Dynamic Power Management for non-stationary service requests *

Eui-Young Chung[#] †
eychung@stanford.edu

Luca Benini [‡]
lbenini@deis.unibo.it

Alessandro Bogliolo [‡]
abogliolo@deis.unibo.it

Giovanni De Micheli [#]
nanni@galileo.stanford.edu

[#]Stanford University
Computer System Laboratory
Stanford, CA, 94305, USA

[‡] Università di Bologna
Dip. Informatica, Elettronica, Sistemistica
Bologna, ITALY 30165

Abstract

Dynamic Power Management is a design methodology aiming at reducing power consumption of electronic systems, by performing selective shutdown of the idle system resources. The effectiveness of a power management scheme depends critically on an accurate modeling of the environment, and on the computation of the control policy. This paper presents two methods for characterizing non-stationary service requests by means of a prediction scheme based on sliding windows. Moreover, it describes how control policies for non-stationary models can be derived.

1. Introduction

Design methodologies for energy-efficient system-level design are receiving increasingly larger attention [2, 6, 7, 10], because of the widespread use of portable electronic appliances (e.g., cellular phones, laptop computers, etc.) and of the concerns about the environmental impact of electronic systems.

Dynamic power management (DPM) [1] is a flexible and general design methodology aiming at controlling performance and power levels of digital circuits and systems, by exploiting the idleness of their components. A system is provided with a *power manager* that monitors the overall system and component states and controls the state transitions. The control procedure is called power management *policy*.

Srivastava et al. [5] proposed heuristic policies to shut down a system during idle periods. The basic idea in [5] is to predict the length of idle periods and shut down the system when the predicted idle period is long enough to amortize the cost (in latency and power) of shutting down and later re-activating the system. A shortcoming of the predictive shutdown approaches proposed by Srivastava is that they are based on off-line analysis of usage traces, hence they are not suitable to non-stationary request streams whose statistical properties are not known a priori. This shortcoming is addressed by Golding et al. [8, 9] and Hwang and Wu [4] proposed on-line methods that dynamically adapt the shutdown policy based on the distribution of past user requests.

All predictive shutdown techniques share a few limitations. First, they do not deal with resources with multiple states of operation (instead of just active and sleep). Second they cannot accurately trade off performance losses (caused by transition delays between states of operation) with power savings. Third, they do not deal with general system models where incoming requests can be enqueued while waiting for service.

These limitations are addressed in [3] where general systems (with multiple states and queueing) and user request are modeled as Markov chains. The Markov model enables a rigorous formulation of the search for optimal power management policies as a constrained minimization problem whose exact solution can be found in polynomial time. Unfortunately a basic assumption in [3] is that the Markov model is stationary. This assumption clearly does not hold if the system experiences non-stationary load conditions.

The contribution of this paper is twofold. We propose prediction schemes for the service requester (e.g., user) that captures the non-stationary property of its behavior. We present two schemes based on sliding windows to capture the time-varying parameters of the stochastic processing modeling the requester. Next we show how policies for dynamic power management under non-stationary service request models can be determined by interpolating optimal policies computed under an assumption of stationariety.

2. System Modeling

In this section, we briefly review the system model introduced in [3]. The overall system model for DPM is shown

*This work was supported in part by MARCO and ARPA.
†Eui-Young Chung was supported by Samsung Electronics. Co. LTD.

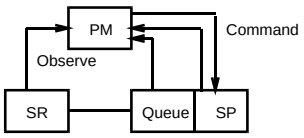


Figure 1. Overall System Model for DPM

in Figure 1. An electronic system is modeled as a unit providing a service, called *service provider* (SP) while receiving requests from another unit, called *service requester* (SR). A *queue* buffers incoming unserviced requests. The service provider can be in one of several states (e.g. *active*, *sleep*, *idle*, etc.). Each state is characterized by the ability/inability of providing a service and by a power consumption level. Transitions among states may have a performance penalty (e.g., latency in reactivating a unit) and a power penalty (e.g., power loss in spinning up a hard disk).

A *power manager* (PM) is a control unit that controls the transitions among states. The power consumption of the PM is negligible with respetct to the overall power dissipation. At given points in time, the power manager evaluates the overall state of the system (provider, queue and requester) and decides to issue a command to stimulate a state transition. A *control policy* is a sequence of decisions. For our purposes, a policy can be thought of as a table, that associates a probability of issuing a command with any system state.

We model the system components as Markov chains [3]. In particular, we use a controlled Marcov chain model for the system provider, so that the transition probabilities can be made dependent on the command issued by the power manager. We use a discrete time setting, *i.e.* we assume that time is divided into time slices.

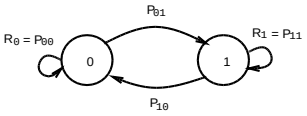


Figure 2. An example of a Markov Chain

whereas SP behaviors can be modeled as a stationary process, because the SP response to stimuli does not change over time, the workload bounce can be highly non-stationary, and that it is appropriate to model SR as a non-stationary process. In this work, we focus on non-stationary service requesters. A generic requester can have s states. For the sake of simplicity, we will discuss the case of $s = 2$, as shown in Figure 2. When in state 0, no request is issued. When in state 1, one request per time slice is issued. The corresponding transition matrix is denoted by P. We call the diagonal elements of P *user request probabilities* and we denote them by $R_i, i = 0, 1$. The probabilities $R_i = Prob(s(t+1) = i | s(t) = i), i = 0, 1$ (and the entire transition matrix P) are fixed for the stationary model [3]. For capturing the non-stationariety of SR, we assume that R_i can change over time and are initially unknown.

3. User Request Prediction

3.1. Single Window Approach

A sliding window is adopted to keep the recent user request history and this information is used to predict future user requests. A sliding window denoted as W, consists of WS slots and each slot, $W(i), i = 0, 1, \cdots WS-1$, stores one previous user request, i.e. $s \in 0, 1, \cdots, S-1$. The basic window operation is to shift one slot constantly every time slice. An example for a window operation for two-state

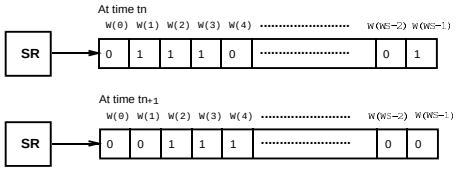


Figure 3. Single window operation for two-state user requests

user requests is shown in Figure 3. As shown in Figure 3, at each time point, $W(i+1) \leftarrow W(i), i = 0, 1, \cdots, WS-1$ and $W(0)$ stores a new user request from SR. The user request prediction for the next time point is done as follows. Let $l = \sum_{k=1}^{WS-1}(W(k) = i)$. Then,

$$P(i,j) = \begin{cases} 1/l \sum_{k=1}^{WS-1}[(\text{W(k) = i}) \wedge (\text{W(k-1) = j})] & \text{if } l \neq 0 \\ 0 & \text{if } l = 0, \text{and } i = j \\ 1 / (\text{S-1}) & \text{otherwise} \end{cases} \quad (1)$$

where, "$=$" is the equivalence operation with a Boolean output, (*i.e.* it yields "1" when the two arguments are same, otherwise returns "0"), and where "$\wedge$" is the "*conjunction*" operation. Thus, a policy applied at a given time point should be optimized for the P predicted at the previous time point. This can be achieved through policy table look-up method described in Section 4.

3.2. Multi Window Approach

The basic structure for multi-window approach is shown in Figure 4. There are as many windows as the number of states S of SR and their sizes are same(WS). At a time point, the user request of the previous time point is stored in the Previous Request Buffer (PRB) and this buffer controls the window selector to choose a window in which the

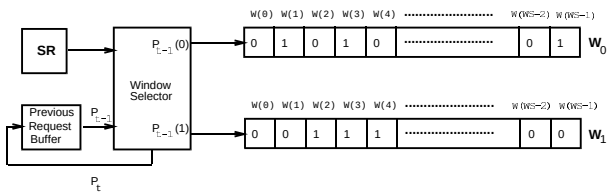


Figure 4. Multi window operation for two-state user requests

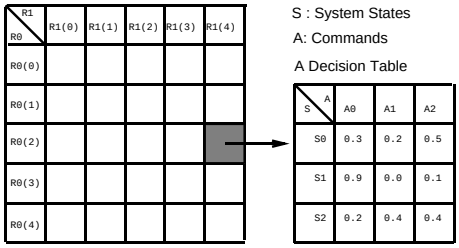


Figure 5. An example of a 2D policy table

current user request is stored. For convenience, the window selected when PRB stores state s of a user request is represented as W_s. Each window slice stores the state of a user request. Note that only the selected window performs the shift operation, while the other windows stay constant. Thus, each window W_s stores WS previous user requests and plays a role to predict the transition probabilities from state i to any other states. Each row of P is mapped to the window corresponding to the state which is source of the transition and $P(i,j)$ can be easily calculated as follows.

$$P(i,j) = \frac{\sum_{k=0}^{WS-1}(W_i(k)=j)}{WS} \quad \text{for all i, j} \qquad (2)$$

One important factor in a window approach is the window size, WS. It highly contributes to the accuracy of prediction because WS determines the precision of the predicted user request probability. If WS is too small, a small change in user requests causes a large effect to user request prediction. Conversely, if WS is too large, a large change in user requests gives only a small amount of effect to the prediction. Extensive experimental work for window size selection is described in later section.

4. Dynamic Power Management

4.1. Policy Table Construction

A look-up table stores the data sampled from a *n*-dimensional function which has *n* input variables. Sampling points of each variable are mapped to the table indices of the corresponding dimension of the table. For two-state service requester, input variables are R_O and R_1. By denoting the number of sampling points of each variable as k_i, i=0,1, the sampling points for R_0 and R_1 can be be generally denoted as $R_O(i)(i = 0, 1, \cdots, k_0 - 1)$, $R_1(j)(j = 0, 1, \cdots, k_1 - 1)$, respectively. An example of a two-dimensional policy table constructed for $k_0 = k_1 = 5$ is shown in Figure 5. Each cell of a policy table is also a two-dimensional table, which we call a decision table. A decision table is a matrix with as many rows as the total system states and as many columns as the command issued by the PM to SP. Each cell of a policy table can be indexed as a pair $(R_0(i), R_1(j))$. For each pair $(R_0(i), R_1(j))$, a policy optimization is performed to get the decision table and the obtained decision table is stored to a cell of the policy table with the corresponding index. The overall table is constructed once for all and its size is $k_0 \times k_1$ times the size of the table used in [3]. Apart from the larger storage overhead, there is no performance penalty in accessing this table.

4.2. Decision Using Interpolation

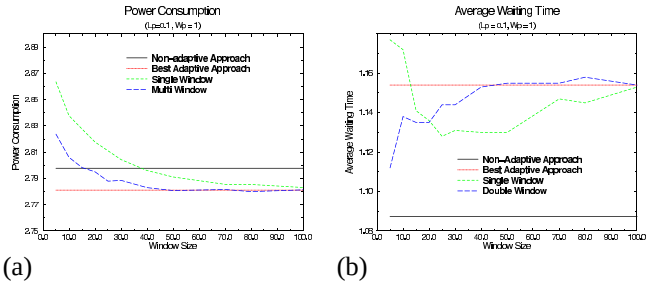
The predicted user request can be used to select cells in the policy table containing optimal policy at a given time. If the predicted user request is one of the sampling points used in policy optimization, we can choose a decision from the cell in the corresponding table index according to the system state denoted as CS. But there may be a gap between the predicted user request and sampling points because predicted user request is a value calculated in a continuous interval (from 0 to 1), while the sampling points are discretely sampled. We use a two-dimensional linear interpolation/extrapolation technique to compute the decision tables for (R_0, R_1) value pairs that do not correspond to sampled values in the policy table. The pseudo-code of the interpolation/extrapolation procedure is shown in Fig. 6. Extrapolation is used if the values of any of the R_i is either larger than the largest sampled value or smaller than the smallest sampled value. In all other cases, the interpolated value is computed as three successive one-dimensional linear interpolations on the table entries corresponding to (R_0, R_1) pairs surrounding the predicted values of R_0 and R_1.

5. Experimental Results

We applied the proposed prediction schemes to a Hard Disk Drive with highly-non stationary workload. The HDD (i.e., the SP) consumes 3W in *active* state and 0W in *sleep* state. The transition time from active to sleep (and vice versa) takes in average 10 time slices. Requests can be stored in a queue with length 2, and require in average 1.25

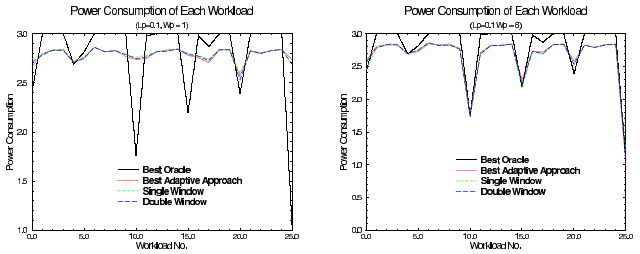
```
2DInterpolation (CS, cell, PR_0, PR_1, k_0, k_1)
for (i = 0; i < 2; i++) {
  if (PR_i ≤ R_i(0)) { /* extrapolation */
    id_i^0 = 0;
    id_i^1 = 1;
  } else if (PR_i ≥ R_i(k_i − 1)) { /* extrapolation */
    id_i^0 = k_i - 2;
    id_i^1 = k_i - 1;
  } else { /* interpolation */
    id_i^0 = j s.t. R_i(j) ≤ PR_i ≤ R_i(j+1);
    id_i^1 = j+1;
  }
}
for (i = 0; i < 2; i++) {
  for (j = 0; j < 2; j++) {
    Select a decision d_{2i+j} from cell(id_0^i, id_1^j)
    for State CS;
  }
}
foreach (command) {
  d_5 = OneDimInterp(d_0, d_1);
  d_6 = OneDimInterp(d_2, d_4);
  d_7 = OneDimInterp(d_5, d_6);
}
return(d_7);
```

Figure 6. 2-dimensional Interpolation



(a) (b)

Figure 7. (a) Power Comparison (b) Average Waiting Time Comparison



(a) (b)

Figure 8. (a) Power Comparison of Each Workload ($W_p = 1$) (b) Power Comparison of Each Workload ($W_p = 8$)

time slices to be served. The Markov model of the system (SP, Queue and SR) has 14 states. The PM can issue 2 commands: GO_ACTIVE and GO_SLEEP. The decision table from the policy optimization is 14×2 matrix, where the row is the total number of states and the column is the number of commands [3]. We tested our adaptive policy on a worst-case, highly non-stationary workload, constructed by concatenating 26 workloads with different R_0, R_1, and different duration (between 30,000 and 50,000 time slices). With a perfect workload estimation scheme, each workload would be immediately identified, the optimal policy computed and used until the change to a new workload. We call this policy **Best-Adaptive**. We can also define another ideal policy, called the **Best-Oracle**, that assumes perfect knowledge not only of workdload parameters, but also of the duration of every single idle period in the request stream. In this case, we can analyze the stream to find idle periods which are longer than the sum of transition time from active state to sleep state and vice versa. These idle periods can be used for system shutdown without performance degradation. Obviously, both the best-adaptive and best-oracle policies are theoretical limits, which cannot be achieved in practice because they require the availability of pefect predictors of future events (they can be tested in simulation by examining in advance the entire input trace that drives the simulator). They will be used for comparison with our adaptive policies. We extended the optimizer and simulator introduced in [3] to generate the policy table and simulate user workload with the proposed prediction schemes. We computed minimum power performance-constrained policies. Performance constraints are expressed by limiting the average time spent by a request in the queue (W_p) and the probability of an incoming request to find the queue full (we call this event *request loss* L_p). Notice that the constraints are not equivalent: L_p is usually tighter for workloads with high number of requests (because of the limited queue lengh), while W_p dominates for light workloads. For the first set of experiments, L_p was chosen to be 10% more than the loss experienced without power management, and W_p was set to increase expected waiting time by 1 time slice. Optimization and simulation were performed on SUN Ultra2 SPARC workstation (200MHz, 520MB main memory) to find out the optimal window size (WS). The computation time for policy table construction (10×10 table) was approximatively 5 minutes. We compared the power and performance of our adaptive approaches with the best-adaptive policy. We also compare with the non-adaptive approach [3] which considers the overall workload as a stationary Markov chain with constant R_0 an R_1. Figure 7 (a) and (b) report respectively power and average waiting time as a function of sliding window width. The non-adaptive approach produces higher power consumption than our adaptive approaches. Also, constraints are now well matched (even though, in this case, the non-adaptive policy is conservative). The graphs also show that both window approaches are very close to the best-adaptive policy when the window size is reasonably large. Notice that double window approach is more stable

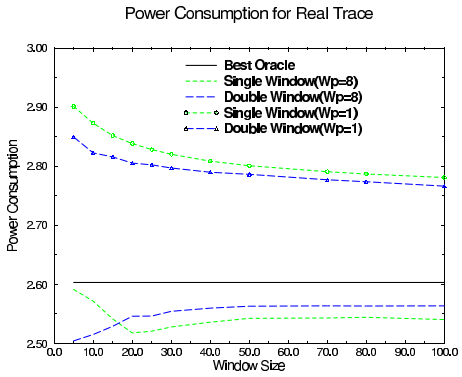


Figure 9. Power Consumption for Real Trace

and accurate than the single window approach for the same window size. The appropriate window size is over 40 for the double window approach and is over 90 for the single window approach. Not only the overall trade-off between the performance and power, but also the local trade-off is important because it represents how well the policy exploits idleness on a short time scale. Figure 8 reports on the x axis the index of the 26 different workloads of the non-stationary trace. Power obtained with best-adaptive, best-oracle and our adaptive policies (with window size 50) is reported on the y axis. When the performance constraint is tight ($L_c = 0.1$, $W_p = 1$) as shown in Figure 8 (a), the window approaches are still comparable to the best-adaptive approach, but even the best-adaptive policy does worse than the best-oracle for light workloads with tight performance constraints. When the waiting time constraint, W_p is relaxed without change of L_p ($L_p = 0.1$, W_p = 8), the best-adaptive approach and both window approaches are comparable to the best-oracle as shown in Figure 8 (b). Notice that relaxing the constraint does not imply that the waiting time increases proportionally: for heavy workloads the request loss constraint dominates and keeps the waiting time low. To confirm this intuitive fact, we observed that the waiting time increase on the entire trace when $W_p = 8$ is only 10% with respect to the waiting time when $W_p = 1$. It is also interesting to observe that for some workloads the adaptive policies save even more power than the best-oracle. This is possible because some performance is traded off for power, while this is not allowed in the best-oracle policy, where performance is constrained to be the same as without power management. Finally, we applied the window approaches to a real trace of time-stamped disk accesses [3]. We performed two experiments with different constraints, namely $L_p = 0.1$, $W_p = 1$ and with $L_p = 0.1$, $W_p = 8$ and compared the power consumption with that of the best-oracle policy. Results are shown in Figure 9. Since the trace is for a relatively light workload, the power consumption of the adaptive policies with tight W_p constraints is high with respect to the best-oracle policy. On the contrary, by relaxing the W_p constraint ($W_p = 8$), we obtain policies that are even more aggressive than the best-oracle with a measured waiting time increase of just 6% compared to the average waiting time when $W_p = 1$. The adaptive policies compare favourably with the non-adaptive approach for both tight and loose W_p: average power is 2.81W for the non-adaptive approach (for both W_p values) while it is 2.78W (with tight constraint) and 2.57W (with loose constraint) for the double-window adaptive policy ($WS = 100$).

6. Conclusions

In this paper, we described adaptive power management policies for non-stationary workloads. Our adaptive approach is based on sliding windows and interpolation to find an optimal policy from a pre-characterized optimal policy table. The proposed approach deals effectively with highly non-stationary workloads: good overall and instantaneous accuracy is achieved. Moreover, our adaptive method offers the possibility of trading off power for performance in a controlled fashion. Simulation results show that our method is comparable to ideal predictive policies which cannot be implemented in practice and outperforms non-adaptive policies.

References

[1] L. Benini and G. De Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer, 1997.

[2] A. Chandrakasan and R. Brodersen. *Low-Power Digital CMOS Design*. Kluwer, 1995.

[3] G. Paleologo, L. Benini, A. Bogliolo and G. D. Micheli. Policy optimization for dynamic power management. *DAC - Proceedings of the Design Automation Conference*, pp.182-187, 1998.

[4] C.-H. Hwang and A. Wu. A predictive system shutdown method for energy saving of event-driven computation. *Proceedings of the Int.l Conference on Computer Aided Design*, pp.28-32, 1997.

[5] M. Srivastava, A. Chandrakasan and R. Brodersen. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. *IEEE Transactions on VLSI Systems*, 4(1), pp.42-55, March 1996.

[6] J. Monteiro and S. Devadas. *Computer-Aided Techniques for Low-Power Sequential Logic Circuits*. Kluwer, 1997.

[7] W. Nebel and J. Mermet (Eds.). *Low-Power Design in Deep Submicron Electronics*. Kluwer, 1997.

[8] R. Golding, P. Bosh and J. Wilkes. Idleness is not sloth. *Proceedings of Winter USENIX Technical Conference*, pp.201-212, 1995.

[9] R. Golding, P. Bosh and J. Wilkes. Idleness is not sloth. *HP Laboratories Technical Report HPL-96-140*, 1996.

[10] J. M. Rabaey and M. Pedram (editors). *Low-Power Design Methodologies*. Kluwer, 1996.