Power Minimization in QoS Sensitive Systems

Jennifer L. Wong †, Gang Qu‡, Miodrag Potkonjak†
† Computer Science Dept., University of California, Los Angeles, Los Angeles, CA 90095
‡ Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742

***Abstract*— Majority of modern multimedia and mobile systems have two common denominators: quality-of-service (QoS) requirements, such as latency and synchronization, and strict energy constraints. However, until now no synthesis techniques have been proposed for the design and efficient use of such systems. We have two main objectives: conceptual and synthesis. The conceptual objective is to develop a generic practical technique for the automatic development of on-line adaptive algorithms from efficient off-line algorithms using statistical techniques.**

The synthesis objective is to introduce the first design technique for quality-of-service (QoS) low power synthesis. We introduce a system of provably-optimal techniques that minimize energy consumption of stream-oriented applications under two main QoS metrics: latency and synchronization. Specifically, we study how multiple voltages can be used to simultaneously satisfy hardware constraints and minimize power consumption while preserving the requested level of QoS. The purpose of the off-line algorithm is three-fold. First, it is used as input to statistical software which is used to identify important and relevant parameters of the processes. Second, the algorithm provides buffer occupancy rate indicators. Lastly, it provides a way to combine buffer occupancy and QoS metrics to form a fast and efficient on-line algorithm. The effectiveness of the algorithms is demonstrated on a number of standard multimedia benchmarks.

***Index Terms*— Quality of Service, Low Power, Synchronization**

I. Introduction

A. Motivation

In the last decade, low power synthesis and optimization techniques received a great deal of attention. A variety of techniques have been proposed for all steps of the synthesis and compilation process. Combination of new logic families and circuits, smaller feature sizes, more power efficient architectures, power-aware CAD tools, and low power dynamic run-time policies resulted in a dramatic increase in energy efficiency. However, the power requirements of new product generations have been constantly challenging the limits of battery capacities.

An illustrative example of the technology or application power trends is the evolution of the wireless phone. The first generation of wireless phones is analog, the second is digital, which is currently prevailing. As mezzanine generation wireless phones with microbrowsers have just emerged. The imminently pending third generation includes powerful Internet access. After that they will include numerous new features encompassing streaming media. Laptops with wireless modems already provide this type of service. The analysis of communication and digital signal processing requirements indicates that each wireless phone generation increases computational requirements by at least two orders of magnitude. Therefore, a need for new power minimization techniques has been constant in wireless communications. In the last decade, low power optimization techniques received a great deal of attention.

The most popular mobile low power applications, such as audio and video, are stream-oriented. The nature of these applications impose a need for addressing the QoS requirements under energy constraints. Latency and synchronization are the most relevant QoS metrics in these types of applications. Our goal is to develop a spectrum of techniques and algorithms which minimize energy consumption under the most important QoS metrics. Specifically, we study how to use multiple voltage technologies to simultaneously satisfy hardware requirements and minimize power consumption, while preserving the requested level of QoS in terms of latency and synchronization. Our starting point is a provably optimal off-line algorithm for power minimization under QoS and buffer constraints. In addition to buffer occupancy, a crucial criteria for deciding which process to run at which voltage, we identified four key properties of streaming processes (latency slack, synchronization slack, relative burstiness, and number of tasks (samples)). We plot a 5xm-dimensional space, where m is the number of concurrent processes, and for each point in this space we use the off-line algorithm as an indicator of a correct decision with respect to the task and voltage selection. This space is analyzed to reduce context switching overhead and speed up the decision process.

B. Objectives

Our primary goal is to present competitive on-line algorithms for power minimization for streaming media applications for given hardware resource constraints: latency and synchronization, as well as context switching overhead. We aim to dynamically adjust the supply voltage in such a way that an incoming statistical stream of data does not overflow the buffer capacity of our processing system while expending the least amount of energy. By considering the long and short term statistics of the media streams and current buffer backlog, we decide which supply voltage to apply. Furthermore, by considering latency and synchronization constraints, we decide which task to schedule at the current moment. Finally, we use the new on-line algorithm to explore the trade-off between buffer size (cost) and energy consumptions.

The first step is the development of two provably polynomial time off-line algorithms for multiple voltage scheduling

of single and multiple processes. The algorithm orders and assigns packets of streaming media in such a way that energy consumption is minimized, while storage requirements are satisfied. The algorithm is dynamic programming-based and can be used for compile time scheduling of movies and audio or as starting point for the development of on-line algorithms for the same task.

II. Related Work

Our research results can be viewed in the context of four related areas: low power modeling and optimization, quality of service, on-line algorithms, and statistical techniques.

Mainly due to the demand for mobile applications, low power research has attracted a great deal of attention in the last decade. Both power modeling and optimization have been addressed on many levels of the synthesis process [26]. More recently, A number of researchers proposed the use of multiple voltages in order to reduce power consumption [10], [22], [27], [30]. Furthermore, several variable voltage techniques have been reported [16], [17], [34]. Numerous algorithms for dynamic priority real-time systems have been proposed including [21], [31], [18]. Also, several industrial multiple voltage low power designs have been reported [17], [3] and prototypes [5], [24], [38]. The common denominator in all the efforts has been the fact that the operations on the critical path are scheduled at higher voltage and therefore are executed faster and other operations are scheduled on a lower voltage and therefore the energy consumption is reduced. Another popular approach to power minimization is dynamic power management which aims to reduce the power consumption of electronic systems by selectively shutting down idle components [11], [2]. From one point of view our work can be interpreted as a combination of these two techniques, multiple voltages and system-level power management. A good survey on low power scheduling is [19].

The first QoS requirements, such as bounded delay, guaranteed resolution or synchronization have been addressed in the network and real-time operating systems (RTOS) communities. The most sound and practically relevant QoS model in the networking community was proposed by R. Cruz [13], [12]. The model assumes periodic segmentation of time. During each period, each process receives a task of generally varying complexity. The cumulative sum of tasks for a process forms a demand curve imposed on the system. The system serves the task sequentially by allocating resources during each time period to one of the processes. The cumulative sum of the processed data forms a service curve. The main conceptual result in RTOS literature was presented by Rajkumar et al. [28]. They introduced an analytical approach for satisfying multiple QoS dimensions under a given set of resource constraints. They proved that the problem is NP-hard and developed an approximation polynomial algorithm for the problem by transforming it into a mixed integer programming problem [29]. A comprehensive survey of QoS research in these two areas is given in [1]. Recently, the first efforts in QoS, and in particular synchronization during the system design process, has been reported in design automation literature [25].

A. On-line algorithms

The notion of an on-line algorithm was introduced in order to define a class of algorithms for which part of the input is unknown at the beginning of the algorithm execution. Most of the research is focused on competitive (worst-case) analysis of on-line algorithms [4], [15]. In particular, a strong emphasis is on how to define competitive analysis to better reconcile the theory and practice of on-line algorithms. A comprehensive survey on the main research areas for on-line algorithms can be found in [4].

B. Statistical Techniques

Statistical techniques can be broadly divided in two groups: parametric and nonparametric [33]. Parametric techniques assume that the knowledge about underlying statical distribution is available (often normal distribution is assumed) and that the task is to confirm the assumption about the distribution, calculate the corresponding parameters, and establish intervals of confidence [32]. Nonparametric techniques do not make any assumption about the statistical distribution. They aim to build the conceptually and quantitatively the simplest (and therefore best) model which fits the recorded data [6], [33]. There are a number of validation techniques, such as histograms, Chi-square tests, Kolmogorov-Smirnov test, and quantile-quantile plots which can be used to validate statistical claims of the model. We opted to use resubstitution techniques for validation [9], [14] because these techniques enable the validation of an arbitrary hypothesis while establishing and accurate confidence interval.

There are three main conceptual novelties in the presented research with the respect to the previous efforts. The first is that we consider QOS requirements for the streaming media task model. The second is that we have developed one or rare probably optimal algorithms for power minimization at the system level. The final novelty is that on-line algorithm is developed using statistical methods starting from the off-line algorithms and therefore they provide a generic paradigm for rapid development of effective on-line algorithms. Comprehensive versions of this paper can be found at [35], [36], [37].

III. Preliminaries

In this section, we outline the abstraction and models used for power consumption, and define latency, synchronization, and context switch overhead.

The dominating component of power consumption is the switching power. Switching power can be modelled as $P = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f$, where $\alpha \cdot C_L$ is the effective switching capacitance. This results from the fact that greater throughput comes with the cost of higher voltage. Specifically, the gate delay of circuits is a function of applied voltage and can be calculated using the formula $T = k(V_{dd}/(V_{dd} - V_t)^2)$ where k is a constant [7].

We assume the design operates using multiple voltage supplies and that the voltages change instantaneously with no overhead. These changes in voltages are assumed to happen only at the beginning or the end of a time unit. Furthermore,

we assume that the voltage units are selected in such a way that the use of two identical voltages, v_i and v_i, on two consecutive points is more effective than the use of consecutive voltage levels, v_i and v_{i+1}, due to the fact that power as a function of voltage is convex.

The Demand-Supply model for QoS was developed by Cruz et al. [12]. The model addresses the burstiness of QoS while handling resource allocation. This model assumes periodic segmentation of the time dimension. During each period, each process receives a task of generally varying complexity. The cumulative sum of tasks for a process can be depicted as a demand curve imposed on the system. The system serves the task sequentially by allocating resources during each time period to one of the processes. The cumulative sum of the processed data forms a supply curve. While the DS-Curve explains many of the QoS metrics, such as latency, backlog, and synchronization. The key problem with this DS-Curve is that its representation of QoS for a small period can be large.

The demand curve measures the burstiness of the service requirement. The service curve guides the resource allocation with QoS guarantees. Backlog is defined as the amount of demand that cannot be processed at that time point, and must then be carried over to the next time unit. The backlog in the QoS model is represented by the difference between the vertical positions of the demand curve and the service curve. Latency is the time between when the demand for a task arrives, and when it is processed. This is shown in the model by the horizontal difference between the demand and service curve at any given vertical position. A process is a program which assumes that it has independent use of the CPU. A process is long, in the sense that it consists of many tasks. These tasks are processed at periodic moments in time. With each task, we associate a processing time and a storage requirement. Note that periodicity is not reducing the generality of our formulation because the tasks of the process could have a requirement of zero. Finally, note that satisfying all latency implicitly implies that the throughput requirements are also satisfied.

Latency is defined as the difference between the time when the data is processed and the time the data arrived, i.e. t_p - t_a. We denote the time in which a particular sample (piece of date) arrives as t_a and the time when that piece of data is completely processed as t_p. At the intuitive level, synchronization indicates how well two or more processes are correlated in their execution. The assumption is that for each piece of data to be processed for one of the processes there exists a corresponding piece of data in each of the other processes which need to be processed at exactly the same time in order to have complete synchronization. However, majority of real life applications such as, movies playing with its video and audio processes do not have to be fully synchronized due to the imperfections of the human sensory system.

We define synchronization in the following way. For the sake of simplicity, consider only two processes, p_1 and p_2. We denote the tasks of the processes p_1 by p_{1i} and p_2 by p_{2j}, where $i, j = \{1, \ldots, n\}$. Perfect synchronization constraints indicate which sample (task or piece of data) of process p_1, which is denoted by p_{1i}, has to be executed at the same time as piece of data p_{2j}. Synchronization tolerance (often for the sake of brevity is solely called "synchronization") indicates the maximal amount of time by which the execution of fully synchronized samples p_{1i} and p_{2j} can maximally differ.

TABLE I

EXAMPLE OF SYNCHRONIZATION AND LATENCY FOR TWO TASKS

Time	0	1	2	3	4
Task Arrival					
P_1	a	b	c	d	-
P_2	w	x	y	z	-
Task Processed					
P_1	-	a	-	b	c,d
P_2	w	x	-	y	z

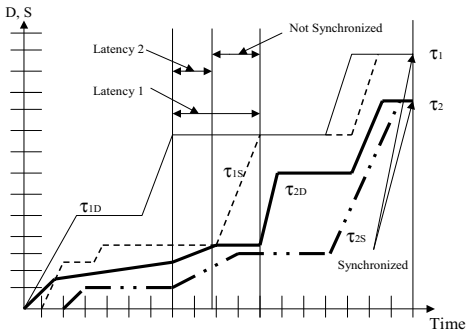

Fig. 1. Latency and synchronization in the DS model.

For example, consider the two processes shown in Table III. For each process, p_1 and p_2, there are four tasks which arrive one at each time unit. For process p_1, we have tasks 'a' through 'd', and for p_2, we have 'w' through 'z'. The latency, or the time between when a task arrives and is processed, of task 'b' is two units (it arrives at time one and is processed at time three). The synchronization between two tasks, for example 'b' and 'x' or p_{11} and p_{21}, is two. However, 'd' and 'z' or p_{13} and p_{23} is zero because both tasks are processed in the same time unit.

One approach for illustrating the Demand-Supply model and synchronization and latency is the use the DS-based approach. Synchronization is the timing relationship between interacting media, which is one of the most important metrics for QoS, such as jitter and burstiness. The synchronization of two processes can be seen in the DS-Curve (Fig. 1). The figure displays two tasks, τ_1 & τ_2, and their corresponding service (dotted lines) and demand curves (solid lines) for the two tasks. We say that τ_1 & τ_2 are synchronized when both τ_1 & τ_2 are serviced at the same time. The latency is the time between when the demand arrives and when it is processed. The amount in which tasks τ_1 & τ_2 are not synchronized is the difference between the latencies.

Note that the DS-Curve model is actually not an accurate abstraction of the media data delivery process [9]. The major limitation is that during delivery the initial periodic nature of tasks can be made either highly dense or very sporadic in short time intervals. Nevertheless, there is an easy way to make the DS-Curve model adequate. Essentially all that is needed to consider all tasks which arrive during our time unit (eg. in the case of MPEG 40ms) as a single task is to create a new tasks which has processing time and storage requirements equal to the sum of the processing times and storage requirements of all tasks which have arrived.

A context switch is the time overhead which is incurred by a multitasking kernel when it decides to process different tasks. The amount of context switching time dramatically depends on the processor. Context switching time for a typical DSP processor is fairly low, around ten cycles, while for a RISC processor it is much higher, approximately 100 cycles. In our experimentation, we used ten cycles.

IV. Off-line Optimal Algorithm

In this section, we formulate the off-line QoS low power problem and present our optimal algorithm. We use a single processor that can operate at multiple supply voltages. The goal is to service multiple processes with minimal energy consumption and the minimal amount of memory while meeting various QoS requirements.

A process consists of a sequence of tasks. With each task t_i, we associate:

- a_i: The arrival time, the time when a task is generated from the process and makes the CPU request;
- p_i: The time needed to complete this task at the nominal voltage v_{ref};
- s_i: The storage demand which is the minimal amount of memory to store this task on its arrival.

Each of the tasks may have QoS requirements such as latency and synchronization. Latency d_i is the time that task t_i has to be served after its arrival, that is, the actual finish time of task t_i must be earlier than $a_i + d_i$. Synchronization measures the interaction among tasks in different processes. We say that task t_i from one process and task t_j from another are k-synchronized if the difference of their finishing times is within k CPU units. We denote this by $syn(t_i, t_j) \leq k$.

The variable voltage processor has multiple supply voltages among which it can switch. The processor's processing speed varies as the voltage changes, so will the actual execution time for a task to receive its required amount of service. Suppose a task needs one CPU unit at the nominal voltage v_{ref}, then the execution time to accumulate the same amount of processing at voltage v_{dd} is given by [8]:

$$\frac{(v_{ref} - v_t)^2}{v_{ref}} \cdot \frac{v_{dd}}{(v_{dd} - v_t)^2} \tag{1}$$

where v_t is the threshold voltage.

Given n processes $\tau^1, \tau^2, \cdots, \tau^n$, each τ^k consists of a sequence of tasks $t_1^k, t_2^k, \cdots, t_n^k$. A *schedule* is a set consisting of the starting time, finishing time, and the voltage level for each task. A schedule is *feasible* if the processor starts each task after its arrival, finishes it before the latency constraint, and satisfies all synchronization requirements. The quality of a schedule is measured by its energy consumption and the memory requirement. Since these two metrics are non-comparable to each other, we introduce the concept of competitiveness. We say two schedules are *competitive* if neither outperforms the other in both energy consumption and memory requirement. We formulate the problem as:

On a processor with multiple voltages, for a given set of processes, find all the feasible competitive schedules.

We make the following assumptions:

Tasks from the same process have to be executed and completed in the FIFO fashion; The common processes that we consider are audio, video, and streams generated by sensor networks. For this type of applications, there is a natural intrinsic order in which consecutive tasks have to be executed. For example, obviously when we talk on the telephone it is required that speech packets are processed and presented to the user in exactly the same order as they are generated.

A task's processing demand, p_i, is proportional to its storage demand, s_i; In general, of course, this assumption is not necessarily correct. However, in many situations, the amount of processing required for a specific set of data is proportional to the amount of data. We mainly adopt this assumption for the sake of simplicity during presentation. However, it is important to notice that this assumption can be removed in a straight forward way without increasing the complexity of the optimal algorithm. This assumption is also needed in many practical situations where the main modeling obstacle is that often it is not possible to predict exactly the amount of time needed to process a particular amount of data. In many situations, it is often adequate to estimate the required computational effort as a linear function of the amount of data.

The memory occupied by a task can be partially freed, but only at the end of a CPU unitit occupies is freed and we can start immediately next task that arrives before the start of this unit of time [1] This assumption is a direct consequence of the way how current operating systems function. operating system updates the requirements of consumption for each type of resources only once per operating system cycle. Operating system cycle is an atomic period of time that is allocated by the operating system to a given task. In the modern operating systems, this cycle is usually long, several hundred milliseconds. Before a new cycle starts, the operating system makes decisions on the next action.

There is no context switching overhead; Obviously, in almost all types of processors, there is context switching overhead. The key observation is that the operating system cycle is significantly longer than the overhead and therefore in the first approximation the overhead does not have to be considered. Common times for context switching overheads are in milliseconds or less and as we just stated, operating system cycles are commonly at least two orders of magnitude longer. Another reason for this assumption is that it is necessary in order to produce the optimal algorithm. Note that the considered problem is NP-complete if context switching overhead is taken into account. Another important observation is that we use the optimal algorithm mainly as a lower bound and the assumption makes the lower bound more optimistic, and therefore all conclusions for our on-line algorithm are conservative.

The processor can instantaneously switch the supply voltage, but only at the beginning of each CPU unit; Physical laws in current technology imply that the change of supply voltage can not be done instantaneously. Recently, there

[1]Memory can be partially freed means that, for instance, if half of the processing demand is fulfilled at the end of one CPU unit, then we are able to free half of the space used to store this task. Our proposed algorithm can be easily modified when this is not allowed.

TABLE II

MEMORY REQUIREMENTS FOR THE MOTIVATIONAL EXAMPLE.

	0	1	2	3	4	5	6	7	8	9	10	11
0	4	8	17	17	19	17	14	11	8	5	2	0
1	7	12	12	14	10	7	4	1	0			
2	12	5	7	5	2	0						
3	5	5	1	0								
4	5	1	0									
5	1	0										
6	0											

TABLE III

STORAGE REQUIREMENTS FOR THE MOTIVATIONAL EXAMPLE.

	0	1	2	3	4	5	6	7	8	9	10	11
0	4	8	17	17	19	19	19	19	19	19	19	19
1	7	12	12	14	19	19	19	19	19			
2	12	12	12	14	14	14						
3	12	12	12	14								
4	12	12	12									
5	12	12										
6	12											

have been several efforts to take this time into account [23]. However, in modern technologies, context switching times are usually two to three orders of magnitude shorter than the operating system cycle. Therefore, this approximation is sound and impacts overall results only nominally. algorithm assigns a high priority to the reduction of context switching time. Another important observation is that this assumption again makes the lower bound more optimistic and therefore all our conclusions about the relative performance of the online algorithm more pessimistic.

A. Optimal Solution for Single Process

In this section, we show how to find all feasible competitive solutions for a single process. Suppose the reference voltage v_{ref} is $0.8v$, and there are two different voltage levels $v_{hi} = 3.3v$ and $v_{lo} = 1.8v$. From equation (1), we approximate the processing speeds to be 3 and 10 at v_{lo} and v_{hi} respectively. Consider a process with six tasks, $t_0, t_1, \cdots, t_5$. For simplicity, we further assume that task t_i arrives at time i, and there are no deadline constraints. Finally, we assume that the processing and memory requirement are **4, 7, 12, 3, 5**, and **1** respectively for the six tasks. The goal is to determine the voltage to use for each unit time, such that the energy consumption is minimized. Even for two different levels of voltages, an exhaustive search will take exponential time[2]. We developed a dynamic programming-based algorithm which achieves polynomial runtime for this task.

Table II shows the memory requirement at the end of each unit time, which is the minimal amount of memory required to store all the arrived but unfinished tasks. The table uses the time (in terms of CPU units) that the processor is operating at v_{lo} and v_{hi} to label the horizontal and vertical axis respectively. For example, entry (i, j) is the minimal memory requirement after running at v_{hi} for i CPU units and at v_{lo} for j units.

Consider entry (1,1), whose content is the storage we need at the end of the second unit of time after we use v_{lo} for one unit of time and v_{hi} for one unit. We can either apply v_{hi} in the first unit and v_{lo} in the second unit, or start at v_{lo} and switch to v_{hi} after one CPU unit. In the first case, since task t_0's processing demand is 4 and we are able to process 10 at v_{hi}, we will finish t_0, free the memory, and wait for t_1; then at v_{lo} in the next unit of time, we can finish 3 out of the 7 units of processing demand from t_1; now t_2 is arriving, therefore we need a total of $(7-3)+12 = 16$ units of memory to store t_1 and t_2. In the second case, v_{lo} then v_{hi}, we can only finish 3 out of the 4 processing demand of task t_0 by the end of the first unit of time due to the slow processing speed at v_{lo}; however, after raising the voltage to v_{hi} during the second unit, we are able to finish both the remaining of t_0 and entire t_1; the storage for tasks t_0 and t_1 are freed and therefore when t_2 arrives, we only need 12 units of storage to store this new task. Thus, we fill entry (1,1) with 12, the smaller storage requirement of the two different strategies.

Let $m(i, j)$ be the content of entry (i, j). We can reach this entry from entry $(i-1, j)$ by applying v_{hi} or from its left neighbor $(i, j-1)$ by applying v_{lo}, hence we have:

$$\begin{aligned} m(i,j) \quad = \min \quad & (s_{i+j} + \max(0, m(i-1,j) - sp_{hi}), \\ & s_{i+j} + \max(0, m(i,j-1) - sp_{lo})) \end{aligned} \tag{2}$$

where sp_{lo} and sp_{hi} are the processing speed at v_{lo} and v_{hi} respectively. The inner $\max$ is introduced to enforce that excess processing resource cannot be used for future work. We build Table II based on formula (2), where every row ends with an entry of 0 meaning that there are no tasks left.

While $m(i, j)$ gives the minimal storage requirement at the instant $i + j$, we may have used more storage already before this time. We further denote $M(i, j)$ as the minimal amount of storage that has been used up to time $i + j$ after running i units of time at v_{hi} and j at v_{lo}. Considering the voltage being used in the $(i + j)$-th unit, we observe that if we use v_{hi}, we can finish at most $\max(m(i-1, j), sp_{hi})$ and need a storage of $s_{i+j} + \max(0, m(i-1, j) - sp_{hi})$. Moreover, previously we have already required a storage at the amount of $M(i-1, j)$. This implies that

$$\begin{aligned} M(i,j) \geq \max(M(i-1,j), s_{i+j} + \\ \max(0, m(i-1,j) - sp_{hi}) \end{aligned} \tag{3}$$

Similar inequality holds if we use v_{lo}, therefore we have

$$\begin{aligned} M(i,j) = \min(\\ \max(M(i-1,j), s_{i+j} + \max(0, m(i-1,j) - sp_{hi}), \\ \max(M(i,j-1), s_{i+j} + \max(0, m(i,j-1) - sp_{lo})) \end{aligned} \tag{4}$$

Based on the recursive formulas (2) and (4), we calculate $M(i, j)$'s and store them in Table III, where the last entry of the i-th row gives the minimal storage requirement to complete all the tasks by using v_{hi} for exactly i units.

The power consumption at $v_{hi} = 3.3v$ is 1, then the power consumption at $v_{lo} = 1.8v$ is 0.1 from our power model. Unlike the storage requirement, energy consumption is path

[2]For example, if one best solution is to use high voltage for m units and low voltage for n units, then there exist $\binom{m+n}{m}$ different ways of finding m units and applying high voltages.

TABLE IV

MEMORY AND ENERGY FOR DIFFERENT SCHEDULES

	(6,0)	(5,1)	(4,2)	(3,3)	(2,5)	(1,8)	(0,11)
M	12	12	12	14	14	19	19
E	6.0	5.1	4.2	3.3	2.5	1.8	1.1

independent. I.E., it depends on the total number of CPU units that we have used at v_{lo} and v_{hi}, not the voltage at every individual time unit. For instance, if we have used v_{hi} for 2 units and v_{lo} for 4 units, then the total energy consumption is calculated as $1 \times 2 + 0.1 \times 4 = 2.4$.

Table IV gives the memory requirements and total energy consumptions by different scheduling policies, where (i, j) in the first row indicates a schedule that uses v_{hi} for i units and v_{lo} for j units. Clearly from this table, we see that there exist three competitive optimal solutions, $(4, 2)$, $(2, 5)$, and $(0, 11)$. They consume different amounts of energy and require different amounts of memory. We can then choose the one that fits our preference of memory and energy, and retrieve the actual schedule (i.e., the voltage for each CPU unit) by using simple backtracking.

Fig. 2 shows the algorithm of finding all the competitive optimal solutions for multiple processes. A schedule in this case has to determine, for each CPU unit, which process to be executed and at which voltage level.

Definitions: Assuming that there are m processes and k different voltages, we have $m \cdot k$ choices: running the ith process at voltage v_j ($1 \le i \le m, 1 \le j \le k$). A *state* $S = (e_1, \cdots, e_m; u_1, \cdots, u_k)$ means that the ith process has been allocated e_i CPU units, and the processor has been working at voltage v_j for u_j CPU units. Notice that $\sum_{i=1}^{m} e_i \le \sum_{j=1}^{k} u_j$ and the equality holds if and only if at any time, there exists unfinished process(es). We say state $S = (e_1, \cdots, e_m; u_1, \cdots, u_k)$ *precedes* $S' = (e'_1, \cdots, e'_m; u'_1, \cdots, u'_k)$ if $i) e'_i \ge e_i, ii) u'_j \ge u_j, iii) \sum_{i=1}^{m} e'_i - e_i \le 1$, and $iv) \sum_{j=1}^{k} u'_j - u_j = 1$. If S precedes S', we say that S' *follows* S. We define *Prev*$(S) = \{S' : S'$*precedes* $S\}$, and *Next*$(S) = \{S' : S \in$ *Prev*$(S')\}$. A state S is *reachable* if *Prev*$(S) \neq \phi$. A *final* state is a state when all processes' requests are satisfied. A schedule is a sequence of states $\{S_1, S_2, \cdots\}$ such that $S_l \in$ *Prev*(S_{l+1}) and all processes' processing loads are satisfied at the final state.

Correctness of the algorithm:

The off-line optimal algorithm consists of three phases. First, we build an $m \times k$ dimensional table which stores the minimal memory requirements. For example, when there is only one process and two voltages, then we will have a table like Table III. Step 1 computes the *Next* set for the initial state S_0, if all m processes require CPU time at the beginning, then this set will have $m \times k$ elements. Steps 2-4 makes all the states in *Next*(S_0) reachable, since each state is one move away from the initial state S_0. We denote the set of reachable states by $\mathcal{S}$. Steps 5-18 build the table recursively until there is no reachable state. We keep all the reachable states in a queue, we calculate the *Next* set for the head of the queue (state S) in Step 15, delete S from the queue and put all elements of *Next*(S) into the queue in Step 16. When we compute *Next*(S), we consider all the timing requirements. For example, if process i has a deadline at the end of next CPU unit and its remaining process requirement can be fulfilled only when we use the highest voltage, then *Next*(S) will contain only one state, which assigns the current CPU unit to process i and applies the highest voltage. Because all other schedules will fail to meet process i's deadline. The memory requirement for each state is calculated using formulas similar to (2,4).

Input: m processes with their arrival time, processing load, and other timing requirement(deadline, synchronization, etc.); k different supply voltages.
Output: All competitive pairs of memory requirement and energy consumption, and one schedule for each such pair.

Algorithm:
Phase I: Configuration for all states.
1. Compute *Next*(S_0) for the initial state S_0;
2. for each $S \in$ *Next*(S_0)
3. { *Prev*$(S) = S_0$;
4. $\mathcal{S} = \mathcal{S} \cup S$;}
5. while ($\mathcal{S} \neq \phi$)
6. { for each $S \in \mathcal{S}$
7. { current_max_memory for state $S = \infty$;
8. for each $S' \in$ *Prev*(S)
9. { calculate the max_memory requirement if S follows S';
10. if (max_memory $\le$ current_max_memory)
11. { current_max_memory = max_memory;
12. current_previous_state = S'; }
}
13. max_memory for state S = current_max_memory;
14. previous_state for S = current_previous_state;
}
15. Compute *Next*(S);
16. $\mathcal{S} = \mathcal{S} \cup$ *Next*$(S) - S$;
17. for each $S' \in$ *Next*(S)
18. *Prev*$(S') =$ *Prev*$(S') \cup S$; }
Phase II: calculation for energy consumption.
19. for each final state S
20. calculate the energy_consumption for S;
21. compute all the competitive final states $\mathcal{F}$;
Phase III: determine one schedule for each competitive state.
22. for each competitive state $S = (e_1, \cdots, e_m; u_1, \cdots, u_k) \in \mathcal{F}$
23. { index $= l = \sum_{j=1}^{k} u_j$;
24. $S_{index} = S$;
25. while (index $\neq 0$)
26. { S' = previous_state for S_{index};
27. index = index - 1;
28. $S_{index} = S'$; }
29. report the schedule $(S_0, S_1, \cdots, S_l)$ for S; }

Fig. 2. Algorithm for all off-line competitive schedules.

From the table built in the first phase, we can easily see the total memory requirement for each schedule, which is the value at its corresponding final state. In Phase II, we calculate their energy consumption. Recall that the energy consumption is path-independent. Let P_j be the power for voltage v_j, then for final state $S = (e_1, \cdots, e_m; u_1, \cdots, u_k)$, all schedulers with this final state will consume energy in the amount of $E = \sum_{j=1}^{k} P_j \cdot u_j$. So for each final state, we associate with the pair (M, E), the memory requirement and the energy consumption. Recall also that two final states S and S' are *competitive* if $i) M \le M'$ and $E \ge E'$, or $ii) M \ge M'$ and $E \le E'$.

In the third phase, we find a schedule for each competitive final state. We achieve this by using backtracking as shown in Steps 23-29. The existence of state S' in Step 26 is guaranteed by the way in which we build the memory requirement table in Phase I. Therefore, we have:

Theorem 4.1: The algorithm in Fig. 2 finds all the feasible

competitive schedules.

We analyze the complexity of the algorithm, for a fixed processor that has k supply voltages to execute m processes, in terms of the total processing demands. Suppose that we need X CPU units to service all the processes at the reference voltage. In Phase I, we essentially fill in the entries of an $m \times k$ dimensional table. (Table III is an example with $m = 1$ and $k = 2$), where entries along dimension $< i, j >$ represent the storage requirements when the i-th process is processed with the j-th voltage. If the i-th process needs $X_{i,j}$ units under the j-th voltage, when all the other $m \times k - 1$ dimensions are fixed, the number of entries we need to fill along this dimension will be $X_{i,j}$, which is in the order of $O(X)$. Since the table is $m \times k$ dimensional, and the cost for computing each entry is constant, the run-time in Phase I will be $O(X^{mk})$.

The calculation of energy consumption in Phase II takes constant time for each final states. According to how many units we have run at the reference voltage, it is clear that we will have at most X different final states (c.f. Table III for an example). Thus, the cost here is $O(X)$. In the last phase, we determine a feasible schedule for each competitive final state by backtracking. In step 27, we move one entry closer to the starting point, and the total number of steps we need is also in the order of $O(X)$. Therefore, we have:

Theorem 4.2: If we need X CPU units to service all the processes at the reference voltage, the run-time of the proposed algorithm is $O(X^{mk})$.

V. On-line Heuristics

In this section, we present the on-line algorithm for power minimization under the QoS constraints, synchronization and latency.

We have multiple on-line streaming processes, with tasks which arrive at periodic time intervals. For each task of each process, we have memory and CPU requirements. Each of these tasks have a given latency constraint, and on some subset of these tasks additional synchronization constraints are imposed. We are given multiple supply voltage levels in which to execute these tasks. The goal of the on-line algorithm is to decide which task from the stream processes to execute at each time interval and at which voltage in such a way that all latency and synchronization constraints are satisfied. Additionally, at no point of time the requirements for storage should exceed the memory size (buffer space).

In order to solve the overall problem, we must answer the following three questions: (i) how much buffer space is needed, (ii) which task to execute, and (iii) which voltage to apply. The answer to each of these questions is determined by our synthesis and on-line scheduling approach which is presented in Fig. 3. The on-line approach uses the optimal off-line algorithm to determine its decision mechanism.

The on-line approach begins with the assembly of a diverse set of test cases. The off-line optimal algorithm provides a lower bound on the memory requirement for the system along with the optimal QoS solution for the test set. The lower bound memory requirement is used to determine the proper buffer allocation size for the on-line algorithm. In this phase, a binary

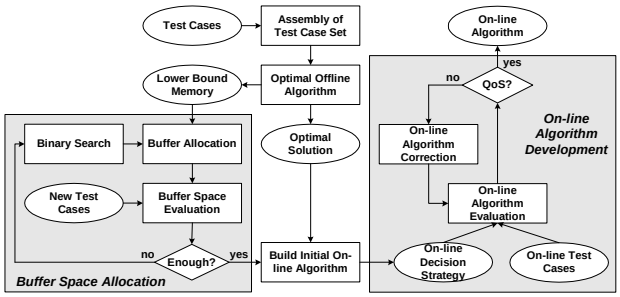

Fig. 3. Overall Flow for the creation of the on-line algorithm.

search on the size of the buffer is conducted. Each iteration tests the new buffer size on a new set of test cases, until the buffer space allocated is sufficient to handle all considered cases.

Next, the buffer size and the optimal solutions are used to build the on-line algorithm. The initial on-line algorithm builds a statistical model from the optimal off-line solutions and creates an on-line decision strategy, which is used in order to select the proper task and voltage in which to execute in each situation. The decision strategy is then evaluated on a set of on-line test cases. If the decision strategy does not provide the level of QoS specified, then modification of not only the statistical model and the decision strategy, but also the allocated buffer space is conducted. We continue to make modifications until the desired level QoS is reached.

The initial on-line algorithm is created using the pseudo-code shown in Figure 4. The initial on-line algorithm is created in five steps. In the first step, we identify the relevant properties for the QoS requirement. For example, in the case of latency and synchronization, we define properties such as average latency, maximum synchronization delay, and buffer occupancy. We evaluate the relevance of these properties in terms of the off-line optimal algorithm in the second step. We eliminate all properties which show little relevance to the outcome of the optimal off-line solutions. Following this step, both the optimal off-line solutions and the relevant properties are used to build the statistical model. We build a nxm-dimensional space, where n is the number of properties and m is the number of processes. The resolution of each property is specified, and for each subspace we determine the statistical values for task selection. Each subspace contains the percentage of time the optimal off-line algorithm selected each of the tasks under the defined property conditions. In a similar way, the statistical values for all situations and each voltage level is calculated.

Before we evaluate the effectiveness of the model, we have to develop the on-line decision strategy. The strategy is responsible for making the decision as to which task and which voltage to select according to the particular combination of property values. The strategy is reliant on the context switch time or penalty. For each subspace, in the task selection statistical model, the decision strategy must decide with task to select based on the values in the subspace and the context switch penalty. If there was no penalty for context switching, then for each situation we would select the statistically

```
Build On-Line Algorithm() {
1.    Identify Relevant Parameters;
2.    Evaluate Relevant Parameters;
3.    Build Statistical Model;
4.    Develop Decision Strategy;
5.    Validate Model;}
```

Fig. 4. Procedure for development of on-line algorithm.

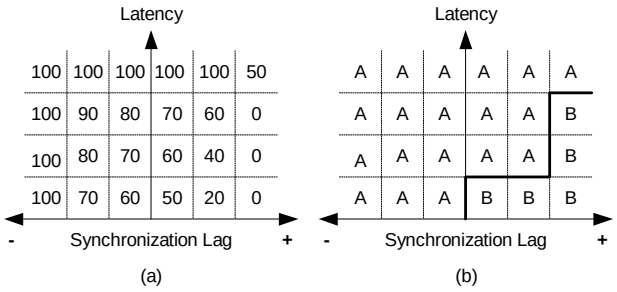

Fig. 5. Example of the statistical model and decision strategy of the on-line algorithm.

strongest task from the statistical model. However, if the context switch penalty is high, we would like to continue to run the tasks of the currently selected process as long as possible. In the moderate case, the proper time to switch between processes needs to be defined, and therefore we propose different points in the statistical model to switch between processes. The final step is to evaluate each of these proposed points to determine the proper switching point in the model. Once the proper switching points has been defined, we compact the nxm-dimensional table by combining subspaces with the same task/process selected to execute. Statistically, the subspaces should not be interleaved, and therefore we shall have continuous subspaces. For each decision the on-line algorithm determines which subspace the properties fall into, and select the assigned task/process to execute. The same process is applied to determining the voltage selection decision strategy. This on-line decision strategy is then passed on to the final stage of the overall on-line approach.

In order to better illustrate the algorithm and to make our ideas more tangible, we use a small example shown in Fig. 5. For the sake of clarity and brevity, we consider only two processes, A and B, and two properties, latency of process A and synchronization lag between processes A and B. The lag is positive if process A is in front of process B in terms of its execution. Fig. 5(a) shows numbers that can be obtained in principle by running the optimal off-line algorithm on a large set of examples. Each defined region of the space contains the percentage of cases in which the optimal off-line algorithm selected task/process A to execute. For example, the top left corner contains the value 100% and indicates that in all cases, process A was selected. The intuition is that process A has very high latency and is lagging behind synchronization with process B.

These numbers are used by the on-line algorithm development process to create the decision table shown in Fig. 5(b). Note that the decision strategy must take into account the context switching penalty, therefore not making the mapping from Fig. 5(a) to 5(b) straightforward. Specifically, the value 40% from Fig. 5(a) was mapped to the decision that process A should be executed in order to reduce context switching overhead. For example, if the latency of task/process A is high and the synchronization of A is behind B, we should run task A with a likelihood of 100%. However, in the case of task A with high positive synchronization lag, the likelihood of running task A is very low, despite the level of latency of task A.

The on-line algorithm builds a statistical model and an on-line decision strategy based on the nxm-dimensional space defined by the properties. The goal is to select properties which provide strong indication of which task should be run at which voltage. We have defined the following five properties. For each property, we state the typical reasonings and examples of why that property should be included in the decision strategy of a high quality on-line algorithm.

Latency. If the latency of multiple tasks of a process are close to their maximum allowed latency, this process should be selected. Additionally, a higher voltage should be run to ensure each of the tasks meet their latency requirements. However, if the latency for all tasks/processes are at lower levels, then the task of the current process should be executed to eliminate a context switching penalty.

Relative Burstiness. The recent burstiness of a process, or rapid arrival of tasks for a process, can play an important role in voltage and task selection. If a task has shown recent burstiness, we should consider the execution of the task/process due to the likelihood that this task will continue to be bursty, therefore consuming more buffer space and extending the latency of each of the tasks if they are not run.

Number of Tasks. The number of tasks which a process has waiting also plays a key in the task and voltage selection process. If the current selected process has more tasks than the other processes, the tasks of the current process should continue to be selected in order to eliminate context switching penalties.

Synchronization. When the synchronization for any task/process is nearing the maximum allowed level for QoS this task should be selected. If the synchronization of the currently selected task/process is high, the algorithm should continue to run this process as long as possible to avoid context switching. For voltage selection, if the selected task is close to the maximum allowed synchronization level, a higher voltage should be applied.

Buffer Occupancy. Buffer occupancy is an indiction of the current demand of the processes as a whole. This property looks at the percentage of the entire buffer in which each process occupies. If the buffer is near capacity, the processes with higher buffer occupancy should be selected.

VI. Experimental Results

In this section, we present the experimental results obtained using comprehensive simulation study. We first describe the used examples (multimedia applications). After that, we present our experimental setup and collected data. Finally, we conduct the analysis of the experimental results.

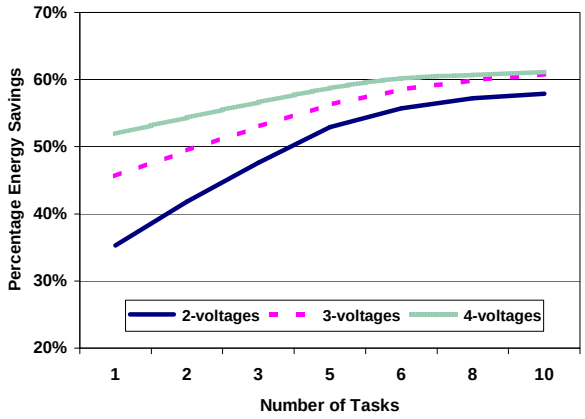

Fig. 6. Energy savings by off-line algorithm using multiple supply voltage assuming $3.3V$ nominal voltage with the same amount of memory.

A. Experimental Study: Goals and Procedure

In order to evaluate the on-line heuristic, we adapt the following procedure. We use four CPU units for the latency constraints. For synchronization, we use eight CPU units. The goals of the our experimentation and results analysis was to answer the following questions: Are multiple voltages useful? How many voltages are needed? What is the relative quality of the on-line algorithm with comparison to the optimal off-line scheme? How much benefit one can obtain for on-line algorithms when the goal is to minimize design costs (buffer storage) under energy consumption constraints?

We used six streaming applications [20] to evaluate the effectiveness of the approach: IJG JPEG encoder and decoder, MSG MPEG encoder and decoder, CCITT G.721 encoder, and PGP encryption and description module.

In order to analyze the effectiveness of our approach, we started with preliminary testing to determine the appropriate number of voltage levels. We considered three cases, 2-voltages at 3.3V and 1.8V, 3-voltages at 3.3.V, 1.8V, and 1.0V, and 4-voltages at 3.3V, 2.4V, 1.8V, and 1.0V. We used the off-line algorithm to determine the energy savings of the three cases on 1, 2, 3, 5, 6, 8, and 10 processes. All energy consumption values were normalized to the single voltage case at 3.3V, and we used the memory requirement for the single voltage case as the basis for the other cases. We present the results in Fig. 6. The figure shows the percentage of energy savings versus the number of processes.

The first three questions are addressed using data from the experiments that are displayed in Fig. 7. The figure shows that for different number of processes the normalized energy requirements when the optimal off-line and on-line algorithms are applied under the same memory requirement. All the values are normalized to the single voltage (3.3V) case found using the off-line algorithm. Since the off-line algorithm is optimal, we use the off-line values as the lower bound. The figure indicates the results after applying the optimal off-line algorithm and the on-line algorithm for both the 2-voltage and 3-voltage cases. The percentage by which the off-line and the on-line algorithm differ in savings compared to the number of processes is presented in Fig. 8.

Fig. 9 presents the results for the dual problem evaluated using Fig. 7. Here we evaluate how much the cost of the system, measured in terms of buffer space, can be reduced under the conditions that energy consumption is fixed. All results are normalized against the base case where storage requirements are first calculated for the set of tasks assuming that a single voltage is used. For the case when we use 2-voltages we compare to 2.5V, and in the case for 3-voltages we use 1.8V. Again, we present the normalized results for both the 2-voltage case, and the 3-voltage case in Fig. 9. Additionally, we present the percentage difference between the optimal off-line memory requirement and the on-line algorithm for both the 2-voltage case and the 3-voltage case in Fig. 10.

Lastly, we consider the relationship between the drop rate of the on-line and off-line algorithm relative to their performance. These results are presented in Table V. For both the 2 and 3-voltage level cases, we varied the drop rate from 0.1 to 0.5 and considered and average and median percentage difference between the drop rates of the off-line and on-line approaches.

B. Analysis of Experimental Results

The first important question that we analyze is what is the optimal number of voltage levels required to obtain essentially all potential benefits from the use of multiple voltages. In Fig. 6, we analyze the potential benefit for the use of 2,3 and 4-voltage levels with our off-line algorithm. Our results show a energy savings of at least 35% when 2-voltages are used, and at least 45% improvement in the 3-voltage case. While the energy saving increases with the number of voltage levels used, the benefit of using 4-voltages over 3-voltages is relatively small. Therefore, we see diminishing returns when using more than 3-voltage levels.

The comparison between the optimum off-line and the heuristic on-line algorithms with respect to storage requirements and energy savings indicates several important conclusions. Evaluation of the on-line and off-line memory consumption is shown in Fig. 8. On average the on-line algorithm indicates an overhead of 25%. However, it also saves energy over the single voltage case with 36.5% savings for 2-voltages, and 44.4% savings for 3-voltages. Therefore, we see that significant savings in energy can be achieved by applying multiple voltages.

We see that although the on-line algorithm is not capable of completely matching the performance of the off-line algorithm in Fig. 8, it nevertheless brings very significant improvements over the single voltage case, and that this difference has only limited additional potential for further energy reduction.

From Fig. 10, we see that the on-line algorithm is not able to completely match the performance of the optimal off-line algorithm, the reduction for storage requirements are significantly larger than the energy savings. This is a consequence of the fact that energy consumption is dictated by the overall average effectiveness of on-line and off-line algorithms, while the storage requirements are primarily a function of how well these algorithms can use high voltages to reduce storage requirements during bursty periods of processes.

The standard procedure for the analysis of on-line algorithms is competitive analysis. Competitive analysis [4],

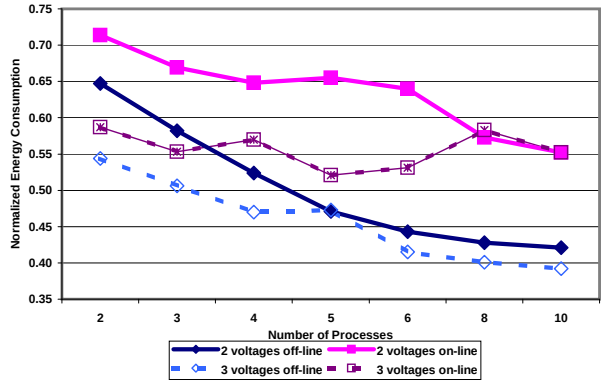

Fig. 7. Normalized energy consumption for 2-voltage and 3 voltages. Normalization is performed with respect to the single voltage case.

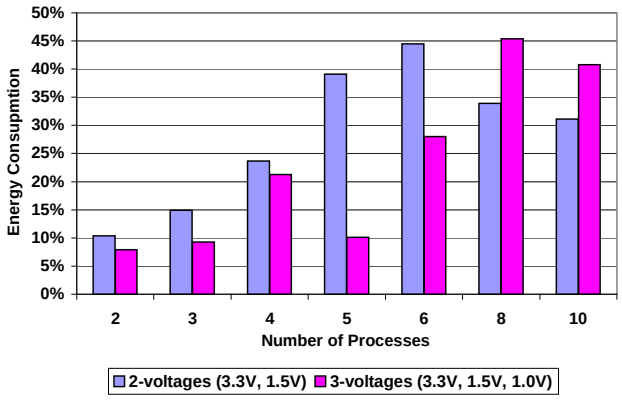

Fig. 8. Increase of energy consumption (in %) by the on-line heuristic over the off-line.

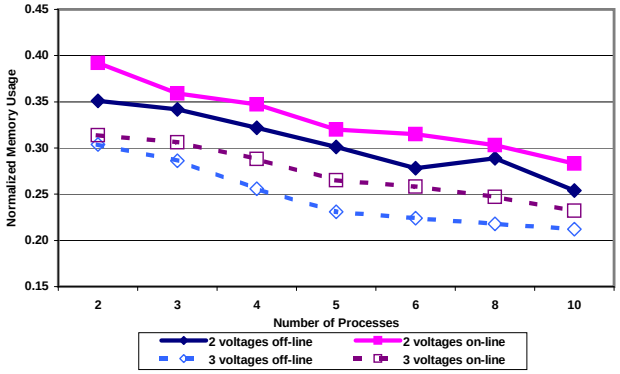

Fig. 9. Normalized memory consumption for 2-voltage and 3 voltages. Normalization is done with respect to the single voltage case.

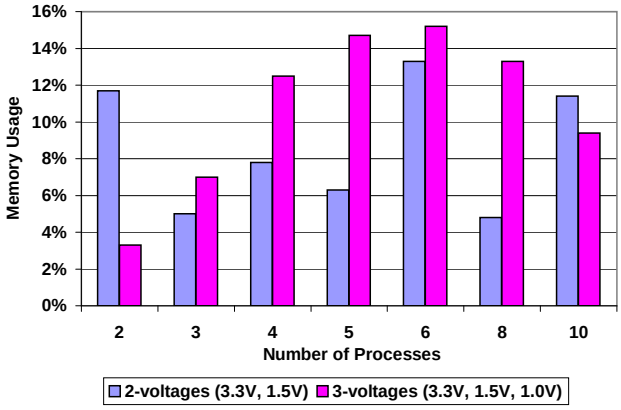

Fig. 10. Increase of memory consumption (in %) by the on-line heuristic over the off-line.

[15] evaluates the on-line algorithm by comparing its performance to the best results that could have been achieved if all inputs had been known in advance. All competitive analysis techniques assume idealized probabilistic distribution for inputs. Such distribution does not exist in closed form for streaming media and therefore classical competitive analysis is not feasible. However, since we have an optimal algorithm for the power minimization using multiple voltages, statistical competitive analysis is a proper alternative. Fig. 7 shows that at average the on-line heuristic is $\frac{1}{4}$ less effective than the off-line algorithm.

The main reason for this situation is that we insist on having a very low drop rate and therefore we were conservative in applying low voltage. If we increase the drop rate to twice the level in which we used, the performance characteristic of the on-line algorithms are only 15% lower than the optimal off-line algorithm. Further increase in the drop rate, shown in Table V, further reduces the discrepancy between the on-line and off-line algorithms. The average and median percentage difference between the off-line and the on-line algorithms are shown for both the 2-voltage and 3-voltage cases. It is worth noting that context switching of the on-line heuristics was by a factor of 8.9 times lower than the corresponding time of the off-line algorithm. Finally, note that the on-line algorithm takes less than 0.1% of the computational effort with respect to actual data processing. The number indicates the percentage by which the on-line heuristic is inferior to the off-line optimal algorithm.

VII. Conclusion

We have developed an optimal polynomial-time algorithm for power minimization of popular streaming media applications, such as audio, video, and sensor network data under QoS requirements and hardware constraints using multiple voltages. Furthermore, we have developed an on-line adaptive policy for power minimization in the same scenario. The on-line approach leverages the insights from the off-line optimal algorithm. By exploiting both long and short term statistical information and by bookkeeping the information about buffer occupancy, we created an on-line algorithm which performs well in a variety of workload scenarios. Both the off-line and on-line algorithms are flexible and adaptable, in the sense that they can address a variety of dual-primal QoS problem formulations, as well as a variety of QoS dimensions, such as latency and synchronization, as well as workload characteristics.

Acknowledgements

This material is based upon work partially supported by the National Science Foundation under Grant No. ANI-0085773. other proper agency and grant). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

TABLE V

RELATION BETWEEN DROP RATE AND RELATIVE PERFORMANCE OF THE ON-LINE AND OFF-LINE ALGORITHMS.

	2-voltages (3.3V, 1.8V)		3-voltages (3.3V, 1.8V, 1.0V)	
Drop Rate	average	median	average	median
0.1	29.8 %	34.5%	31.8%	36.1%
0.2	14.7%	15.6%	14.9%	17.6%
0.25	10.8%	11.2%	11.5%%	12.3%
0.3	9.1%	9.9%	10.1%	11.5%
0.4	4.8%	5.8%	6.2%	7.0%
0.5	2.8%	2.6%	2.8%	2.5%

REFERENCES

[1] C. Aurrecoechea, Andrew T. Campbell, and Linda Hauw. A survey of qos architectures. In *Multimedia Systems*, volume 6, pages 138–151, 1998.

[2] L. Benini, A. Bogliolo, G.A. Paleologo, and G. De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on CAD*, 18(6):813–833, June 1999.

[3] D. Boerstler and et al. Dynamically scalable low voltage clock generation system. US patent pending.

[4] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[5] T.D. Burd, T.A. Pering, A.J. Stratakos, and R.W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal on Solid-State Circuits*, 35(11):1571–1579, November 2000.

[6] W. J. Canover. *Practical nonparametric statistics*. 3rd ed. New York : Wiley, 1999.

[7] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen. Optimizing power using transformations. *IEEE Transactions on CAD*, 14(1):12–31, January 1995.

[8] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, April 1992.

[9] E. Chang and H. Garcia-Molina. Medic: Memory and disk cache for media servers. In *Multimedia Computing and Systems*, pages 493–499, 1999.

[10] J. Chang and M. Pedram. Energy minimization using multiple supply voltages. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 157–162, 1996.

[11] E-Y. Chung, L. Benini, and G. De Micheli. Dynamic power management using adaptive learning tree. In *International Conference on Computer-Aided Design*, pages 274–279, 1999.

[12] R.L. Cruz. Quality of service guarantees in virtual circuit switched networks. *Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.

[13] R.L. Cruz, H. Sariowan, and G.C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Fourth International Conference on Computer Communications and Networks (ICCCN'95)*, pages 512–520, 1995.

[14] B. Efron and R. Tibshirani. *An introduction to the bootstrap*. Chapman & Hall, 1993.

[15] A. Fiat and G.J. Woedinger. *On-line algorithms: the state of the art*. Germany, 1998.

[16] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *ACM Mobile Computing and Networking*, pages 13–25, 1995.

[17] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M.B. Srivastava. Power optimization of variable voltage core-based systems. *IEEE Transaction on CAD*, 18(12):1702–1714, December 1999.

[18] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED*, pages 197–202, 1998.

[19] N.K. Jha. Low power system scheduling and synthesis. pages 259–263, 2001.

[20] C. Lee and et al. Mediabench: a tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

[21] J. Luo and N.K. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *Asia and South Pacific Design Automation Conference*, pages 719–726, 2002.

[22] A. Manzak and C. Chakrabarti. A low power scheduling scheme with resources operating at multiple voltages. In *IEEE International Symposium. on Circuits and Systems*, volume 1, pages 354–357, 1999.

[23] B. Mochocki, X. Hu, and G. Quan. A realistic variable voltage scheduling model for real-time applications. In *IEEE International Conference on Computer Aided Design*, pages 726– 731, 2002.

[24] K. Nose, M. Hirabayashi, H. Kawaguchi, L. Seongsoo, and et al. V/sub TH/-hopping scheme to reduce subthreshold leakage for low-power processors. In *IEEE Custom Integrated Circuits*, pages 93–96, 2001.

[25] G. Qu, M. Mesarina, and M. Potkonjak. System synthesis of synchronous multimedia applications. In *International Symposium on System Synthesis (ISSS)*, pages 128–133, 1999.

[26] J. M. Rabaey and M. Pedram. *Low power design methodologies*. Kluwer Academic Publishers, 1996.

[27] S. Raje and M. Sarrafzadeh. Scheduling with multiple voltages. *Integration, The VLSI Journal*, 23(1):37–59, 1997.

[28] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *IEEE Real-Time Systems Symposium*, pages 298–307, 1997.

[29] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. Practical solutions for qos-based resource allocation problems. In *IEEE Real-Time Systems Symposium*, pages 296–306, 1998.

[30] V. Sandararajan and K.K. Parhi. Synthesis of low power cmos vlsi circuits using dual supply voltages. In *Design Automation Conference*, pages 72–75, 1999.

[31] A. Sinha and A.P. Chandrakasan. Energy efficient real-time scheduling microprocessors. *IEEE/ACM International Conference on Computer Aided Design*, pages 458–463, 2001.

[32] M. A. Stephens. *Goodness-of-fit techniques*. New York : M. Dekker, 1986.

[33] R. Thisted. *Elements of statistical computing*. Chapman and Hall, 1988.

[34] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *USENIX Operating Systems Design and Implementation*, pages 13–23, 1994.

[35] J. L. Wong, G. Qu, and M. Potkonjak. Power minimization under QoS constraints. In *International Packetvideo Workshop*, 2002.

[36] J. L. Wong, G. Qu, and M. Potkonjak. An on-line approach for power minimization in QoS sensitive systems. In *Asia South Pacific Design Automation Conference*, 2003.

[37] J. L. Wong, G. Qu, and M. Potkonjak. Power minimization in QoS sensitive systems. Report, University of California, Los Angeles, 2003.

[38] T. Yamashita, N. Yoshida, M. Sakamoto, T. Matsumoto, and et al. A 450 MHz 64 b RISC processor using multiple threshold voltage CMOS. In *IEEE International Solid-State Circuits Conference*, pages 414–415, 2000.