Energy-Efficient Policies for Embedded Clusters [*]

Ruibin Xu Dakai Zhu [†] Cosmin Rusu Rami Melhem Daniel Mossé

Computer Science Department, University of Pittsburgh
Pittsburgh, PA 15260
{xruibin,zdk,rusu,melhem,mosse}@cs.pitt.edu

Abstract

Power conservation has become a key design issue for many systems, including clusters deployed for embedded systems, where power availability ultimately determines system lifetime. These clusters execute a high rate of requests of highly-variable length, such as in satellite-based multiprocessor systems. The goal of power management in such systems is to minimize the aggregate energy consumption of the whole cluster while ensuring timely responses to requests. In the past, dynamic voltage scaling (DVS) and on/off schemes have been studied under the assumptions of continuously tunable processor frequencies and perfect load-balancing. In this work, we focus on the more realistic case of discrete processor frequencies and propose a new policy that adjusts the number of active nodes based on the system *load*, not system frequency. We also design a threshold scheme which prevents the system from reacting to short-lived temporary workload changes in the presence of unstable incoming workload. Simulation and implementation results on real hardware show that our policy is very effective in reducing the overall power consumption of clusters executing embedded applications.

Categories and Subject Descriptors D.4.7 [*Operating systems*]: Organization and Design - Real-time systems and embedded systems; D.4.8 [*Operating Systems*]: Performance - Measurements

General Terms Algorithms, Management, Experimentation

Keywords Dynamic Voltage Scaling, Load Balancing, Space Applications, Distributed Systems, Cluster Computing

1. Introduction

In portable or untethered devices that deal with large amount of requests, power consumption and cooling account for a significant fraction of the total operating cost. Furthermore, system overheat resulting from excessive power consumption can lead to intermittent system failures.

An example of the type of system that requires power management is satellite-based signal processing. Signal data collected through external sensors (equivalent to the front-end of a cluster) may be disseminated to several processing units for further analysis by a signal processing application. Currently, we are investigating two such signal processing applications, referred to as *SBT* (Subband Tuner) and *CAF* (Complex Ambiguity Function), each provided with several realistic traces. Based on the observation that a system designed for peak load is rarely fully utilized, applying power management schemes can successfully lead to significant power savings while maintaining adequate system performance.

Power management mechanisms can be divided into two categories: vary-on/vary-off (VOVO) and dynamic voltage scaling (DVS). Node VOVO [14, 7] makes cluster nodes inactive (i.e., puts them in a lower-power mode such as sleep or off) when the incoming workload can be adequately served by a subset of the nodes in the cluster and makes nodes active again when the workload increases beyond the capacity of the active nodes.

On the other hand, individual power management at local nodes is possible because current power-efficient systems have management functions that can be invoked to choose among different power states for each component. An increasing number of processors [19, 2, 6] implement DVS, which can yield quadratic energy savings.

It has been shown that combining the VOVO and DVS mechanisms can achieve significant power savings for clusters [7]. However, the existing policies are based on the assumptions of continuous frequencies and cubic rule of the power-frequency relation, which do not hold in practice. Currently available commercial DVS processors only provide about 4-10 discrete operating frequencies and many of them do not comply with the cubic rule of the power-frequency relation. Furthermore, some processors have inefficient frequencies that must be eliminated [16].

In this paper, we propose a new power management policy for embedded clusters. Our policy applies directly to the case of discrete frequencies and does not make any assumption on the power-frequency relationship. Even though our policy achieves maximum power savings when the incoming workload is stable and balanced across the whole cluster, we do not rely on these assumptions[1]. Our proposed policy, to be implemented at the front-end of the cluster, assumes that each node in the cluster performs DVS independently, to meet a desired QoS requirement (e.g., a goal is to keep up with the rate of request arrivals). This is the case for many existing systems, such as the satellite-based multiprocessor systems that we are dealing with.

[*] This work has been supported by NSF grant ANI-0125704 and ANI-0325353.

[†] The author is now a faculty member in Computer Science Department of University of Texas at San Antonio. He can be reached at Computer Science Department, UTSA, San Antonio, TX 78249, email:dzhu@cs.utsa.edu.

[1] Our motivation, the *CAF* and *SBT* applications, do not at all comply with the assumption of a stable workload.

The remainder of the paper is organized as follows. We first present related work in Section 2. The system model and applications are in Section 3. We present the theoretical results on deciding the optimal number of active nodes in Section 4. Our new policy is presented in Section 5. Simulation and implementation results are reported in Section 6. We conclude the paper in Section 7.

2. Related Work

Dynamic voltage-scaling (DVS), which involves dynamically adjusting the voltage and frequency of the CPU, has become a major research area. Quadratic energy savings [19, 15] can be achieved at the expense of just linear performance loss. For real-time systems, DVS schemes focus on minimizing energy consumption in the system while still meeting the deadlines. Yao et al. [21] provided a static off-line scheduling algorithm and a number of on-line algorithms with good competitive performance, all for aperiodic tasks. However, the algorithms assume cubic rule of power-frequency relation and knowing the computational requirement of tasks a-priori, which do not apply in our case. Weiser et al. [18] recommended interval-based DVS algorithms. Govil et al. [10] proposed to separate the interval-based DVS algorithms into two parts: prediction and speed-setting. Prediction methods and speed-setting policies were extensively studied in [11, 13]. The interval-based DVS algorithms incur low overhead and are easy to implement. From our experience, they are very effective in terms of keeping up with the request arrival rate. Automatic DVS for Linux running in general-purpose computers with distinction between background and interactive jobs was presented in [8].

Power management has traditionally focused on portable and handheld devices. IBM Research broke with tradition and presented a case for managing power consumption in web servers [4]. Elnozahy et al. evaluated five policies which employ various combinations of DVS and node VOVO for cluster-wide power management in server farms [7]. Sharma et al. [17] investigated adaptive algorithms for dynamic voltage scaling in QoS-enabled web servers to minimize the energy consumption subject to service delay constraints. Aydin et al. [3] incorporated DVS scheduling of periodic task sets to partitioned multiprocessor real-time systems.

Among the related work, the policies in [7] for power management in clusters are most relevant to our work. These policies determine the number of active nodes and apply a certain DVS scheme independently or coordinately. A vary-on/vary-off coordinated voltage scaling policy (VOVO-CVS) resulted in most power savings. For every possible number of active nodes, the policy precomputes an on-frequency and an off-frequency such that if the average frequency of the cluster exceeds the on-frequency, a sleeping node will be turned on and if the average frequency of the cluster falls below the off-frequency, one of the active nodes will be turned off. The policy precomputes the on-frequencies and off-frequencies based on the assumptions of perfect load-balancing, continuous frequencies and cubic power functions. In practice, these assumptions do not hold, and thus the policy is suboptimal.

3. System Model and Applications

3.1 Models

The system we consider is a cluster (Figure 1) consisting of a front-end and N identical nodes, each equipped with a DVS processor. At any given time, each node is in one of three states: active, idle, and inactive. When a node is active, its processor is running at some frequency f, where f is between a maximum frequency f_{max} and a minimum frequency f_{min}. Without loss of generality, we assume that $0 \leq f_{min} \leq f_{max} = 1$, that is, we normalize the frequency values with respect to f_{max}. We consider both the ideal case, where the frequency can be tuned continuously, and the realistic case, where the CPU only provides a finite number of discrete frequencies. The node's power consumption when active is P_{active}, which is dependent on its operating frequency f. The node's power consumption when idle (i.e., the processor is in the idle state and memory/bridge are in doze mode) is P_{idle}. The node's power consumption when inactive is $P_{inactive}$, which is the power consumed when the processor is off and the memory is in self-refresh mode. We consider the ideal case because its analysis can give insight into power management policies for clusters.

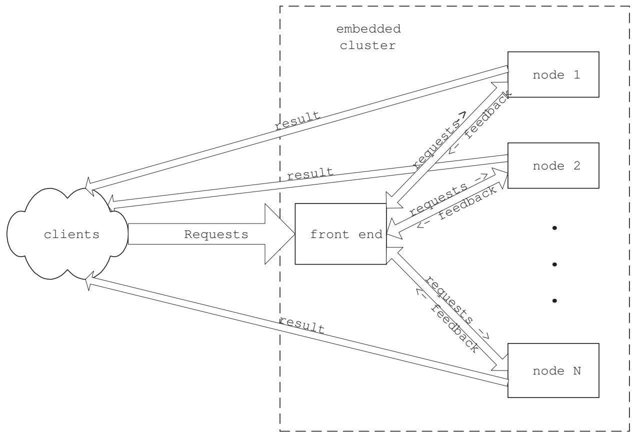


Figure 1. Cluster architecture

The front-end is responsible for collecting requests from clients and for distributing the requests to the active nodes. Neither the request arrival rate nor the computational requirement for each request is known *a priori*. Due to this characteristic of the systems under consideration, we express the system load as the amount of work (in cycles) that the front-end receives in one second. The front-end is also capable of making the nodes inactive and bringing them back to active state (that is, VOVO), according to some policy. The front-end tries to distribute the incoming requests among the active nodes in a balanced fashion.

Each active node carries out DVS independently, running at the lowest frequency that keeps up with the request arrival rate. Nodes service the requests that the front-end sends to them and return the results directly to the clients. Nodes also send feedback to the front-end, such as speed changes. Individual node policies were investigated in [15].

A note on load balancing: While load balancing makes perfect sense if the CPU speed can be tuned continuously, it is not energy optimal for the discrete case, where load *unbalancing* may be desired. For example, for a two-node cluster where the CPU only provides two speeds: 100 MHz and 200 MHz, if the system load is 300 MHz, it is optimal to distribute 100 MHz of system load to one node and 200 MHz to the other, instead of distributing 150 MHz to each of the nodes. However, because the exact load is unknown (i.e., execution times and the arrival rate are not known a-priori), such a load unbalancing mechanism is impractical to implement. In contrast, load balancing techniques have been well studied and are effective even when the incoming load is unstable (e.g., sending the next request to the node with the lowest load) [5].

3.2 Space Applications

As mentioned in Section 1, two digital signal processing applications are our motivation in this work, namely *SBT* and *CAF*. These applications were provided by our industrial partners, who are moving their embedded platforms from a application-specific processor (in this case a DSP) to a general-purpose processor (in this case a PowerPC 750 with DVS capability). For these applications, the system can determine the request *type*, upon request arrival. *Type* is the sort of semantic information that helps improving the predictions about the workload. In general, several types can be associated with the application processing the request.

For the *CAF* application there are three event types: the first time an event occurs, the second time and all times after that (typically resulting in long, short to medium, and relatively short events, respectively). For *SBT*, there are two types of requests (long and short). In these applications, the request type can be determined solely from the header.

SBT is an application that searches digital signal data that is related to frequency and time domain for certain patterns. It uses filters for finding contiguous chunks of data that have a specific characteristic for a certain interval of time. After finding such patterns, there is some processing that occurs. There are two possible paths to be followed, depending on the type of the event. From our own measurements, in about 19% of the events, there are not enough details to quickly extract the correct data and thus there is extra processing that is incurred. For the other 81% of the events, the data is sufficient for quick processing.

CAF is an application that collects data in low orbiting satellites (LEOs), correlates it with data collected from geo-stationary satellites (GEOs), for object recognition and location. *CAF* processing is done in the LEO through calculations of the difference between arrival time (dT) and frequency (dF) signals from the object of interest. This object may be on Earth's surface or may be flying. The *CAF* application can determine an object's location with an accuracy from 4 to 7 significant digits (corresponding to 1K to 16K data point correlation, respectively).

We measured the execution times of the requests in an experimental platform consisting of PowerPC 750 boards, using real-life inputs and inter-arrival times. We noted that the variability of execution lengths was extremely large (2 to 750 million cycles in *SBT* and 1 to 5,000 million cycles in *CAF*) and the predictability of the execution length was very poor. Our measurements are shown in Tables 1 and 2. From the execution of the applications, we generated realistic traces and used them in our evaluation of the system (see Section 6).

Table 1. Request execution times (in millions of cycles)

	SBT		CAF		
	Type 1	Type 2	Type 1	Type 2	Type 3
Min	2.9	2.0	8.2	4.1	1.3
Max	82.6	753.6	5045	210.2	32.9
Avg	9.7	123.2	820.2	45.0	5.8
%	79%	21%	5.4%	2.9%	91.7%

Table 2. Request inter-arrival times (in seconds)

	SBT		CAF
	81 min	1030 sec	1800 sec
Min	0.13	0.1	0
Max	6.7	11	5
Avg	0.37	0.44	0.7
events	13045	2307	2564

4. Determining the Optimal Number of Active Nodes

In this section, we present the theoretical results on how to determine the optimal number of active nodes based on the system load, assuming that the incoming workload is stable and balanced across the active nodes in the cluster. We consider the ideal and realistic cases (continuous and discrete speeds).

4.1 Continuous Case

Suppose that the CPU frequency/speed can be changed continuously. In this section we follow the power model used in [7]. In this model, the node power consumption when active (running at frequency f) is

$$P_{active}(f) = c_0 + c_1 f^3$$

where c_0 and c_1 denote the static power and the maximum dynamic power respectively. A node consumes the static power as long as it is active or idle, which includes the power consumption of all components except for the CPU, plus the base power consumption of the CPU. The dynamic power is determined by the CPU operating frequency and the maximum dynamic power is the dynamic power consumed when CPU is operating at the maximum frequency. For the purpose of analysis, we assume that $P_{inactive} = 0$ and that clock gating is used when the processor is put to the idle mode, thus resulting in $P_{idle} = c_0$. If $P_{inactive} \neq 0$, then this value can be subtracted from c_0 and the analysis results still hold.

The authors in [7] derived the policies turning on/off nodes based on the frequency of active nodes. In our approach, we decide the optimal number of active nodes based on the system load. Although both approaches solve the same problem, our approach enables the system to react much quicker to workload change ($O(1)$ vs. $O(N)$). Furthermore, our approach can identify the sufficient conditions when it is best to minimize the number of active nodes and when it is best to maximize the number of active nodes, as will be discussed below (see Results 1 and 2).

Let x be the normalized system workload ($0 < x \leq 1$, normalized with respect to the maximum workload that the system can handle, i.e., $Nf_{max} = N$). Thus, the minimum number of active nodes that can handle the workload x is $n_{min} = \lceil xN \rceil$. Let n be the number of active nodes, then the frequency of the processor of each node is $\frac{xN}{n}$. If $\frac{xN}{n} \geq f_{min}$, the total power of the whole system for load x, denoted by $p_1(n, x)$, is

$$p_1(n, x) = n \left(c_0 + c_1 \left(\frac{xN}{n} \right)^3 \right)$$

If $\frac{xN}{n} \leq f_{min}$, the frequency $\frac{xN}{n}$ can be emulated by running $\frac{\frac{xN}{n}}{f_{min}}$ ($= \frac{xN}{nf_{min}}$, also called *utilization*) of the time at frequency f_{min} and idle for the rest of the time. Thus, the total power of the whole system for load x, denoted by $p_2(n, x)$, is

$$\begin{aligned} p_2(n, x) &= n \left(c_0 + c_1 f_{min}^3 \frac{xN}{nf_{min}} \right) \\ &= nc_0 + c_1 f_{min}^2 xN \end{aligned}$$

Therefore, the total power of the whole system for load x, denoted by $p(n, x)$, is

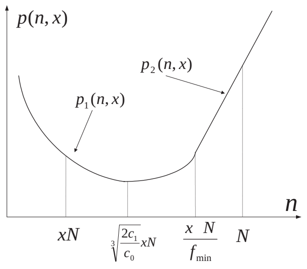
$$p(n, x) = \begin{cases} p_1(n, x) & \text{if } n \leq \frac{xN}{f_{min}} \\ p_2(n, x) & \text{otherwise} \end{cases}$$

The problem of deciding the optimal number of active nodes, n, can be expressed as a mathematical program:

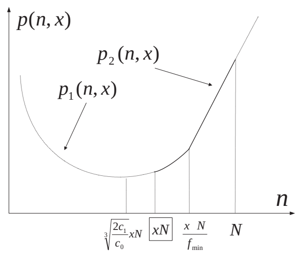
```
Minimize      p(n, x)
Subject to    xN ≤ n ≤ N
              n ∈ {1, 2, ..., N}
```

We solve the above mathematical program by first solving the continuous version of the program, that is, removing the constraint $n \in \{1, 2, \ldots, N\}$. The function $p(n, x)$ (Figure 2a) is composed of parts of two functions: $p_1(n, x)$, a convex function with the only critical point $\sqrt[3]{\frac{2c_1}{c_0}}xN$ (i.e., the point where $p_1(n,x)$ achieves its minimum), and $p_2(n,x)$, a linear function with positive slope. Note that $p_1(n) = p_2(n)$ when $n = \frac{xN}{f_{min}}$, which may be inside the range $[xN, N]$ or $[N, \infty]$. It is easy to verify that the solution to the continuous version of the program (see Figure 2) is

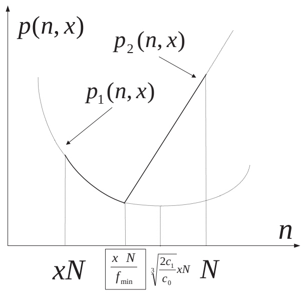
$$\hat{n} = \begin{cases} xN & \text{if } \frac{2c_1}{c_0} \leq 1 \text{ (Figure 2b)} \quad (1) \\ \frac{xN}{f_{min}} & \text{if } x \leq f_{min} \text{ and } \frac{2c_1}{c_0} \geq \frac{1}{f_{min}^3} \text{ (Figure 2c)} \quad (2) \\ N & \text{if } x \geq f_{min} \text{ and } \frac{2c_1}{c_0} \geq \frac{1}{x^3} \text{ (Figure 2d)} \quad (3) \\ \sqrt[3]{\frac{2c_1}{c_0}}xN & \text{otherwise} \quad (4) \end{cases}$$



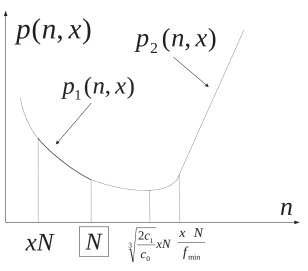
a. General plot of $p(n,x)$ for a given x



b. $p(n,x)$ when the minimum occurs at xN



c. $p(n,x)$ when the minimum occurs at $\frac{xN}{f_{min}}$



d. $p(n,x)$ when the minimum occurs at N

Figure 2. Function $p(n,x)$.

The optimal solution n^* to the discrete version[2] of the mathematical program is either $\lceil \hat{n} \rceil$ or $\lfloor \hat{n} \rfloor$, subject to $xN \leq n^* \leq N$. This is easily found by a simple comparison.

From the solution to the mathematical program, we have the following results:

Result 1: *If $\frac{2c_1}{c_0} \leq 1$, the minimum energy consumption is attained when the number of active nodes is just large enough to handle the system load (from condition (1)).*

Result 2: *If $\frac{2c_1}{c_0} \geq \frac{1}{f_{min}^3}$ and the load distributed to each active node is no less than f_{min}, the minimum energy is attained when the maximum number of nodes is used (from condition (3)).*

Results 1 and 2 are intuitive, as follows. Result 1 states that, when the static power dominates the dynamic power ($c_0 \geq 2c_1$, which satisfies condition (1)), it is better to put as many nodes inactive as possible to save the static power, as long as the number of the active nodes is enough to service the workload. Result 2 says that, when the incoming workload is reasonably large ($x \geq f_{min}$) and the dynamic power dominates the static power ($c_1 \geq \frac{c_0}{2f_{min}^3}$, which implies that $c_1 \geq \frac{c_0}{2x^3}$, satisfying condition (3)), it is better to use as many active nodes as possible to lower the dynamic power of each node.

[2] While considering the cubic rule, take n to be integer

4.2 Discrete Case

Today's processors only provide a few number of discrete speeds (that is, the continuous speed assumption is not valid) and the cubic rule does not usually apply in practice [20].

In the discrete case, when a node is active, the processor runs at one of the M discrete operating frequencies: $f_1, f_2, \ldots, f_M$ and the corresponding node powers are $P(f_1), P(f_2), \ldots, P(f_M)$.

In the presence of discrete speeds, if the incoming workload for one particular node is w where $0 \leq w \leq 1$, it will need to run at the frequency $\|w\|$ where $\|w\|$ denotes the lowest available discrete frequency higher than or equal to w. Thus, the utilization of the node is $u = w/\|w\|$ and the power consumption of the node is $u \times P(\|w\|) + (1-u) \times P_{idle}$.

As above, if x is the normalized system workload, the minimum number of active nodes that can handle the workload x is $n_{min} = \lceil xN \rceil$. Let n be the number of active nodes, then the frequency of the processor of each node is

$$f = \|\frac{xN}{n}\|$$

and the utilization of each processor is

$$u = \frac{\frac{xN}{n}}{f} = \frac{xN}{nf}$$

and the total power of the whole system for load x, denoted by $p(n,x)$ is

$$p(n,x) = (uP(f) + (1-u)P_{idle})n + P_{inactive}(N-n)$$

Thus, determining the optimal number of active nodes for a particular system load x is equivalent to finding the value of n which minimizes $p(n,x)$, that is, computing function $g(x)$ that returns the number of active nodes that minimizes $p(n,x)$.

An alternative solution to dynamically computing the optimal number of nodes is to precompute $g(x)$ offline and store its values in a table, which converts the online computation of $g(x)$ into a table lookup operation. This solution is feasible since $g(x)$ is a piece-wise constant function.

To see why function $g(x)$ is piece-wise constant, we first show that for a given x, $0 \leq x \leq \frac{n}{N}$, the function $p_n(x) = p(n,x)$ is a piece-wise linear function. We can express $p_n(x)$ in the following form, which does not contain the $\|\cdot\|$ operation:

$$p_n(x) = \begin{cases} \left(\frac{xN}{nf_1}P(f_1) + (1 - \frac{xN}{nf_1})P_{idle}\right)n + \\ \quad P_{inactive}(N-n) \quad \text{if } 0 \leq x < \frac{nf_1}{N} \\ \left(\frac{xN}{nf_2}P(f_2) + (1 - \frac{xN}{nf_2})P_{idle}\right)n + \\ \quad P_{inactive}(N-n) \quad \text{if } \frac{nf_1}{N} \leq x < \frac{nf_2}{N} \\ \quad \vdots \\ \left(\frac{xN}{n}P(1) + (1 - \frac{xN}{n})P_{idle}\right)n + \\ \quad P_{inactive}(N-n) \quad \text{if } \frac{nf_{M-1}}{N} \leq x \leq \frac{n}{N} \end{cases}$$

Each component of $p_n(x)$ is a linear function in x, so $p_n(x)$ is a piece-wise linear function. Thus $p_n(x)$ can be expressed by M straight line segments geometrically. Each function $p_n(x)$ ($n \in \{1, 2, \ldots, N\}$) divides the system load range $[0,1]$ into $O(M)$ pieces, and all of them together divide the system load range $[0,1]$ into $O(MN)$ parts. Corresponding to each of these $O(MN)$ parts, there are $O(N)$ straight line segments, each corresponding to one of the functions $p_n(x)$. These $O(N)$ straight line segments will divide each part into $O(N)$ pieces. For each of these pieces, we can find its corresponding optimal number of nodes in $O(N)$ time. The algorithm runs in polynomial time and the number of pieces for $g(x)$ is $O(MN^2)$. To decide the optimal number of active nodes, we perform a binary search which takes $O(\log(MN^2))$.

While the above algorithm is quite involved, we can instead use a simple approximation algorithm. This algorithm discretizes the system load range $[0, 1]$ into b bins of equal width. For a particular cluster, we precompute $g(y)$ where $y \in \{0, \frac{1}{b}, \frac{2}{b}, \ldots, 1\}$ store the values in a table H with $b+1$ entries. This algorithm runs in $O(bN)$ time. Once the cluster is running, for a given system load x, we let $g(x) = H[\lfloor bx \rfloor]$, that is, finding $g(x)$ for a system load x can be done in constant time. The good value of b is dependent on the system and can be decided by offline experiments.

Both the exact and approximation algorithms are offline algorithms, resulting in a table (to be stored at the front-end) that decides for each load what the optimal number of active nodes is. At runtime, the front-end periodically detects the system load, performs a table lookup to determine the optimal number of active nodes, and compares with the current number of active nodes to decide whether it should turn on/off nodes.

5. Power Management Policy

In this work, our goal is to minimize the aggregate energy of the whole cluster (excluding the front-end). We assume that each node in the cluster performs DVS independently to keep up with the request arrival rate [15]. Thus, we focus on the front-end algorithms, that is: how to estimate the system load, when to turn nodes on and off, and how to distribute requests to active nodes.

5.1 System Load Estimation

We determine the number of active nodes based directly on the system load, which can be obtained by getting feedback from each node. This is in contrast with the policies in [7], which turn on/off nodes based on the frequency of active nodes. The assumption in [7] is that the active nodes are not idle most of the time (continuous frequencies) and that they are all running at the same frequency (perfect load-balancing).

However, we argue that the frequency of a node at a given time poorly correlates with its actual load (unless the load is well balanced and the frequency is continuous), thus resulting in a poor estimation of the average frequency needed. For example, for a CPU with only 2 discrete speeds, 0.5 and 1, if the incoming workload is 0.25, the CPU would run at frequency 0.5 half of the time and idle half of the time to keep up with the workload. In this case, using any instantaneous frequency of the node to estimate the load will result in overestimation or underestimation.

In our policy, we determine the number of nodes needed based directly on the load in the recent past (rather than the node frequencies). This results in a more accurate estimation and eliminates the strong dependency on perfect load balancing. We also perform this decision without the assumption of continuous frequencies. The front-end will record the history of recent speed changes of each node, which means that each node needs to send the history of its speed changes as feedback to the front-end. For each node, the front-end computes the average load (different from the average frequency because idleness is regarded as frequency zero in computing average load, while the CPU is operating at some frequency when idle) over the past $look\_back$ seconds, where $look\_back$ is a system parameter. The summation of the average loads of all nodes is the estimation of the system load.

We also need to deal with a special case in our load estimation scheme. The average load is at most 1 because it is normalized to $f_{max} = 1$. If a node's average load is 1, it has been running at full speed without being idle over the past $look\_back$ seconds. However, average load of 1 does not indicate whether the node is overloaded or simply keeping up with the request arrival rate. Underestimating the system load would result in increased response times or even in system failures. To be on the safe side, if a node's average load remains at 1 for a predefined period of time, we increase that node's average load from 1 to $1 + \frac{1}{b}$ (for the meaning of b, see Section 4.2), resulting in an increase on the estimation of the system load, and thus possibly also in the change of number of active nodes.

The estimation of the system load and deciding the number of active nodes based on the system load are the highlights of our policy. Therefore, we call our policy Load-Aware On-off with independent Voltage Scaling (LAOVS).

5.2 The Threshold Scheme

After the front-end obtains the estimation of the system load x, it computes the number of active nodes needed, $n_o = g(x)$, as discussed in Section 4. Let the current number of active nodes be denoted by n_c. If $n_o = n_c$, there is no need to make any nodes active/inactive; if $n_o > n_c$ and $n_c < N$, make an inactive node active; if $n_o < n_c$ and $n_c > 1$, make an active node inactive. To be conservative, we do not make more than one node active or inactive at a time.

In real-life workloads, there may be some short-lived temporary workload changes. This may force the front-end into making a node active or inactive if we do that once the front-end detects a workload change. To prevent this from happening, we design a threshold scheme. We define two variables: $shut\_down\_threshold$ and $turn\_on\_threshold$. The variable $shut\_down\_threshold$ is the time the front-end will wait before it is sure to make a node inactive. Similarly, $turn\_on\_threshold$ is the time the front-end will wait before it is sure to make a node active. During the waiting time, the front-end will continue tracking the system load. At the end of the waiting time, an active node will be made inactive only if the decision to turn off a node is true every time it was checked during the waiting time. Similarly, an inactive node will be made active only if the decision to turn on a node was true every time it was checked during the waiting time.

5.3 Workload Distribution

Since neither the request arrival rate nor the computational requirement for each request is known a-priori, the front-end approximates the load-balancing by sending the next request to the active node with the lowest average frequency over the past $look\_back$ second(s) [5].

To decide which node to become active or inactive, we number all the nodes from 1 to N and use a variable $active\_number$ to keep track of the number of active nodes. To simplify the tracking of the status of the nodes, our policy is to keep nodes $1, 2, \ldots, active\_number$ active and node $active\_number + 1, \ldots, N$ inactive. The value of $active\_number$ is at least 1 which means node 1 is always active. Under this policy, when we want to make a node active, node $active\_number + 1$ is the target; when we want to make a node inactive, node $active\_number$ is the victim. This implementation was preferred because of its great simplicity, with complexity of $O(1)$.

Table 3. IBM PowerPC 750:frequency and system total power (measured)

f(MHz)	idle	4.125	8.25	16.5
P(mW)	1150.0	1150.0	1369.0	1811.0
f(MHz)	33	99	115.5	132
P(mW)	2661.0	4763.0	5269.0	6533.0

Table 4. Intel XScale: frequency and system total power (datasheets [19])

f(MHz)	idle	150	400
P(mW)	355.0	355.0	445.0
f(MHz)	600	800	1000
P(mW)	675.0	1175.0	1875.0

6. Evaluation

6.1 Theoretical Power Consumption

First, we compare the power consumption of our proposed policy, LAOVS, and VOVO-CVS policy under the conditions of stable load and perfect load balancing (the plots here are the theoretical results from Section 4). We used many different power models, but we only show the results of IBM PowerPC 750 and Intel XScale due to lack of space. The power consumptions for the Intel XScale (Table 4) and IBM PowerPC 750 (Table 3) were obtained from datasheets [19] and from our actual measurements, respectively. A constant power of 275mW was included for the XScale system, simulating an Infineon Mobile-RAM (very similar results are obtained for the XScale model with different values for the constant power, such as 2W).

Inefficient frequencies [16] cause some lower frequencies to have higher energy consumption (combined with less performance) than other higher frequencies. We applied the method in [16] and eliminated (for PowerPC 750 system) the 16.5MHz, 33MHz, and 99MHz frequencies. Unless otherwise noted, for all experiments, the number of nodes in the cluster is $N = 8$. Similar results are obtained for different values of N, but are not shown for lack of space.

Figures 3 and 4 show the number of processors and the total system power consumption as a function of the system load, assuming perfect load balancing. We also show the minimum number of processors and the corresponding power consumption, for comparison. Considering the system load rather than the frequencies of the individual systems and eliminating the cubic and the continuous assumptions result in LAOVS outperforming the VOVO-CVS policy. To better illustrate the limitations of the continuous assumption, we performed the following experiment: we artificially eliminated the $400MHz$ and $800MHz$ frequencies of the XScale model (the continuous assumption is more inaccurate for fewer discrete frequencies). The disadvantage of VOVO-CVS becomes more evident with fewer frequencies (see Figure 5).

An interesting result from our experiments is that the number of active nodes does not necessarily increase with the system load, as seen in Figure 4. This is somewhat counterintuitive and is due to the fact that frequencies are discrete. VOVO-CVS cannot capture this behavior, as it assumes continuous frequencies. Another interesting result is the jump of VOVO-CVS to 8 processors at load < 0.1 in Figure 3. To explain why, we start by noting that the PowerPC 750 processor has no efficient frequencies between 8.25MHz and 115.5MHz. If the only active node receives a relatively large workload (but no greater than what can be handled by a 115.5MHz CPU), it would run at frequency 115.5MHz in order to keep up with the workload (due to the gap between 8.25MHz and 115.5MHz). Worse yet, the front-end will use 115.5MHz as the system frequency, which is greater than all on-frequencies in Table 5. This will make the front-end turn on another node; this will happen continuously until all the nodes in the cluster are on.

When taking the actual load into account, this anomaly does not happen.

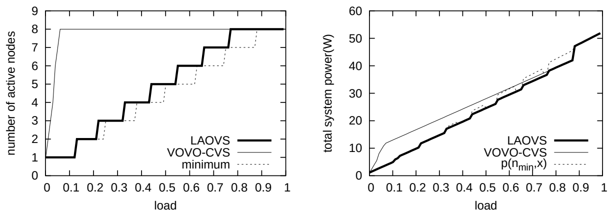


Figure 3. IBM PowerPC 750 System with 8 nodes (theoretical results)

Table 5. On-frequencies and off-frequencies used in VOVO-CVS for the PowerPC 750

#Nodes	Off-frequency(MHz)	On-frequency(MHz)
1	4.125	86.8516
2	43.4258	76.1867
3	50.7911	72.0717
4	54.0538	69.8786
5	55.9028	68.5138
6	57.0948	67.5821
7	57.9275	66.9055
8	58.5423	66.3916

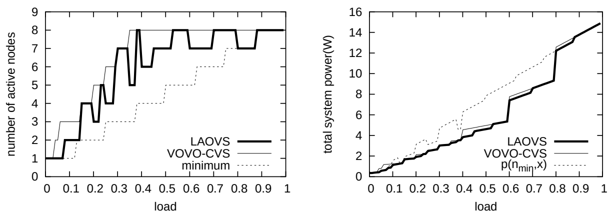


Figure 4. Intel XScale System with 8 nodes (theoretical results)

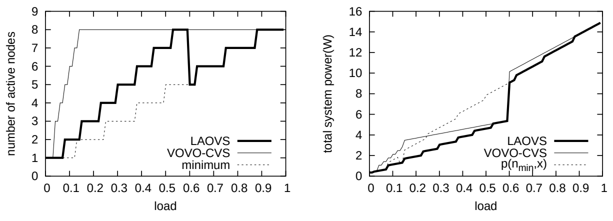


Figure 5. 8-node Intel XScale System without the frequencies $400MHz$ and $800MHz$ (theoretical results)

6.2 Simulation Results

We implemented three request distribution policies—Round-Robin (RR), VOVO-CVS, and LAOVS—in our simulator [1] designed to investigate power management schemes for rate-based systems. In RR, all the available nodes are always kept active and requests are distributed in a round-robin fashion.

Each simulated node is a PowerPC 750, as shown in Table 3. Many system parameters in the simulator are tunable. Since the applications under consideration (i.e., *SBT* and *CAF*) can tolerate average delay of 5-10 seconds, we set the following parameters. In each node, the processor speed is adjusted every 1 second based on CPU utilization over the past 1 second. Nodes take 33 seconds (boot time) for the inactive-to-active transition and the energy consumed during boot time is $190J$ (these numbers are from the PowerPC 750 platform, but we examine this issue further below: it has an important influence on the schemes). For the front-end, it checks the system load every 1 second, deciding whether to turn on or off a node. The variable $look\_back$ is set to 4, which means the front-end computes the average load over the last 4 seconds. Both $shut\_down\_threshold$ and $turn\_on\_threshold$ are 4 seconds.

To validate the theoretical power consumption for VOVO-CVS and LAOVS, we generated a synthetic workload in which the requests lengths (in cycles) and the request arrival rate are constant. This will result in perfect balance in load distribution by the front-end. We generated various such workloads for different size (N) clusters. The performance of LAOVS is never worse than the other two policies (see Figure 6). In most of the experiments, LAOVS outperforms RR and VOVO-CVS, both of which use the same number of nodes (both plots in Figure 6 are very similar; the raw data shows that the number of nodes is the same). Note that the energy savings are higher at low system loads because when the system load is high, most of the nodes have to be active to keep up with the request arrival rate, independent of the request distribution technique. However, when the system load is low, various number of active nodes can service the load and LAOVS will choose the optimal number. The values of $look\_back$, boot time, and thresholds have very little significance in the case of the synthetic workload.

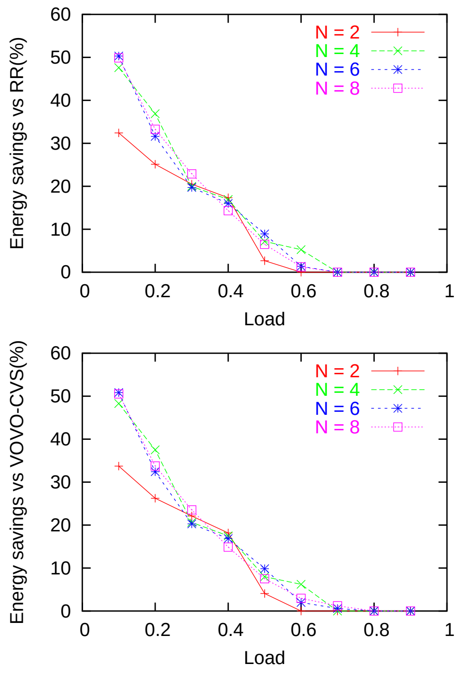


Figure 6. Energy savings of LAOVS as a function of load, compared to RR (top) and VOVO-CVS (bottom)

The above experiments assume that the load is perfectly balanced across the active nodes. While this assumption may be safe for some applications, our two applications (*SBT* and *CAF*) have much more unpredictable behavior, as shown in Tables 1 and 2. As mentioned before, perfect load-balancing is not a realistic assumption for such workloads. In our experiments even the type information provided by the application was not useful, due to the huge variability in the event processing requirements within a single event type. The simulation results shown in Figure 7 are based on a 41-minute trace of *SBT*, with average request arrival rate of 26%. We used this trace to test RR, VOVO-CVS, and LAOVS policies for $N = 2, 4, 6, 8$ nodes respectively and scaled the trace accordingly so that the cluster can handle the peak load with N processors. In all experiments, LAOVS outperforms RR and VOVO-CVS in terms of energy by 6-14%. The energy savings are significant despite being not as much as for the stable load; this is because in the realistic trace, there are some very long requests that make the load unbalanced and thus consume more energy and because in these experiments the impact of the energy consumed during boot time is significant.

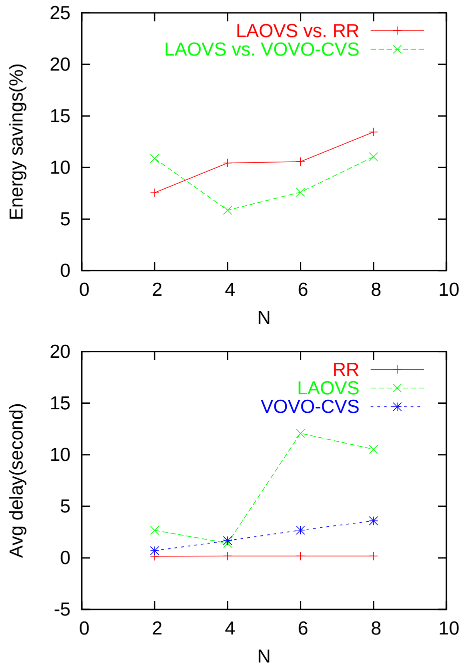


Figure 7. Simulation results on the 41-minute trace of *SBT*, with a boot time of 33 seconds (delay for RR is close to zero, since it never turns off any nodes)

Clearly, the time and energy spent during boot will have a significant impact on the effectiveness of any on-off scheme. For our PowerPC 750 platform, boot time is long (33 seconds) due to copying and uncompressing the operating systems from SUROM to main memory (SDRAM). The long boot time results in higher delay in LAOVS than VOVO-CVS. However, non-volatile memory is currently being rapidly developed, such as FRAM [9] and carbon nanotube memories [12]. When using such technology, the boot time will be drastically reduced, to the millisecond range. The same holds true for the energy consumed during boot time. We varied the boot time in our simulator and Figure 8 shows the energy savings and average delay versus boot time. From Figure 8, we can see that LAOVS outperforms VOVO-CVS both in terms of energy and average delay for boot times below 20 seconds.

We also looked into how summing up the average loads of all nodes to estimate the system load would improve VOVO-CVS. In Figure 8, VOVO-CVS1 is the modified VOVO-CVS with the use of summation of all average loads of all nodes to estimate the system load. As we can see from the figure, VOVO-CVS1 achieves a significant improvement over VOVO-CVS, but is still outperformed by LAOVS in general; one exception when the boot time is 30 seconds and the size of the cluster is 4. The exception is because we did not consider the time and energy during boot in our model; in that case, LAOVS turns on servers more often than VOVO-CVS1 and the gain from using the optimal number of servers and the threshold scheme cannot offset the loss of the energy during boot. However, when the boot time is below 20 seconds, LAOVS is always better than VOVO-CVS1 in terms of energy and average delay.

6.3 Experiments on A Testbed

As a proof of concept, to build a real system in a hardware platform, we implemented our scheme on a testbed consisting of IBM PowerPC 750 nodes. This platform, provided by our industrial partners with RR as the request distribution policy at the front-end, only had 2 nodes available at the time of testing. Our goal was not to do extensive evaluation in comparison to other schemes (see previous section), but to implement a proof-of-concept system, to take actual measurement, and to evaluate power management policies to a certain extent. Our collected data can be seen through our web interface [1].

The voltage/frequency scaling of PowerPC 750 processors is done through external voltage/frequency regulation. The voltage scaling takes around $2ms$ for each $0.1V$ adjustment and the frequency scaling takes negligible amount of time. Linux is used in the nodes and the power management (DVS and on/off) is supported through APIs. To power on/off a node, programs just need to send out a specific message to a host machine that has been wired to all nodes. Powering off a node takes into effect immediately while powering on a node (booting from ROM including uncompressing the kernel) takes 33 seconds. The power consumption of each part in a node (e.g., processor, memory and I/O) was measured and collected using a data acquisition system.

In our experiments, we used an additional node to be the front-end, which emulates the sensors that generate events; we report results of the 41-minute trace of *SBT*. The front-end sends events to nodes every second. On each node, a self-adaptive DVS scheme is employed, which uses the utilization information during the last interval (i.e., one second) to determine the frequency/voltage level for next interval. For our LAOVS policy, whenever a node changes its voltage/frequency or becomes idle, a feedback message is sent to the front-end.

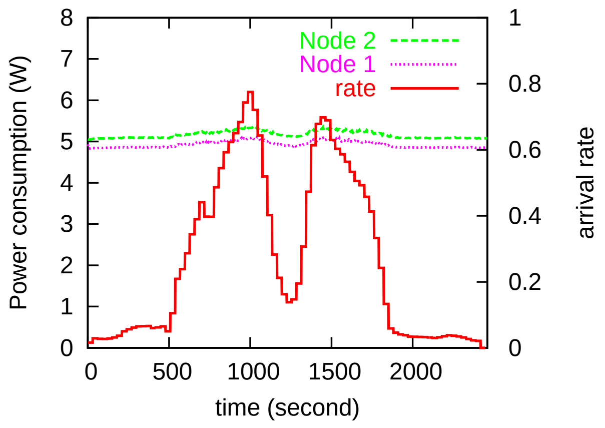


Figure 9. The power consumption for PowerPC 750 system with RR being used at the front-end

First, we show the results when the RR policy is used at the front-end and both nodes are always kept on. The solid line in Figure 9 is the normalized arrival rate (Y-axis on the right side of the plot) of events used in our experiments. Notice that the arrival rate is a rough indicator of system load. The arrival rate of 1 corresponds to the maximum number of events that can be handled by two nodes at the maximum voltage/frequency level. The average arrival rate of this trace is 26%.

The other two curves in Figure 9 are the power consumption for each PowerPC 750 system when there is no power management (NPM) being employed on each node. From the plots, we can see that the power consumption of PowerPC 750 systems under NPM is independent of event arrival rate (i.e., system load).

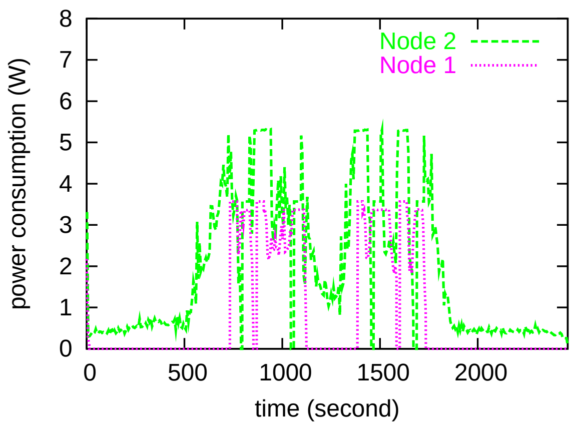


Figure 10. The power consumption for PowerPC 750 systems with LAOVS being used at the front-end

Figure 10 shows the power consumption for each PowerPC 750 system when LAOVS is used at the front-end. The power consumption of 0 corresponds to the node being powered down during that period. When DVS is employed on each node, Figure 10 shows that using DVS in each node consumes much less power than not using any power management. However, we can also see that nodes may be mistakenly powered down due to the inaccurate estimation of system load based only on the feedback of load information from each node (network communication delay is one of the factors). The solution to this problem is left for future work.

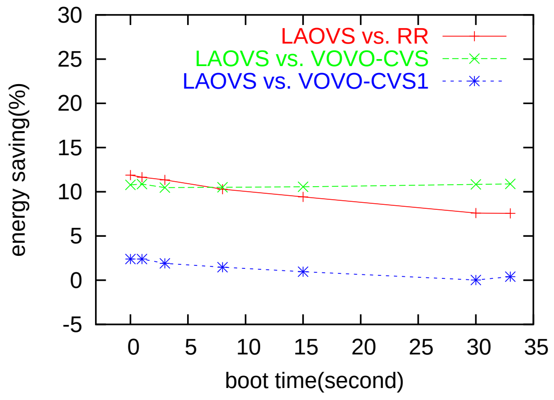
Table 6. System energy consumption for different front-end distribution and node DVS policies, normalized to RR/NPM

Policies	Energy(kJ)	$\%$
RR/NPM	37.7	100.0
RR/DVS	13.4	35.5
LAOVS/NPM	26.3	69.5
LAOVS/DVS	10.7	28.4

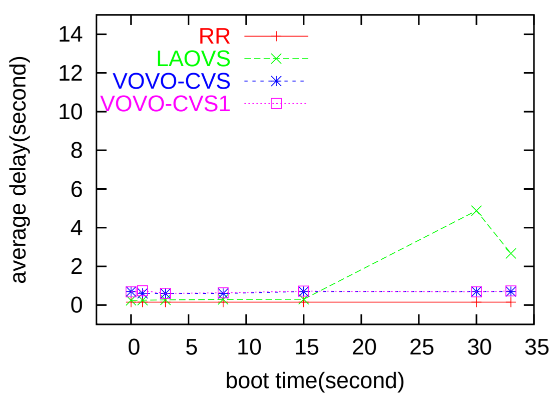
Table 6 shows the total energy consumption in the system when different policies are employed. We can see that only 69.5% energy was consumed by LAOVS compared to RR when NPM is employed on nodes. When DVS is used on each node, LAOVS uses around 7% less energy compared to RR.

7. Conclusions

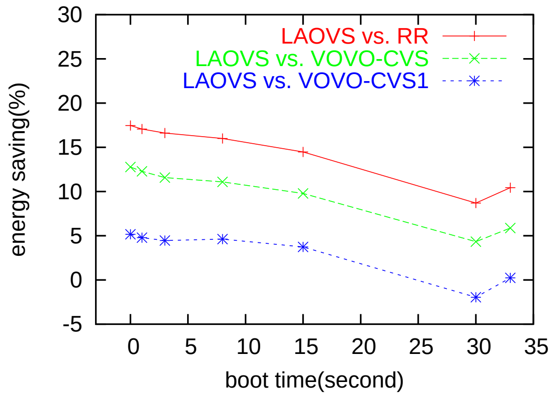
In this work, we studied energy-efficient policies for embedded applications using clusters. We proposed a new power management policy that is implemented on a front-end node, which carries out



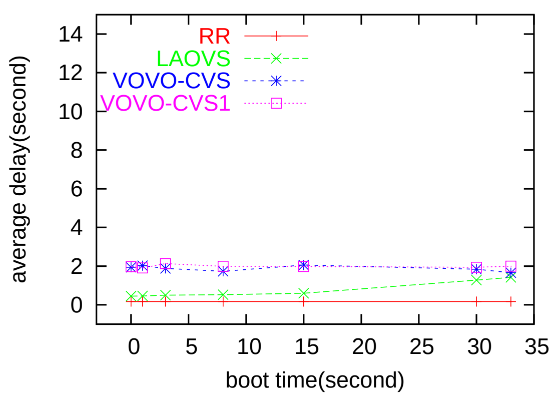
a. $N = 2$, energy savings



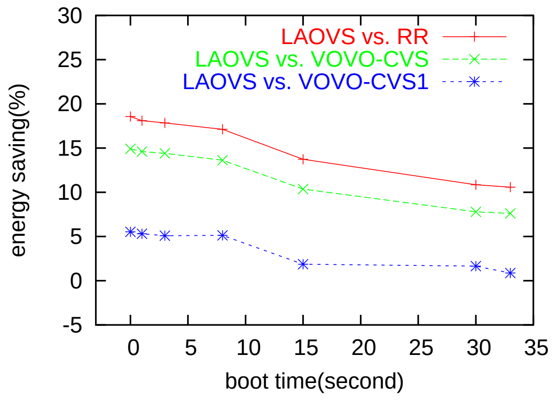
b. $N = 2$, average delay



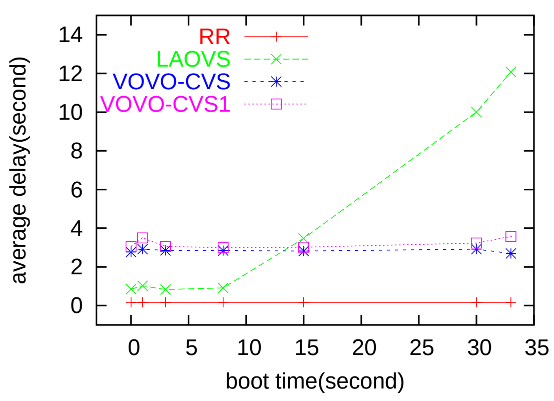
c. $N = 4$, energy savings



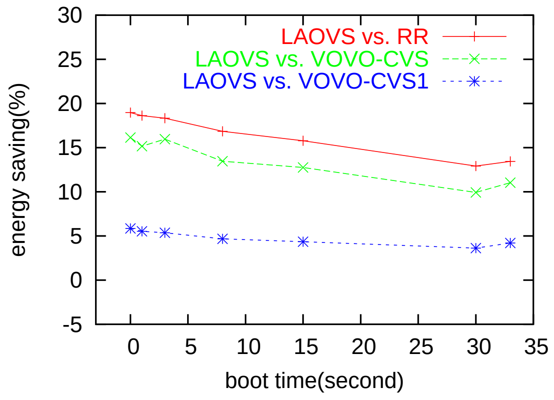
d. $N = 4$, average delay



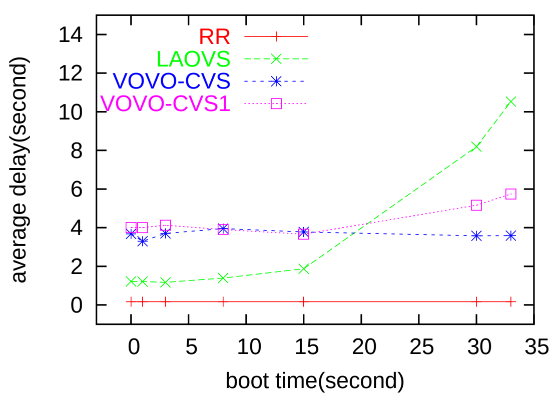
e. $N = 6$, energy savings



f. $N = 6$, average delay



g. $N = 8$, energy savings



h. $N = 8$, average delay

Figure 8. Simulation results for various boot times

the request/load distribution. The main novel aspects of our policy are that it has been designed directly for processors with discrete frequencies, does not make any assumption on the power-frequency relationship, accounts for the idle power of nodes when making decisions about nodes being active or inactive, and includes a threshold scheme to prevent the system from reacting to short-lived temporary workload changes in the presence of unstable incoming workload.

We implemented our policy in our suite of simulators [1], and, for proof of concept, in a testbed at the site of one of our industrial partners. The testbed consists of PowerPC 750 nodes. Simulation and implementation results show that our proposed policy is effective and outperforms previously proposed policies.

Unlike VOVO-CVS, LAOVS can be extended to heterogeneous clusters, which is part of our ongoing work. Future work will also investigate how to dynamically adapt the LAOVS system parameters to a changing workload.

References

[1] Power-sim. http://www.cs.pitt.edu/PARTS/PACC/NEW.

[2] AMD PowerNow! Technology. http://www.amdboard.com/powernow.html.

[3] H. Aydin and Q. Yang. Energy-Aware Partitioning for Multiprocessor Real-Time Systems. In *Workshop on Parallel and Distributed Real-Time Systems(WPDRTS'03)*, pages 113–121, March 2003.

[4] P. Bohrer, E. Elnozahy, M. Kistler, C. Lefurgy, C. McDowell, and R. R. Mony. The case for power management in web servers. In *Power Aware Computing, Kluwer Academic Publications*, 2002.

[5] H. Bryhni, E. Klovning, and O. Kure. A Comparison of Load Balancing Techniques for Scalable Web Servers. In *IEEE Networks*, pages 58–64, July 2000.

[6] Transmeta Corporation. LongRun Technology. http://www.transmeta.com/crusoe/longrun.html.

[7] E.N. Elnozahy, M. Kistler, and R. Rajamony. Energy-Efficient Server Clusters. In *Workshop on Power-Aware Computer Systems (PACS'02)*, 2002.

[8] K. Flautner and T. Mudge. Vertigo: automatic performance-setting for Linux. In *Proceeding of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, December 2002.

[9] FRAM: Ferroelectric RAM. http://www.fujitsu.com/downloads/MICRO/fma/pdf/framppt_tech.pdf.

[10] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU. In *Proceedings of the 1st ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pages 13–25, 1995.

[11] D. Grunwald, P. Levis, K.I. Farkas, C.B. Morrey III, and M. Neufeld. Policies for Dynamic Clock Scheduling. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.

[12] http://www.nanotech-now.com/.

[13] E. Pering, T. Burd, and R.W. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. In *Proceedings of 1998 International Symposium on Low Power Electronics and Design*, August 1998.

[14] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Workshop on Compilers and Operating Systems for Low Power*, September 2001.

[15] C. Rusu, R. Xu, R. Melhem, and D. Mossé. Energy-Efficient Policies for Request-Driven Soft Real-Time Systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, Catania, Italy, July 2004.

[16] S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, May 2003.

[17] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Liu. Power-aware QoS Management in Web Servers. In *Proceedings of the 24th IEEE Real-Time systems Symposium (RTSS'03)*, Cancun, Mexico, December 2003.

[18] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of the 1st Symposium on Operating Systems Design and Implementation*, November 1994.

[19] Intel XScale Microarchitecture. http://developer.intel.com/ design/intelxscale/benchmarks.htm.

[20] R. Xu, C. Xi, R. Melhem, and D. Mossé. Practical PACE for Embedded Systems. In *Proceedings of the 4th ACM International Conference on Embedded Software*, Pisa, Italy, September 2004.

[21] F. Yao, A. Demers, and S.Shankar. A Scheduling Model for Reduced CPU Energy. In *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.