Device Independent Remote Adaptations for Power Optimization using Distributed Middleware

Shivajit Mohapatra, Christopher Bell & Nalini Venkatasubramanian
School of Information & Computer Science
University of California, Irvine, CA 92697-3425
{mopy,cbell,nalini}@ics.uci.edu

Abstract

Providing quality of service(QoS) guarantees for power vs. performance tradeoffs in distributed middleware frameworks is a crucial research challenge. Future middleware frameworks need to address the issue of providing acceptable performance guarantees in the presence of low-power mobile devices. Most current energy optimization schemes (e.g. DVS) target individual processing elements(nodes) and lack a global view of the distributed system; therefore the inherent advantages of distribution remains unharnessed. In this work, we introduce a middleware based distributed dynamic voltage scaling scheme (R-DVS) for voltage setting in distributed nodes remotely, while exploiting the knowledge of the global system state. To further optimize the power gains at the devices, an optimal task partitioning and offloading scheme is integrated with R-DVS. This paper makes two significant contributions - i) we show through extensive simulations that R-DVS is a viable middleware solution and ii) a middleware framework combining high-level adaptations and low-level power management can yield significant power gains, while retaining the advantages of distribution and device level performance. In our extensive simulations we show, that our approach yields significant energy gains under a correct choice of operating parameters and provides power gains as high as 75% for low power devices.

1 Motivation

Rapid advances in processor and wireless networking technology are ushering in a new generation of low-power mobile computers(e.g handheld computers) into ubiquitous environments. These devices have modest sizes and weights, and therefore inadequate resources - lower processing power, memory, display capabilities, storage and limited battery lifetime as compared to desktop systems. Using these heterogeneous resource deficient computers alongside high-end systems, introduces new challenges for resource and power management in ubiquitous environments. In this paper, we address the issue of power management for large scale ubiquitous environments, by using a proxy-based distributed adaptive middleware solutions.

Recent years have witnessed researchers aggressively trying to propose and optimize techniques for power management in low-power computers. Several interesting solutions have been proposed at various computational levels - hardware level architectural optimizations, OS level dynamic voltage scaling(DVS) [16, 9] for optimal CPU power consumption, dynamic power management of disks and network interfaces and efficient compilers. Unfortunately, these power management techniques have concentrated on single autonomous low-power systems, potentially missing opportunities for substantial benefits achievable by exploiting a global knowledge of the system. The GRACE project [24] uses both coarse grained and fine grained tuning through global co-ordination and local adaptation of hardware, OS and application layers. In GRACE, both global (coarse) adaptations are performed using a global coordinator and a scheduling adapter is used for fine grained local adaptations. In our approach, we shift the scheduling analysis to a proxy and communicate the voltage settings to the device over the network.

Ubiquitous environments can facilitate a more global approach to power optimization to achieve high power and performance benefits at each distributed system, while constantly adapting to the i) device specific constraints (power/computing/

communication capabilities) ii) nature of executing tasks (soft realtime, non realtime) and iii) dynamically changing system conditions (task migration, mobility). Most existing distributed middleware platforms such as CORBA, J2ME, and DCOM provide high-level programming abstractions that facilitate distributed computing; however, adaptations for power management on distributed low-power mobile devices remains largely unaddressed. Providing a framework that facilitates power management is a desirable goal.

The purpose of our study is to develop a proxy-based distributed middleware solution that uses global system view to facilitate unified operation of two power management techniques - dynamic task partitioning and offloading and dynamic voltage scaling [16, 22]. DVS typically involves the following two steps; first by using standard real-time schedulability analysis [16, 22] for a given *static* task set, a static CPU slowdown factor(SSF) is determined, by which a processor can be slowed without missing deadlines. Subsequently, a dynamic CPU slowdown(cycle stealing) [16, 22] is employed by identifying and utilizing the dynamic cpu slack introduced by executing tasks. However, existing DVS analysis assumes a *static* set of tasks in its schedulability analysis for determining execution feasibility and cpu utilization. Due to its static nature, DVS performance can suffer significantly in the presence of device mobility and dynamic task changes that occur commonly in ubiquitous environments.

We propose the use of distributed proxies with knowledge of mobility patterns and task loads of low-power devices, to predictively determine future changes in operating loads(task set changes due to task migration/mobility) for a low-power device and perform the static DVS analysis remotely. Using an optimal schedulabilty analysis remotely, the proxy can determine the CPU slowdown factors and relay them to the device(albeit at a delay equal to the network communication delay). Moreover, the dynamic cycle stealing can be achieved through feedback based remote adaptation techniques. Additionally, proxies can be employed to remotely execute tasks on behalf of low-power devices thereby reducing load at the devices. The advantages of partitioning tasks between a proxy and a low-power device to save power have been studied in [13, 11, 10, 18, 15]. We combine an effective proxy based task-partitioning and offloading mechanism to compliment the power gains achieved through DVS. By adopting a distributed middleware solution, we can achieve the following benefits: (a) A seamless integration of application level task-partitioning algorithms and OS level dynamic voltage scaling. (b) The complex optimal schedulabilty analysis is shifted away from the resource deficient computers to a proxy; as a result, the middleware and the scheduler on a distributed node can be made "light-weight" with no changes to the operating system components of the devices (c) Various OS level scheduling optimization techniques for DVS (e.g Rate Monotonic, EDF) can be used for different low-power devices, without the scheduler having to implement them; furthermore switching scheduling algorithms on the fly would remain transparent to the node.

Research Contributions: In this paper, we integrate two independent cross layer power optimization techniques, namely DVS and task partitioning/offloading using a proxy-based distributed middleware solution. Specifically, we (i) employ an adaptive middleware that uses global knowledge of the system to perform DVS analysis remotely on behalf of mobile low-power devices in ubiquitous environments. (ii) propose feedback based policies that can be used to achieve further improvements over the static slow-down factor. (iii) implement an optimal dynamic task partitioning algorithm at the middleware for further optimizing the power gains. (iii) we simulate the middleware adaptations using a prototype proxy and low-power device and study the power gains achieved through the above adaptations. Our performance results indicate that significant gains in both energy and performance are observed when a global adaptation scheme is employed to combine dynamic voltage scaling and task partitioning. A local DVS implementation is unable to adapt to dynamic changes and a middleware based global adaptation provides better energy and performance gains(as much as 30% less number of deadline misses with a similar energy overhead) under dynamic conditions.

2 System Architecture

A simplified distributed system architecture is envisioned in Fig. 1(i). The system consists of five primary components - the application servers (e.g. multimedia server), the network (wired, wireless), the proxy server, a directory service and

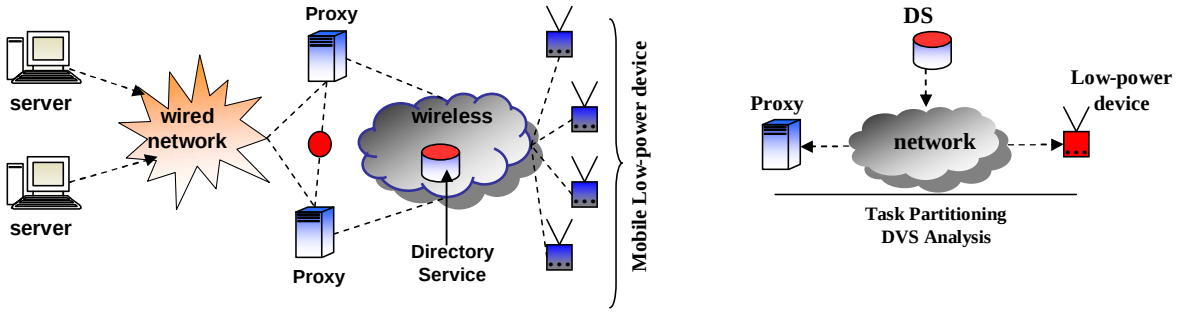


Fig. 1: **(i) The overall System Architecture(l) (ii) The Proxy, the low-power device and the Directory Service are used for the power optimizations**

the distributed mobile clients. We first describe the various components within the framework and their interactions. A middleware runtime component executes on the proxy and the client systems and all interactions are addressed through the runtime.

• *Application Servers* : The application servers are high end servers on the network that provide an assortment of services (e.g. video/audio streaming, web hosting, online banking services etc.). These are connected to a wired network and are capable of delivering data reliably at a high speed. The architecture makes no assumptions on the software packages used at the servers, nor does it require servers to manage content (e.g. videos) in specific formats or versions.

• *Network* : The framework uses two different types of network resources: the wired backbone networks, linking the application servers to the proxies, have large bandwidths and high reliability. The mobile low-power devices communicate with the proxy over wireless networks that typically have a limited, variable bandwidth.

• *Directory Service* : The directory service stores the real-time transient information about the global system state, such as mobility information, applications(task load) on the client etc.

• *Proxy* : As illustrated in Fig. 1(i), data is routed from the servers to the mobile clients, through intermediate servers, called proxies, where additional treatment to the requested data can be performed. Proxy servers are delegated to perform one or more of the following functions: 1) execute tasks on behalf of low-power devices, 2) transcode data for improved performance at a client, 3) filter tasks destined for a device (e.g. outdated events) and 4) predict new tasks that may be required to be started on a device and perform the necessary schedulability analysis on behalf of the device, based on updated information from the directory service. The proxy hosts a middleware entity called the "*power broker*", that co-ordinates and controls the task load, the operating cpu voltages and the data QoS for devices attached to it. Additionally, we assume that the proxy has the necessary location management capabilities, that allow it to locate the mobile devices at any point in time.

• *Clients* : The heterogenous clients are the end-users of the requested data. They run applications tasks on various platforms and are connected to the proxy through either wireless or fixed networks. Fig. 1 depicts the clients as mobile, low-power devices that communicate over a wireless network. Upon startup, all clients register with a proxy and update current state information at the directory service. Mobile client devices can move from one wireless zone to another and may detach from the current proxy and re-attach to another (possibly nearer) proxy.

In an ubiquitous environment, there would be a large number of servers inhered in the wired network. A number of proxy servers would be accessible within each wireless cell. The load (CPU, Disk, Network B/W availability) on each of the proxy servers would be distinct, depending on the number of clients being processed through the proxy. We make the following additional assumptions. *DVS capable mobile clients* : The mobile clients within the ubiquitous environment are equipped with processors that are capable of operating at different voltage(frequency) levels. However, each client could have a different processor. *Energy consumption profile* : Each application task is separately profiled for its energy usage with respect to computation and communication. Considerable amount of work has been done to profile the energy

usage of typical applications [13, 2, 8] and middleware services. We assume that these figures are available for use by our middleware. *Task migration* : Addressing the issue of how tasks are migrated while preserving their state and semantics remains outside scope of this paper. This topic has been widely studied [17, 19] and we assume that task migration can happen seamlessly. *Real-time tasks* : We assume that all the real-time tasks executing on the device are soft real-time in nature (i.e. they are relatively immune to small number of deadline misses). Moreover, the directory service is updated in real-time when new applications are started at the client by the user.

Benefits of Composing Adaptive Strategies

We use a distributed middleware solution to accomplishes power savings in low-power portable computers operating in ubiquitous environments using two concurrent strategies. We use a middleware entity called a "power broker", that executes at a proxy and has knowledge of individual device state, to optimally partition the tasks executing on the device and migrate one set of tasks to the proxy, saving the extra computation energy at the device. The task partitioning can dynamically change the task set on the device. Using this knowledge, the power broker also performs an optimal schedulabilty analysis for the new task set apriori, and relays the new cpu operating point to the device. Subsequently, it performs remote adaptations for slowing down CPU even further for further power gains. Note that the tasks/applications executing on the client device could change due to the following reasons: (i) the user starts a new application, (ii) the power broker executes the task partitioning algorithm and determines a new task partition and/or (iii) the mobile client moves to a new zone. For example, if the mobile client is about to enter an insecure zone, the middleware(proxy) might detect this in advance and decide to start a new security(e.g encryption/decryption) module causing a change of the task set. The proxy can use its location management module to detect the task set changes ahead of time, and perform DVS analysis for the new task set apriori. DVS analysis is performed when task set changes occur at the device possibly due to one of the above reasons. In the following two sections, we first propose our middleware based remote dynamic voltage scaling technique and a power-aware task partitioning algorithm.

3 R-DVS: A Distributed Dynamic Voltage Scaling Scheme

R-DVS extends the concept of dynamic voltage scaling by remotely computing a static slowdown factor for a processor and using a feedback based approach to adapt and further improve the CPU slowdown over an interval. In this section, we introduce the distributed dynamic voltage scaling scheme and contrast our approach to the traditional local-DVS (implemented within operating system schedulers on individual nodes). Later, we develop a set of policies based on statistical analysis techniques [3] that are used for dynamic voltage adaptation remotely over a network. Specifically, we apply time-series analysis to dynamically detect cycles(seasonality) in the execution times of tasks and identify parameters for optimally tuning power and performance for a client. We characterize each real-time task on the mobile client as a 4-tuple $< T_i, D_i, C_i, B_i >$, where T_i is the periodicity of the task, D_i is the relative deadline, $< C_i, B_i >$ represent the worst case execution time(WCET) and the best case execution time of the task (BCET) respectively, assuming it is the only task executing at the device.

Fig. 2(i) presents a flow-chart of the algorithm followed at the proxy for remotely setting the voltage on a node. The R-DVS technique is explain as follows:

- The first step in dynamic voltage scaling is to determine a static slowdown factor(SSF), based on the WCETs of all the tasks executing on a node. This is a static onetime (offline) calculation and therefore can easily be performed at a proxy (note that DVS assumes a static task set). The power gains due to static slowdown are the same for both local and remote-DVS. The only difference being that in our approach, the proxy middleware determines the SSF and relays it to the respective client device. The middleware on the device then sets the processor voltage

to the estimated value. This provides a significant power(from 20% to about 60% [9]) saving at the client device depending on the applications executing at the device.

- Subsequently, a dynamic cpu slowdown is employed based on actual execution times of tasks(WCETs are pessimistic estimates), by analyzing and utilizing the slack introduced by a task in the current execution cycle. This typically employs a prediction based strategy [9] for accurately predicting the execution time of the next task instant. Using R-DVS, we perform dynamic slowdown(slack stealing) by predicting a slowdown over an interval rather than the next executing task instant. Slack stealing techniques in local DVS attempt to utilize every single slack cpu cycle. This may require substantial computation to be performed at each scheduling point and adds complexity to the scheduler; R-DVS cannot make such fine-grained adaptations because of two primary reasons: i) Adaptations are made remotely using a proxy, and the communication delays and overheads involved in sending/receiving feedback data every task cycle is unreasonable, and ii) it makes no assumptions on the capabilities of a client device. The scheduler on the client(e.g. sensor) may not be capable of such fine grained analysis and optimizations. Distributed DVS, therefore uses a larger interval and optimizes the execution over an interval. Moreover, local DVS techniques calculate cpu slowdown assuming an entirely real-time task set. However, in ubiquitous computing environments, there exist a mixture of periodic, aperiodic and best-effort tasks. Some CPU idleness in the execution of the real-time tasks is desirable as it can be used to execute these tasks, preventing starvation. Additionally, the proxy can implement a whole selection of algorithms and adaptation policies; changing of analysis or adaptation policies remains transparent to the client. As an example, the analysis for independent task sets would be different from inter-dependent task sets requiring synchronization. At the device level dependency is hard to detect and the analysis is very expensive to perform in real-time, due to resource limitations. R-DVS can eliminate this problem as seen in Fig. 2(i).

3.1 Remote Adaptation Policies

To improve upon the static slowdown gains, the R-DVS scheme adopts a remote feedback/adaptation based approach for performing dynamic slowdown. The proxy makes dynamic adaptation decisions based on a feedback report(bulletin) from the client. Based on the capability of the client, we devise several policies for remote adaptation. Three such policies are described here. The first two policies assume very simple bulletins containing simply the total number of deadline misses and the average cpu utilization over the last interval(since previous bulletin). The third policy uses an additional larger bulletin to include the list of individual task periods, the actual execution times of the tasks over a cycle. Before describing the remote adaptation strategies, we define the following parameters. Let,

- f_{update} : The frequency of the update bulletins. If this is made sufficiently low, the delay in the transmitting the feedback bulletins will not significantly impact the adaptation.
- D_T: Total number of deadline misses at the node in the latest bulletin.
- α_{wcet}: cpu utilization assuming WCET for all tasks.
- α: Average cpu utilization in the current bulletin.
- ϱ: Delay involved in the transmission of bulletins.
- D_{min}: Lower threshold of deadline misses. If number of missed deadlines are less than this, then we can further slowdown the processor.
- D_{max}: Higher threshold of deadline misses. If number of missed deadlines are more than this, then we need to increase the processor speed.
- A_{factor}: Adaptation Factor: a factor by which the operating voltage is shifted. We predict the value of the slowdown factor based on f_{update}. The update interval is set to the LCM of task periods and scaled a desired value. To reduce the overhead of communication at the node and to smooth the effect of transmission delay, we choose a sufficiently small value of f_{update}(such that $(\frac{1}{f_{update}} \gg \varrho)$. A time series analysis is done to calculate the adaptive slowdown factor(Fig. 3).

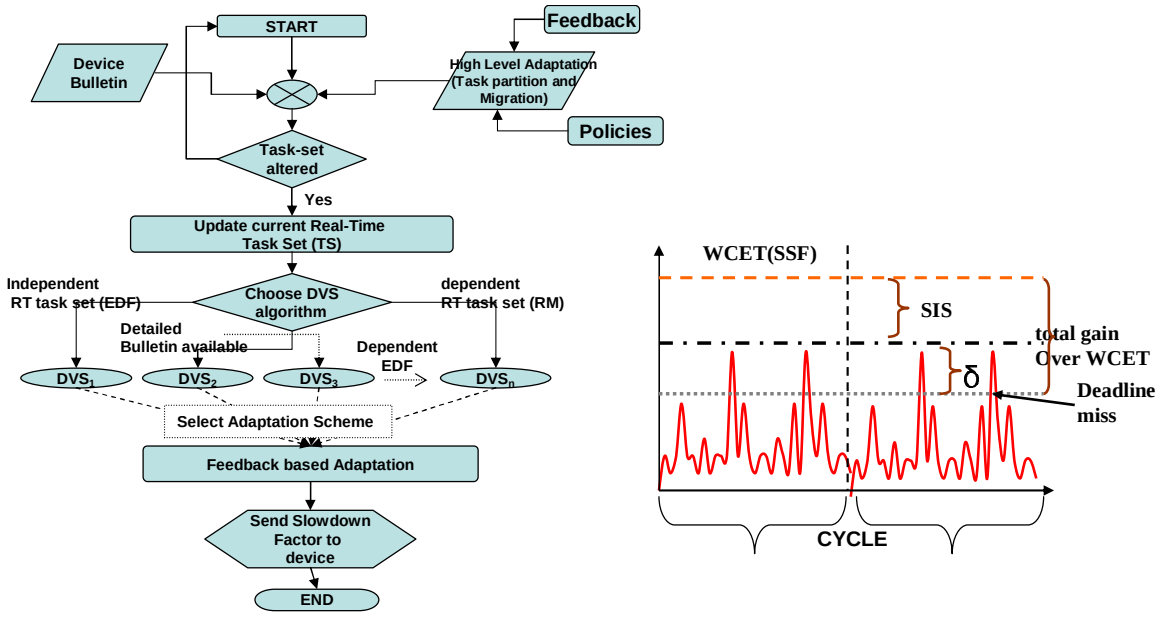


Fig. 2: **(i)Flowchart for performing Remote DVS using a Proxy (ii) Cyclic execution times**

The remote speed setting policy is explained below and can be used when EDF based analysis is used. In reality, the proxy will implement a variety of policies and pick the optimal policy for a client device.
Below we describe our adaptation policies.

• **k-Weighted Moving Average with Prediction by Exponential Smoothing**: This scheme uses the "k" most recent bulletins from the device to adapt the cpu voltage for the next upcoming interval. It predicts the number of deadline misses for the upcoming interval by using a weighted average of the deadline misses in the previous intervals. The predicted value is compared to D_{min} and D_{max}, and a decision is made whether to whether to increase or decrease the processor speed(Fig. 3). In effect, this tries to smooth a global average of missed deadlines, while providing adequate consideration for local variations. We compute the slowdown value by using a simple exponential smoothing of the cpu utilization as in Fig. 3.

• **Flat**: This method sets the utilization of the next interval to be a constant real number in the range [0,1], and then uses this utilization to predict the slowdown or speed up of the processor as above. The value of the constant(F_{factor}) can be conservative (closer to α_{wcet}) or aggressive depending on the balance of power and performance desired.

• **Dynamic Adaptation with Online Cycle Detection**: In this approach a detailed bulletin containing the actual execution time of the tasks is analyzed at the proxy for detection of cycles. If a pattern is discovered in the execution times of the tasks, then an optimal adaptation scheme can be generated. If no cycle is detected, the scheme reverts back to one of the above adaptation policies. A detailed bulletin can be used to build a complete picture of the execution of the task set over the length of the interval. Patterns can be detected by using a time-series analysis [3] of the feedback data. To optimize the communication costs, a limit is set on the number of detailed bulletins sent by a node. Again, only nodes with enough resources to keep track of the necessary data, and the communication capabilities can respond with a detailed feedback. Fig. 2 shows a case when the proxy detects a recurring pattern in the actual execution times of the tasks on the node. This could happen if individual tasks on the node have cyclic execution times. In such a case, the proxy can dynamically identify a cycle/pattern from the feedback data and make optimal adaptations. It can now statically reduce the voltage to the highest peak in the graph (see Fig. 2(ii)). We call this the Static Initial Slowdown(SIS). Additionally, it can also decide how the execution is further affected by dynamic slowdown(e.g.δ in Fig. 2). The knobs provided by the adaptation can tune both the number of deadlines missed and the aggressiveness

```
Procedure ComputeDynSlowdown
Input : Device_Bulletin [k], αWCET
{
   D := PredictedDeadlineMiss();
   A_FACTOR = Calc_Adaptive_Factor()
   IF (D >= D_MAX)
      increase speed by A_FACTOR.
   ELSE IF (D < D_MIN)
      decrease speed by A_FACTOR
   ELSE
      retain current slowdown.
}
```

```
Procedure Calc_Adaptive_Factor;
{
 // Simple exponential smoothing
 α_T+1 = βα_PREV + (1-β) α_T
 A_FACTOR = | α_T+1 - α_PREV |

 return (A_FACTOR)
}
```

```
Procedure PredictDeadlineMiss;
Input : Device_Bulletin [k]
{
   set local := l;     /* recent history l=3 for example*/
   set past := k-l;

   predicted := A weighted avg. of the deadline misses
                over the past  k bulletins, giving more
                 weight to the recent past
   return (predicted);
}
```

Fig. 3: **k-Weighted Moving Average**

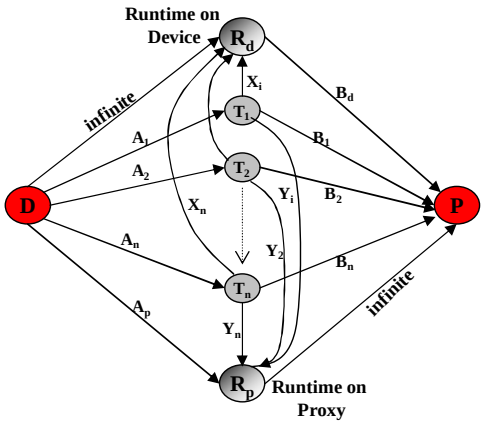
of the dynamic adaptation. Moreover, it becomes possible to identify the task instances that will miss deadlines. In practice, exact cycles are less likely to occur, so the detection algorithm can approximately determine the presence of cycles, which would still provide substantial improvements over the other policies.

4 Adaptive Task Partitioning

In low-power devices, cpu computation and network communication account for a very high percentage of energy consumed. For many cpu intensive applications, the energy cost of executing these tasks on low-power devices can be excessively high. This can very quickly drain power from these devices. One effective solution to this problem is to execute the tasks on a remote proxy server, and simply use the results locally. With this approach, the entire cost of cpu computation at the device is saved, but a communication cost and a performance overhead is incurred in transferring the task and receiving the results. If the cost of communication is smaller than the cost of computation, then significant energy gains can be achieved using this technique. As seen in the previous section, R-DVS adapts the voltage scaling for a device to dynamically changing application sets. We identify the tasks on the device that can be offloaded to the proxy and then optimally partition the tasks into two sets, one of which is remotely executed at the proxy. This technique can induce additional gains due to DVS, as the CPU utilization is reduced, allowing for further lowering of the CPU operating voltage.

[11, 10, 15, 18, 13] all explore the use of computation offloading to remote machines for power gains in low power devices. We briefly describe and use the optimal task partitioning algorithms similar to [13]. To optimally partition the tasks, we define the parameters used for characterizing the "energy overheads" of the each task(note that all power costs are incurred at the device). The communication cost between tasks on different computers(proxy and client) is characterized as the cost of sending messages from the task on one computer to the middleware runtime of the other computer. The computation costs are energy overheads for executing the tasks on the device. Note that the proxies can store images of the tasks, so that task offloading need not involve the transfer of the actual address space. Instead, the transfer can be achieved by starting the task on the proxy and transferring just the state information from the device to the proxy [13]. To achieve an optimal distribution of tasks between the device and the proxy, the partitioning problem is mapped as a source parametric flow graph problem [1], such that the minimum cut of the flow graph gives us an optimal mapping of tasks for the device and the proxy. We incorporate the energy costs of network communication and cpu computation into a energy flow network. To create the flow network(graph in Fig. 4), we distinguish two special nodes in the network: *a source node D and a sink node P*. Additionally, we define two conceptual nodes $\boldsymbol{R_d}$ and $\boldsymbol{R_p}$, that represent the core runtime frameworks on device and the proxy respectively(see Fig. 4). Fig. 4 presents a high level algorithm for execution of the partitioning algorithm. In our previous work [13, 12], we discussed in detail the particulars of the network flow algorithm used and the results of our extensive simulation.

The flowchart in Fig. 2(i) represents the high-level functionality of the proxy. The proxy middleware first tries to make



BEGIN
WHILE (*Change_In_Task_Set == TRUE*) **DO**
1. Create a source parametric flow network using the costs of computation and communication. Use Residual power of the device to compute B_i .
2. Find "minimum cut" of the graph.
3. Determine tasks that will be assigned to the proxy (set of "task nodes" that are connected to the node P).
4. Migrate the tasks to the proxy
5. Execute identified tasks on proxy and return the results back to device.
END WHILE;
END

Fig. 4: **Flow Network and High level Task Partitioning Algorithm**

high-level adaptations(task partitioning) based on certain administrative policies. If the partitioning results in a change in the task set at the device, it makes an optimal offline DVS analysis of the new task set and relays a new cpu slowdown factor to the device. We identify the conditions under which the CPU slowdown needs to be recalculated for a device and present three high-level adaptation policies that dictate the times and the frequencies at which the task partitioning is initiated. The slowdown for a distributed device needs to be recomputed when (i) the feedback from the device indicates a change in the task set (some task has been stopped/killed), (ii) a real-time task has been relocated to the proxy from the device (task partitioning) and (iii) a high-level adaptation resulted in the change of a task characteristic (e.g. deadline of a task is changed). We define the following three policies that dictate the task partitioning frequency between a device and the proxy:

- Application Triggered - The task partitioning is triggered every time a new task is started(imminent) at a device.
- Threshold Triggered : Partitioning occurs when the device power level drops below a certain predefined threshold.
- Periodic Trigger : Partitioning occurs at regular periodic intervals.

5 Performance Evaluation

We have developed a simulator to evaluate the potential energy savings achievable through our integrated approach. Within our simulator, we independently model the low-power device, the proxy server and the tasks. We first describe the various models, the simulation methodology and the assumptions in the simulation. Later, we present and analyze the simulation results.

5.1 Experimental Setup

The Device Model : Each device runs the core runtime middleware components and has a variable speed processor(VSP [9]). A VSP supports multiple frequency levels for cpu operation. For each frequency level there is a corresponding operating voltage, that is supplied by the manufacturer. We assume that the speed of the processor can be varied continuously from some $\boldsymbol{S_{min}}$ to $\boldsymbol{S_{max}}$, the minimum and the maximum supported speeds, and normalize the values such that the operating range varies from $[\boldsymbol{\nu_{min}}, \mathbf{1}]$, where $\boldsymbol{\nu_{min}} = \frac{\boldsymbol{S_{min}}}{\boldsymbol{S_{max}}}$. We use the energy model as presented in [9] to calculate the power P as a function of slowdown($\boldsymbol{\nu}$).

$$\boldsymbol{P = f(\nu) = 0.284 \cdot \nu^3 + 0.225 \cdot \nu^2 + 0.0256 \cdot \nu + \sqrt{311.16 \cdot \nu^2 + 282.24 \cdot \nu} \times (0.0064 \cdot \nu + 0.014112 \cdot \nu^2)} \quad (1)$$

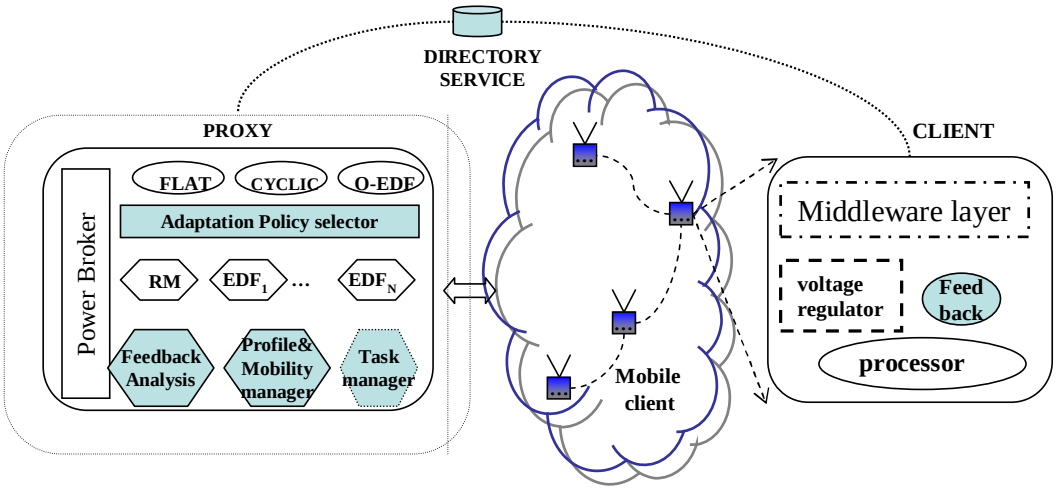


Fig. 5: **Implementation Prototype**

In reality, most processors will have discrete operating frequency levels, allowing voltage stepping in discrete steps and not in a continuous manner. In this case, the voltage can be set to the next highest available voltage.

The Task Model: The task model typifies the individual application and middleware tasks executing on a particular device. Each device model has an associated task model. In our simulation framework, we assume(without loss of generality) an independent set of soft realtime tasks, each of which can be migrated to the proxy. Some tasks are considered bound to the device(e.g. display), and are assumed to consume energy at a constant rate. The energy overheads for individual task computation and communication are based on accurate and comprehensive profiling of data on a specific device type. Our simulator's profiled data is based on measurements made on a Compaq iPaq 3650, driven by a 206Mhz Intel StrongArm processor, with 16MB ROM, 32MB SDRAM, using a Lucent Orinoco Silver 11Mbps wireless PCMCIA network interface card for communication. [2] presents the energy consumption characteristics of mpeg video playback on hand-held computers. We borrow the some of the results presented in this paper to model a typical video playback application for our simulation. In [8, 13], power usage rates of some typical computation intensive applications are empirically determined. We draw on some of the results illustrated in these papers to model the power usage patterns of some of our simulated application tasks. The simulator also assumes that all tasks executing on the mobile device are available at the proxy, and task migration involves stopping the current task on the device, starting the task at the proxy and then transferring necessary state information, without having to transfer the process image itself.

The Proxy: We model the proxy as a server with seemingly unlimited system resources and very high computing capability. It implements various DVS algorithms and remote adaptation schemes, and a selector for choosing the optimal combination of algorithm and adaptation policy for a device. It also implements the optimal task partition algorithm described earlier. We simulate the mobility by using a static grid and a predefined set of changes to the task set at each grid. A random direction and velocity is generated for each device within the grid.

The Execution Model: The simulation is conducted for a specific device model(iPAQ) and the corresponding application models. The simulator implements a device level dynamic voltage scaling scheme similar to the one described by Mani et al. in [9], the optimal task partitioning scheme and the remote DVS schemes described earlier. We first study the impact of task partitioning in the absence of any form of DVS. Then we compare the performance of our remote DVS scheme to that device level DVS. During this optimization we identify the ideal operating parameters for our adaptation scheme for a given realtime task set. Next we present the performance of both the DVS schemes in the presence of task partitioning. We use three different QoS metrics in our experiments - (1) relative number of deadlines missed, (2) energy gains achieved over static slowdown factor(SSF) and (3) total energy consumed (indicative of the residual power in the device). As in [9], we calculate BCET as a percentage of the WCET. We generate the actual execution times of the task

instances using a Gaussian distribution with mean($\boldsymbol{\mu} = \frac{\boldsymbol{WCET+BCET}}{\mathbf{2}}$) and standard deviation($\boldsymbol{\sigma} = \frac{\boldsymbol{WCET-BCET}}{\mathbf{6}}$). Moreover, as the static slowdown factor is calculated offline in both local and remote DVS, we simply compare the gains(over SSF) due to the dynamic adaptation used in these schemes, all through our simulation. Note that the gains due to static slowdown are the same for both. All simulations are run for significantly long periods of time(in the order of minutes).

Network Communication Overhead

The overhead of the network communication has a significant impact on both the strategies described above. In our analysis, we assume a fairly strong wireless signal and use typical network delays in wireless environments of 10 to 100 milliseconds. Note that both the task partitioning (or task-set changes) and the remote adaptation(in secs) happen in the order of seconds (or even minutes), so that the effect of the network delay on the task partitioning and R-DVS is minimized (perceived for the cycle where network communication actually happens). In our evaluation, we use the update frequency to measure the performance, keeping a constant network delay. As the update frequency is decreased, the effect of network delay is reduced significantly and vice-versa (as seen in Fig. 13).

5.2 Experimental Results

5.2.1 Task Partitioning Only:

In this section we show the impact of task partitioning on the energy gains at a low-power device in the absence of dynamic voltage scaling. In our simulation, the device registers itself with a proxy and the runtime on the device updates the device state at the directory service. The power broker at the proxy then runs the partition algorithm and announces the optimal configuration to the device and the proxy. A reconfiguration agent on the device then performs the necessary component migration. As all tasks components are replicated at the proxy, only a minimal amount of state(e.g queued messages) is communicated and this includes a small communication overhead. We divide our set of representative tasks into three classes: computation intensive (class-1, e.g *Image processing, interactive games*), communication intensive(class-2, e.g. *web browsing, network monitoring*) and both computation and communication intensive (class-3, e.g *multimedia streaming, GIS/navigation*). For each class of tasks we assess the energy gains using a set of

- *sporadic-start applications*: applications that start and stop irregularly over time.
- *non-sporadic applications* : applications that run continuously till the device runs out of power.

Our primary metric of evaluation is the *gain* achieved in the

- *Residual Energy (ER)*: showing the unexpended energy left in the device.
- *Remaining Time of Service(TOS)*: indicating the remaining time for which the device can be operational, under the current load.

By "*gain*", we mean the TOS/ER saved by running the tasks with and without the partitioning algorithm. We study how the TOS/ER gains are impacted as the number of applications scale; we also examine how often the broker should reconfigure the tasks in order to achieve optimal energy benefits.

Fig. 6 shows the impact of varying the number of tasks (of type class1) randomly on the time of service gain at the device. The service time gains are seen to be significant in all the cases. The irregularity of the gains is due to the sporadic nature of the tasks. Fig. 7 shows the gain in residual energy for various sporadically started class-2 applications. The highest gains are seen when the number of tasks are more. Fig. 8 shows the residual energy gains for randomly started class-3 applications. Note that the gains achieved are not always positive as shown in our previous work in [13]. This is due to the random nature of the tasks. However, a general observation was that the gains were significant if reconfigurations were more frequent(¡ 5 minutes).

In conclusion, we see that significant energy gains can be achieved due to dynamic task partitioning. Under a proper

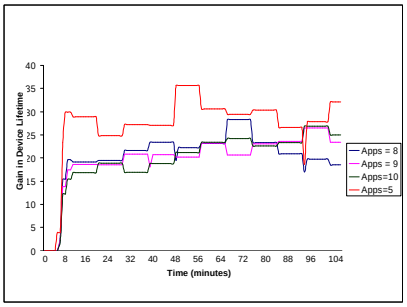


Fig. 6: TOS(class1,sporadic)

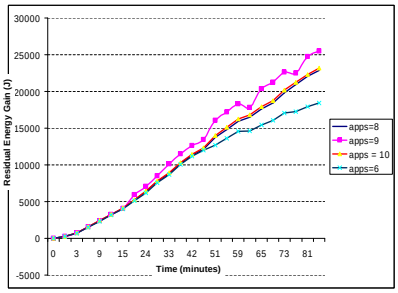


Fig. 7: ER(class2, sporadic)

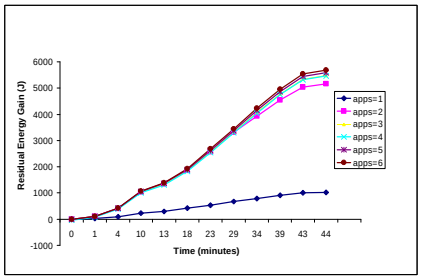


Fig. 8: ER(class3,sporadic)

choice of reconfiguration frequency, a gain from 10% to 50% in the energy consumption is observed. Also, the gains were observed to be higher for the CPU intensive class-1 applications and moderate for the class-3 applications.

5.2.2 R-DVS Based Adaptation vs. Local-DVS:

In this section, we compare the performance of our remote DVS adaptation scheme with that of local-DVS using two separate real-time task sets and using the Earliest Deadline First scheduling algorithm for schedulability analysis. Fig. 9 and Fig. 10 compare the percentage gains over static slowdown and the total energy consumption respectively, for D-DVS and R-DVS(using the k-weighted (A1 in graph) and FLAT) for the INS task set. Fig. 11 gives the corresponding percentage of deadlines missed. A comparison shows that at $\frac{\boldsymbol{BCET}}{\boldsymbol{WCET}} > \mathbf{50\%}$, the number of deadlines missed for local-DVS is zero and both the R-DVS adaptation policies show a small number of deadline misses. However, the R-DVS schemes show energy gains over D-DVS in that range. The deadline misses are due to two factors, the threshold deadline values($\boldsymbol{D_{min}}$&$\boldsymbol{D_{max}}$) and the network delay. A tighter bound on these thresholds indeed reduces the number of deadline misses. Fig. 12 compares the performance of the two R-DVS schemes with respect to a real-time task set with an initial CPU utilization of 50.6%. As seen from the figure, our remote adaptation scheme closely matches the performance of device level DVS. The FLAT scheme shows a relatively large number of deadline misses. This was due to the fact that the adaptation factor($\boldsymbol{F_{factor}}$) used was too large(0.9), resulting in sizable jumps in the speed/voltage adjustment, causing deadlines to be missed. A lower adaptation factor was found to reduce the number of deadline misses.

In Fig. 13 we study the the impact of the update frequency (frequency at which the device sends out bulletins) on the number of deadlines missed for the R-DVS schemes using our real-time task set. Note that this does not affect the device level DVS. As the update frequencies decrease, the R-DVS schemes match the performance of D-DVS. However, at low update frequencies the network delay strongly affects the performance of the remote DVS schemes. Fig. 14 presents the effect of the threshold deadlines($\boldsymbol{D_{min}}$&$\boldsymbol{D_{max}}$) on the number of deadlines missed and energy gains over static slowdown, for the proposed R-DVS adaptation scheme. The graph indicates that at($\boldsymbol{D_{min}} = \mathbf{0.2}, \boldsymbol{D_{max}}$=0.4), the our adaptation matches the performance of D-DVS in both number of deadlines missed and energy saved over static slowdown. However, at higher ranges the device level DVS performs much better than R-DVS, saving a lot more deadlines while using nearly the same amount of energy.

We also studied the effect of the network delay on the power uptake, shown in Fig. 17. It was observed that the power saved reduced significantly as the network delay increased. Fig. 16 shows the effect of the k-weight on the number of deadline misses for the 3 task sets. As the k-Weight goes down the number of deadlines missed only goes up slightly. Even at 70% the algorithm shows favorable response to outlying data. As k-Weight varies the power saved over SSF is almost constant. Almost the same number of deadlines are missed so not much power is saved.

As seen above, our R-DVS scheme can closely match the performance of device level DVS. The energy gains seen from static slowdown varies between 40% to 80% depending on the nature of the task set considered. We show that using our remote adaptation strategies an additional gain of as much as 30% can be achieved over the gains due to static slowdown. This is similar to the the gains achieved through device level DVS. We now study the performance of R-DVS

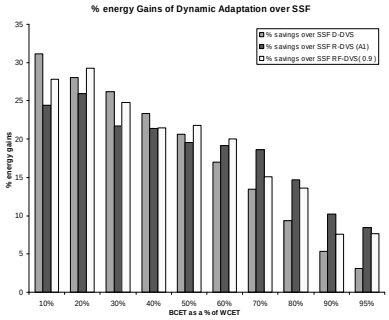


Fig. 9: Energy Gains over SSF

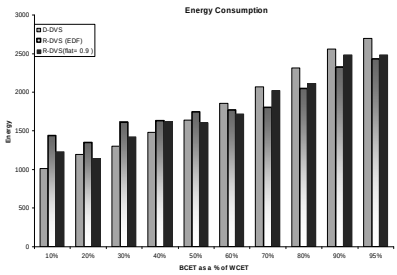
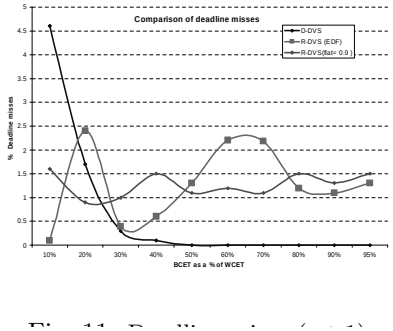


Fig. 10: Total Energy consumed



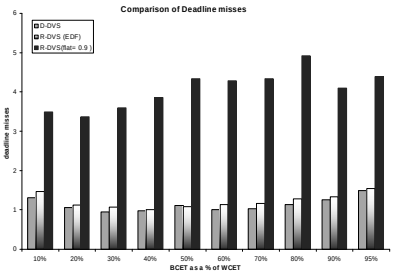


Fig. 12: Deadline misses(set 2)

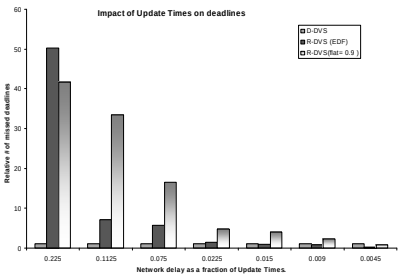


Fig. 13: Update Frequency

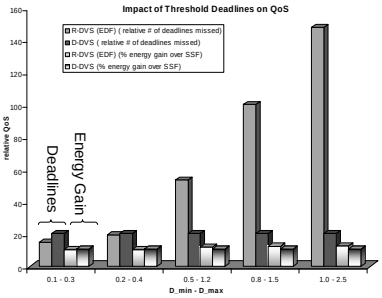


Fig. 14: Threshold Deadlines

in the presence of task partitioning and show how the remote adaptation using global system knowledge can provide significant improvements in performance and power.

5.2.3 Task Partitioning with DVS:

We present the energy gains achieved in the presence of both task partitioning and DVS(both Local-DVS and R-DVS). Fig. 18 and Fig. 19 show the relative number of deadline misses and the total energy consumed respectively, for a real-time task set(with initial util = .506). From Fig. 18 we see that the k-weighted adaptation scheme shows significantly less number of deadline misses over device level DVS and the FLAT adaptation schemes, while still maintaining a small gain in the total energy consumed. This shows that both the local-DVS and the FLAT adaptation schemes are unsuitable for dynamic task sets. They are inherently designed for static task sets and their performance suffers in the presence of changing task sets and network delays due to the partitioning. It is intuitive that for task sets having more cpu intensive (higher utilization) tasks, the partitioning scheme should show higher gains.

Overall comparison: Here we compare the energy gains over SSF and the corresponding number of deadline misses(Fig. 20) for three different real-time task sets with different initial cpu utilizations. For each policy(D-DVS,R-DVS) we study the performance in the presence of task partitioning and without task partitioning. In the figures, 'L' stands for Device(local)DVS, 'R' for R-DVS, 'NP' for "No partitioning", 'P' for partitioning, 'F' for FLAT and 'O' for our remote adaptation scheme. The first three sets of results in each graph indicate the performance in the absence of task partitioning. The energy gains are seen to be significantly higher in the presence of task partitioning and R-DVS adaptation scheme. There is also a significant reduction in the number of deadlines missed. This shows that a global adaptation strategy, with its knowledge of overall system state can significantly outperform locally optimized algorithms. This is because local optimizations are oblivious of global changes therefore cannot avail the benefits of global adaptation.

Summary of performance results: We first present the results of our optimal task partitioning scheme, demonstrating that simple task partitioning without DVS can yield substantial power gains. We then compared the performance of our proposed R-DVS scheme with a device level DVS scheme. The results indicated that our scheme performs as well as the local DVS scheme with a proper choice of the threshold deadlines and network update frequencies, while being able

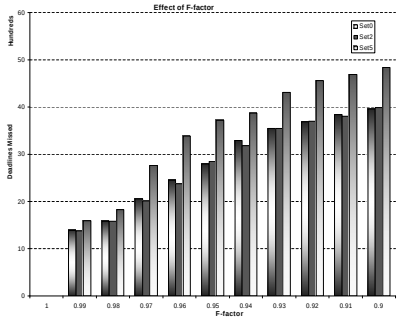


Fig. 15: F-factor on Deadlines

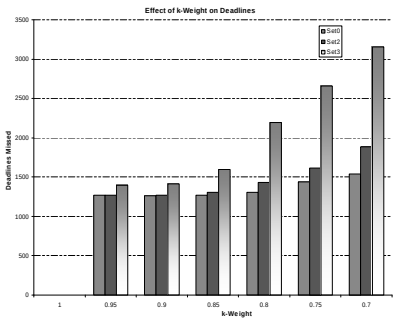


Fig. 16: k-weight on deadlines

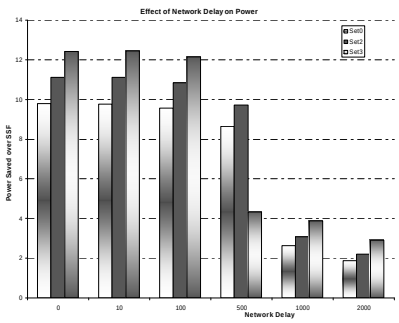


Fig. 17: Network Delay

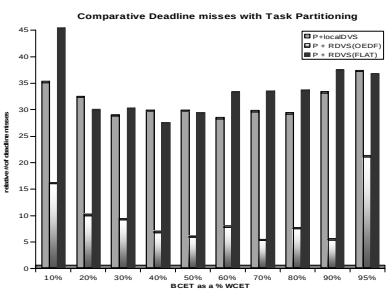


Fig. 18: R-DVS + partitioning

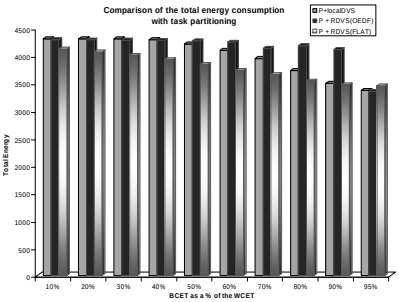


Fig. 19: R-DVS + partitioning

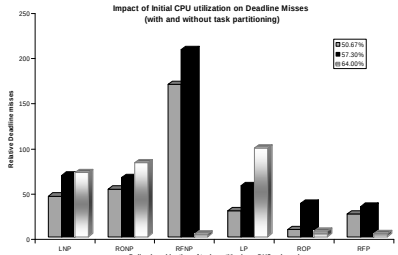


Fig. 20: Overall comparison

to dynamically changing task sets. We then evaluate both the DVS schemes in the presence of dynamic task partitioning. We show that in presence of task partitioning, our adaptation scheme performs better than the local DVS scheme. As the R-DVS scheme employs global knowledge of the system, it is able to combine the gains of task-partitioning and DVS effectively, to provide significant energy gains with smaller number of deadline misses. Additionally, the computation time for calculating an optimal EDF schedule was found to be significantly high and needs to be performed offline for low-power devices. Moreover, there is an inherent cost(power& delay) for switching voltage levels of the cpu. Dynamic device level schemes require the voltages to be adjusted for every task instance, which may not prove to be an overkill for soft real-time tasks. Our results indicate that a global system knowledge can be used to effectively combine an OS level dynamic voltage scaling with an application/middleware level task partitioning algorithm to provide substantial energy gains for low-power mobile devices, operating in ubiquitous environments.

6 Related Work & Future Research Directions

Current research and development efforts at the operating system level, have been focussed on techniques like dynamic voltage scaling (DVS) [21, 22, 9], and dynamic power management (DPM) [4]. The objective is to transition the enabled devices into low power dissipation states when the device is idle. Our work both contrasts and extends the dynamic voltage scaling technique to work effectively in distributed environments. The primary difference is that our approach does not require a DVS algorithm to be executing locally at the low-power device; we use a distributed feedback approach to fine tune the voltage scaling. Moreover, we control and direct task load on the low-power device for additional energy gains. Our future work includes integration of dynamic power management techniques into the power aware distributed middleware framework. In DPM, the primary assumption is that peripheral devices(e.g hard drives) are associated with single or multiple "power off" state(s).

The GRACE project [24] professes the use of cross-layer adaptations for maximizing system utility. They suggest both coarse grained and fine grained tuning through global co-ordination and local adaptation of hardware, OS and application layers.ECOSystem [25] is an OS level prototype that incorporates energy allocation and accounting mech-

anisms for various power consuming devices. In [23], a middleware framework for integrating soft real-time scheduling and DVS is presented. The idea is to apply DVS as far as possible, but while meeting resource reservation requirements of soft-realtime applications. It presents a resource reservation scheme that can be used by a middleware framework for effective processor allocation. Puppeteer [6] presents a middleware framework that uses transcoding to achieve energy gains. Using the well defined interface of applications, the framework presents a distilled version of the application to the user, in order to draw energy gains. PowerScope [7] is an interesting tool that maps energy consumption to program structure. It first profiles the power consumption and system activity of a computer and then generates an energy profile from this data. Odyssey [14] presents an applications aware adaptation scheme for mobile applications. In this approach the system monitors resource levels, enforces resource allocation and provides feedback to the applications. The applications then decide on the best possible adaptation strategy. In our approach we try to integrate the the positive aspects of all the three levels: OS, middleware and application. Application based adaptation will therefore enhance the performance of our framework. However, applications have to be specifically designed for the framework. [5] presents some drawbacks of present adaptive middleware systems. Current middleware technology relies mostly on application driven adaptation, providing notifications when system state changes. Adaptation entirely at the application level can lead to conflicts in applications implementing different adaptation strategies. This problem can largely be alleviated by using an effective middleware framework. [20] discusses some current reflective middleware techniques for component based middleware adaptations.

Offloading of tasks [18, 15] on to a remote machine has been widely studied. [18] shows that the task offloading can deliver significant energy savings over a noiseless wireless network channels, while the gains are offset over noisy communication channels. The use of proxies to aid mobile devices has also been explored before. [13] suggests presents a reconfigurable middleware solution that distributes middleware components between a proxy and a device dynamically. In [11, 10] a static task partition scheme is presented for offloading application subtasks onto a remote machine for energy savings. In this scheme, each application is divided into different computational tasks and the cost of execution and communication are optimized using a flow graph to yield an optimal partitioning of tasks. In contrast, our approach uses a dynamic task redistribution scheme that uses the residual power on the device as a parameter for determining the allocation of remote tasks. Moreover, our goal was to provide adaptations so that DVS could be used with dynamically changing application sets.

Concluding Remarks: In this paper, we presented a distributed middleware solution that can be used to achieve significant power gains for low-power devices in distributed environments. We introduced a remote dynamic voltage scaling adaptation scheme that uses global system state to adapt the static DVS technique for dynamically changing task loads. The power saving was further enhanced by optimally offloading expensive tasks to a remote proxy. Our extensive simulations show that our scheme provides significant benefits over the device level adaptation while showing improvements in both performance and power savings. We conclude, that for optimal power and performance deliverance in ubiquitous environments, adaptations based on global system knowledge provides several important benefits that cannot be availed by local optimizations. In future, distributed middleware will be the enabling technology for supporting lower-power mobile and embedded devices in ubiquitous environments. Distributed middleware frameworks hide the heterogeneity of underlying distributed environments. This, coupled with their scalability, flexibility and affinity to mobile and wireless architectures makes middleware an attractive technology for ubiquitous environments. It is therefore imperative that middleware technology adapt itself for performing advantageously in such environments.

References

[1] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. *"Network Flows: Theory, Algorithms, and Applications"*. Prentice-Hall, Englewood Cliffs, N.J., 1993.

[2] Chakraborty, S., and Yau, D. K. Y. "Predicting energy consumption of mpeg video playback on handhelds". In *In Proc. IEEE International Conference on Multimedia and Expo* (August 2002).

[3] Chatfield, C. *"The Analysis of Time Series, an Introduction"*. Chapman and Hall, 1975.

[4] Douglis, F., Krishnan, P., and Bershad, B. "Adaptive disk spin-down policies for mobile computers". In *2nd USENIX Symposium on Mobile and Location-Independent Computing* (April 1995).

[5] Efstratiou, Davies, C. K., and A, F. "Architectural requirements for the effective support of adaptive mobile applications". In *Middleware* (2000).

[6] Flinn, J., de Lara, E., Satyanarayanan, M., Wallach, D. S., and Zwaenepoel, W. "Reducing the energy usage of office applications". In *IFIP/ACM International Conference on Distributed Systems Platforms* (2001).

[7] Flinn, J., and Satyanarayanan, M. "Powerscope: a tool for profiling the energy usage of mobile applications". In *In Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications* (1999).

[8] Krintz, C., Wen, Y., and Wolski, R. "Application-level prediction of program power dissipation". Tech. rep., University of California, San Diego, 2002.

[9] Kumar, P., and Srivastava, M. "Predictive strategies for low-power rtos scheduling". In *ICCD* (2000).

[10] Li, Z., Wang, C., and Xu, R. "Computation offloading to save energy on handheld devices: A partition scheme". In *Proc. of International Conference on Compilers, Architectures and Synthesis for Embedded Systems* (November 2002).

[11] Li, Z., Wang, C., and Xu, R. "Task allocation for distributed multimedia processing on wirelessly networked handheld devices". In *in Proc. of IPDPS* (April 2002).

[12] Mohapatra, S., and Venkatasubramanian, N. "Optimizing Power using a Reconfigurable Middleware". Tech. rep., UC, Irvine, 2002.

[13] Mohapatra, S., and Venkatasubramanian, N. "PARM: Power-Aware Reconfigurable Middleware". In *ICDCS* (2003).

[14] Noble, B. D., Satyanarayanan, M., D.Narayanan, J.E.Tilton, and Flinn, J. "Agile application-aware adaptation for mobility". In *In Proceedings of the 16th ACM Symposium on Operating Systems and Principles, Saint-Malo, France,* (October 1997).

[15] Othman, M., and Hailes, S. "Power conservation strategy for mobile computers using load sharing". In *Mobile Computing and Communications Review* (January 1998).

[16] Pillai, P., and Shin, K. G. "Real-time dynamic voltage scaling for low-power embedded operating systems". In *In Proc. of the 18th ACM Symp. on Operating Systems Principles* (2001).

[17] Robinson, J., and et. al., S. R. "A task migration implementation for the message passing interface". In *HPDC* (1995).

[18] Rudenko, A., Reiher, P., Popek, G., and Kuenning, G. "Portable computer battery power saving using a remote processing framework". In *Mobile Computing Systems and Application Track of the ACM SAC* (February 1999).

[19] Suen, T., and Wong, J. "Efficient task migration algorithm for distributed systems". In *IEEE transactions on Parallel and Distributed Systems* (1992).

[20] Wang, N., Kircher, M., Parameswaran, K., and Schmidt, D. C. "Applying reflective middleware techniques to optimize a qos-enabled corba component model implementation". In *COMPSAC* (2000).

[21] Weiser, M., Welch, B., Demers, A., and Shenker, S. "Scheduling for Reduced CPU Energy". In *In Symposium on Operating Systems Design and Implementation* (1994).

[22] Y.Shin, and et.al. "Power optimization of real-time embedded systems on variable speed processors". In *CAD* (2000).

[23] Yuan, W., and Nahrstedt, K. "A middleware framework coordinating processor/power resource management for multimedia applications". In *IEEE Globecom* (Nov 2001).

[24] Yuan, W., Nahrstedt, K., Adve, S., Jones, D., and Kravets, R. Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems. In *MMCN-03*.

[25] Zeng, H., Ellis, C., Lebeck, A., and Vahdat, A. "Ecosystem: Managing energy as a first class operating system resource". In *ASPLOS-02*.