS_{node}$ for two different scenarios in a heterogeneous system involving 3 nodes. The scenario where the source node is the critical node with the forwarder being the non-critical node (i.e., $pay-off_{source} > pay-off_{sink} > pay-off_{forwarder}$) is illustrated in Figure 5. The end slack in the system is first utilized by the energy-critical source node to lower its frequency and minimize its energy consumption. The end-to-end latencies are maintained with the specified constraint by accordingly scaling the frequencies of

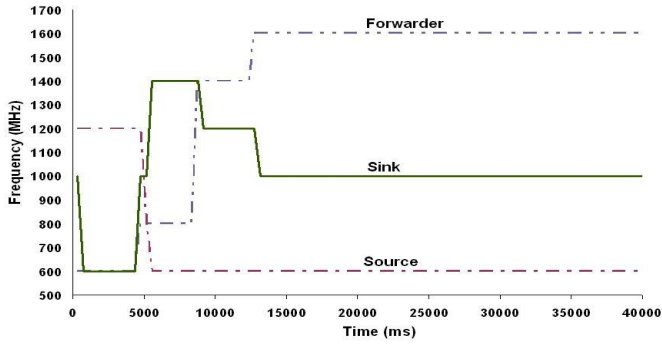


Fig. 5. Node frequencies with $Co-DVS_{node}$ when source is critical node.

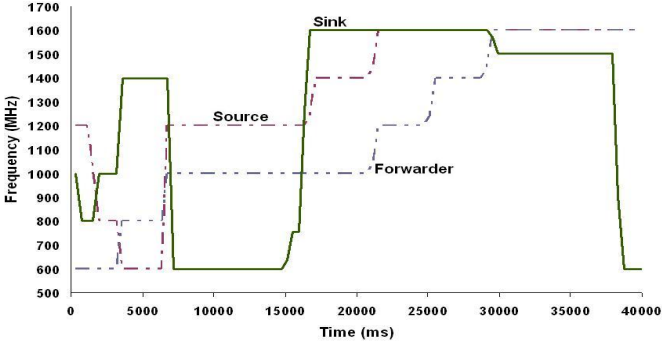


Fig. 6. Node frequencies with $Co-DVS_{node}$ when sink is critical node. the sink and forwarder nodes. An important observation is that, when the source node has lowered its frequency to the lowest available level, the sink node is also allowed to execute at a lower frequency (rebalancing at 8, 13s) by further increasing the frequency of the non-critical forwarder node. This leads to comparatively higher energy savings at the source node (most critical) and the sink node (intermediate critical) thereby prolonging the life-time of the distributed application while maintaining its specified timeliness requirements.

Figure 6 represents the scenario when the sink is the critical node and the source is the non-critical node. This figure also shows the adaptations due to load disturbances being introduced at the sink node at 15 seconds. In the period prior to the load disturbances it can be seen that the sink node begins lowering its frequency by using the slack in the system. After the sink reduces to its lowest frequency, the slack is fed to the next critical node (forwarder) and any remaining slack is finally fed to the non-critical source node. However since the non-critical nodes are upstream to the sink node, any lowering of frequencies at the upstream nodes induces latencies downstream causing the sink node to execute at a higher frequency. The rebalancing procedure, invoked after the slack has been distributed to the nodes based on their pay-off factors (at 7s), ensures that the critical node (sink) is maintained at its lowest possible frequency by forcing the non-critical nodes to run at higher frequencies.

Load disturbances. Load disturbances are introduced in the system in the form of variations in the link latencies and background tasks. For example, at 15ms in Figure 6, the link latency L_{link} (forwarder to sink) is increased by 50% of its

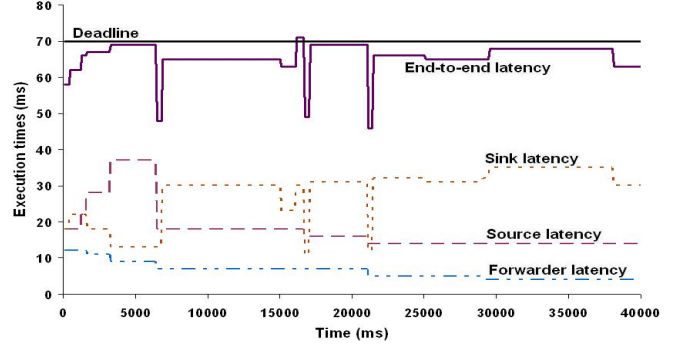


Fig. 7. Latencies incurred at different nodes in $Co-DVS_{node}$ when sink is critical node.

initial value (5ms). This results in the sink node scaling its frequency accordingly (800MHz) to keep the end-to-end latencies within the specified limits. A periodic background task (period = 80ms) is then introduced at the sink node at 16ms which results in the sink scaling to its highest frequency. Figure 7 shows the introduced disturbance to cause end-to-end deadline violations at the sink. In response, the mechanism forces the non-critical node (source) to scale its frequency (1400MHz) to reduce the latencies within the deadline constraint. After the slack in the system settles in the presence of the load disturbances, the rebalancing procedure scales the non-critical nodes (source, forwarder) to allow the energy-critical sink to lower its frequency (at 29s) and maximize its energy savings. The sink node reduces to its lowest frequency when the load disturbances are removed at 38s.

B. Energy Evaluations

Figure 8 compares $Co-DVS_{node}$ and the uncooperative ‘standard’ implementation of la-EDF (Std-DVS) in terms of the energy consumption of 3 nodes represented by their battery charge levels over a stream of 850 packets. The figure represents the scenario when the source has a low battery level of 20% (critical-node) while the forwarder has maximum charge (100%) followed by the sink with 66% charge. The battery discharge rate was computed using a linear depletion model based on the energy consumed during task run-times.

The $Co-DVS_{node}$ mechanism favors higher energy savings at the source node, to prolong its life-time, by scaling the operating frequencies of the non-critical nodes. While the energy consumption of the non-critical nodes increase, the source node is able to conserve higher energy (observed by a slower battery discharge) compared to Std-DVS. It is also important to note the rate of battery discharge among the non-critical nodes. The most non-critical node (forwarder) has a faster battery discharge compared to the intermediate critical node (sink) in order to balance the energy savings according to their respective battery levels. This allows for increased operation times of all nodes in the system (source drains its battery at 65s in Figure 8) compared to a non-cooperative approach where the energy-critical node (source) would drain its battery faster (at 32s which is 51% faster than Co-DVS), leading to early disruption of services in the system.

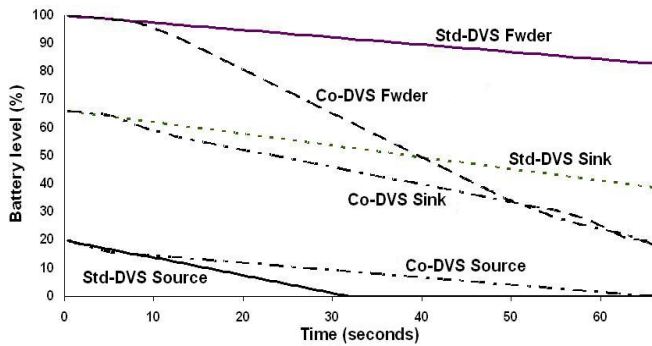


Fig. 8. Comparison of the node battery charge levels in a distributed system with 3 nodes.

VI. RELATED WORK

Deadline assignment for sub-tasks executing a global task in a distributed soft real-time system was extensively studied in [4]. A set of deadline assignment techniques are proposed for sub-tasks such that the end-to-end timeliness guarantees are maintained. However, the techniques presented in [4] do not account for energy conservation in the system.

Slack sharing mechanisms for energy management in multiprocessor systems have been studied extensively [2], [13]. The authors in these efforts adapt a slack sharing approach to achieve power-aware scheduling in multiprocessor real-time systems. By dynamically allocating slack between the processors in a multi-processing system in accordance, their proposed mechanisms achieve global energy savings while satisfying timeliness requirements for the tasks scheduled.

Our work is closely related to the work presented in [7]. They propose a slack distribution scheme that dynamically allocates slack to the various nodes in a distributed system based on the service demands at those nodes. Our work differs by addressing the coordination of DVS techniques across multiple communicating systems, where the deployment of DVS on one system affects the energy conservation on another system. In our work, the slack is allocated to nodes based on their order in a hierarchy of nodes ranked according to their pay-off factors or position in the communication path.

Finally, this paper makes the case for end-to-end energy management and argues that directives to express the overall management goal in a distributed system are needed. This is related to the establishment of QoS parameters in distributed systems, including the negotiation of QoS levels [10]. The dynamic slack-allocation mechanism introduced in this paper is achieved by sharing feedback on the end-to-end latencies and energy status of the devices in the distributed system. While feedback-based solutions have found wide-spread use in a variety of real-time systems problems [12], [14], our work applies it to an existing localized energy management technique to i) prevent it from violating end-to-end constraints and ii) achieve coordination to allow more energy-constrained devices to save higher energy.

VII. CONCLUSIONS AND FUTURE WORK

In this work, we presented an extension to the commonly adapted DVS mechanism with the goal to provide coordinated

end-to-end energy and latency management. Cooperative Dynamic Voltage Scaling or Co-DVS targets real-time mobile environments, where both timeliness and energy-efficiency are important. A key attribute of Co-DVS is that it can reduce end-to-end deadline misses by setting bounds to how much each node can exploit DVS and thereby limiting the DVS-induced latencies. Secondly, when determining these limits, Co-DVS considers the overall energy management objectives in a system, thereby shifting the focus of energy conservation from non-critical to critical nodes in a heterogeneous distributed system with individual energy resources. Co-DVS was achieved by employing selective slack distribution algorithms that adapt the deadlines of the local tasks of the distributed application based on the energy management directives (system-wide vs. node-specific) and end-to-end slack.

Our future work will continue to study and investigate the deployment of the proposed mechanisms in realistic mobile and ad-hoc settings and analyze its behavior if multiple real-time streams cross each other (e.g., the source of one stream can be the forwarder or sink of another stream).

REFERENCES

- [1] T. Burd and R. Brodersen. Energy Efficient CMOS Microprocessor Design. In *Proceedings of 28th Annual Hawaii International Conference on System Sciences.*, pages 288–297, January 1995.
- [2] J.-J. Chen, C.-Y. Yang, and T.-W. Kuo. Slack Reclamation for Real-time Task Scheduling over Dynamic Voltage Scaling Multiprocessors. *IEEE International conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, 1:358–367, 2006.
- [3] D. P. Helmbold, D. E. Long, and B. Sherrod. A Dynamic Disk Spin-down Technique for Mobile Computing. In *Proc. of the Intl. Conference on Mobile Computing and Networking*, November 1996.
- [4] B. Kao and H. Garcia-Molina. Deadline Assignment in a Distributed Soft Real-time System. *IEEE Transactions on Parallel and Distributed Systems*, 8(12):1268–1274, 1997.
- [5] R. Kravets, K. L. Calvert, and K. Schwan. Payoff-based Communication Adaptation based on Network Service Availability. In *International Conference on Multimedia Computing and Systems*, pages 33–42, 1998.
- [6] R. Kravets and P. Krishnan. Application-driven Power Management for Mobile Communication. *Wireless Networks*, 6(4):263–277, 2000.
- [7] R. N. Mahapatra and W. Zhao. An Energy-Efficient Slack Distribution Technique for Multimode Distributed Real-time Embedded Systems. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):650–662.
- [8] M. A. Marsan, G. Balbo, G. Conte, and F. Gregoretti. Modeling Bus Contention and Memory Interference in a Multiprocessor System. *IEEE Transactions on Computers*, C-32(1):60–72, 1983.
- [9] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of the 18th ACM Symposium on Operating Systems Principles*, October 2001.
- [10] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A Resource Allocation Model for QoS Management. In *Proc. of the IEEE Real Time Systems Symposium*, December 1997.
- [11] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proc. of the Design Automation Conference*, June 2001.
- [12] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao. The Case for Feedback Control Real-Time Scheduling. In *Proc. of the 11th Euromicro Conference on Real-Time Systems*, June 1999.
- [13] D. Zhu, R. Melhem, and B. R. Childers. Scheduling with Dynamic Voltage/Speed Adjustment using Slack Reclamation in Multiprocessor Real-Time Systems. *IEEE Transactions on Parallel Distributed Systems*, 14(7):686–700, 2003.
- [14] Y. Zhu and F. Mueller. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2004.