Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 65478, 6 pages
doi:10.1155/2007/65478

Research Article

Hardware Architecture of Reinforcement Learning Scheme for Dynamic Power Management in Embedded Systems

Viswanathan Lakshmi Prabha[1] **and Elwin Chandra Monie**[2]

[1] *Department of Electronics and Communication Engineering, Government College of Technology, Coimbatore 641-013, Tamil Nadu, India*
[2] *Thanthai Periyar Government Institute of Technology TPGIT, Vellore 632002, Tamil Nadu, India*

Received 6 July 2006; Revised 7 November 2006; Accepted 28 May 2007

Recommended by Rajesh K. Gupta

Dynamic power management (DPM) is a technique to reduce power consumption of electronic systems by selectively shutting down idle components. In this paper, a novel and nontrivial enhancement of conventional reinforcement learning (RL) is adopted to choose the optimal policy out of the existing DPM policies. A hardware architecture evolved from the VHDL model of Temporal Difference RL algorithm is proposed in this paper, which can suggest the winner policy to be adopted for any given workload to achieve power savings. The effectiveness of this approach is also demonstrated by an event-driven simulator, which is designed using JAVA for power-manageable embedded devices. The results show that RL applied to DPM can lead up to 28% power savings.

1. INTRODUCTION

Dynamic power management (DPM) techniques aid energy efficient utilization of systems by selectively placing system components into low-power states when they are idle. A DPM system model consists of Service provider, Service queue, Service requestor and Power Manager. Power manager (PM) implements a control procedure (or policy) based on observations of the workload. It can be modeled as a power state machine, each state being characterized by the level of power consumption and performance. In addition, state transitions have power and delay cost. When a component is placed into low-power state, it becomes unavailable till it is switched on to the active state. The break-even time, Tbe, is the minimum time a component should spend in the low-power state to compensate the transition cost [1]. Hence it is critical to determine the most appropriate policy that the Power Manager will implement to achieve optimal power.

Appropriate policy of the Power Manager will implement to achieve optimal power.

2. SYSTEM LEVEL-POWER MANAGEMENT POLICIES

Power management policies can be classified into four categories based on the methods to predict the movement to low power states. The categories are greedy, timeout, predictive, probabilistic, and stochastic. The greedy based [2] power management will simply shutdown the device whenever it becomes idle. It is simple; however, the performance is not very good. A timeout policy [2] has a timeout value τ. Timeout policies assume that after a device is idle for τ, it will remain idle for at least Tbe. An obvious drawback is the energy wasted during this timeout period. Timeout-based policies include fixed timeout, such as setting τ to three minutes. Alternatively, timeout values can be adjusted at runtime. History, based or predictive policies predict the length of an idle period. If an idle period is predicted to be longer than the break-even time, the device sleeps right after it is idle. Requests make a device change between busy and idle. Probabilistic policies [1] predict idle time online and dynamically change the threshold that decides the state movement. Stochastic policies model [2] the arrival of requests and device power state changes as stochastic processes, such as Markov processes. Minimizing power consumption is a stochastic optimization problem [3–7]. DPM based on idle time clustering [8] using an adaptive tree method helps in moving the system to one of the multiple sleep states decided by the density of the clusters.

3. REINFORCEMENT LEARNING- BASED DPM

3.1. Motivation

From the discussion of all the previous works carried out, it is evident that success rate of each policy is dependent on the workload.

For example, when the requests come in at long time intervals, the greedy policy can give the best power optimization. When the requests come in continuously without inter-arrival time, worst policy (always on) can give best result. To effect further improvement in the battery life of portable devices, one new energy reduction scheme will be needed which has to predict the best and most suitable policy from the existing policies. This warrants for the use of intelligent controllers [9] that can learn themselves to predict a best policy that can balance the workload against power. This paper focuses on implementing an intelligent Power Manager that can change policy according to workload.

3.2. Reinforcement learning

A general model for Reinforcement Learning is defined based on the concept of autonomy. Learning techniques will be analyzed based on the probabilistic learning approach [10]. The Reinforcement Learning model considered learning agent (or simply the learner) and the environment. Reinforcement Learning relies on the assumption that the system dynamics has the *Markov property*, which can be defined as follows:

$$P_r\{s_{t+1} = s', r_{t+1} = r \mid s_0, a_0, r_0, \ldots, s_t, a_t, r_t\}, \tag{1}$$

where P_r is the probability of state [11] s and reward r that a system will reach at time $t+1$. The Markov property means that the next state and immediate reward depend only on the current state and action.

Given any state and action, s and a, the transition probability of each possible next state, s', is

$$P^a_{s,s'} = P_r\{s_{t+1} = s' \mid s_t = s, a_t = a\}. \tag{2}$$

Similarly, given any current state and action, s and a, together with any next state, s', the expected value of the next reward is

$$R^a_{s,s'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \quad \forall_s, s' \in s, a \in A(s). \tag{3}$$

These quantities, $P^a_{s,s'}$ and $R^a_{s,s'}$, completely specify the most important aspects of the dynamics of a finite MDP.

A policy, π, is a mapping from each state, $s \in S$, and action, $a \in A(s)$, to the probability $\pi(s,a)$ of taking action a when in state s,

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}, \tag{4}$$

where $E_\pi\{\cdot\}$ denotes the expected value given that the agent follows policy π, and t is any time step, γ is the discount factor. Similarly, we define the value of taking action a in state s under a policy π, denoted $Q^\pi(s,a)$, as the expected return starting from s, taking the action a, and thereafter following policy π,

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{R_t \mid s_t = s, a_t = a\} \\ &= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}, \end{aligned} \tag{5}$$

where, Q^π is the *action-value function for policy* π.

```
For every T sec
{
  IF (request)
    {
      - - - - -
    }
  ELSE (no request)
  State movement
  By winner
  policy;
  Compute cost or
  energy with all
  policies;
  Declare success
  or failure of
  winner policy
  based on energy;
  CALL AGENT;
}

AGENT
IF (success)
{
Reward winner
policy;
Update Reward
Table;
}
ELSE (failure)
 Punish policy;
 Policy with
highest reward
is winner policy;
```

Algorithm 1

3.3. Pseudocode

The general pseudocode for proceeding with the Reinforcement Learning DPM is as given Algorithm 1.

Temporal Difference Learning Algorithm (SARSA).

This learning scheme achieves better policy convergence than linear and nonlinear learning schemes. SARSA that stands for State- Action- Reward- State- Action [10] is an on-policy TD control method. On-policy methods evaluate or improve the current policy used for control. The first step is to learn an action-value function rather, that is, $Q(s,a)$ for the current behavior policy and for all states **s** (idle time) and actions **a** (choice of winner policy).

SARSA algorithm

Algorithm 2 repeatedly applies the learning rule to the set of values corresponding to the states in the environment. Starting with a state s, the algorithm chooses an action a using the maximum action state value and observes the next state s' besides the reward r. The value $Q(s,a)$ is updated using the SARSA algorithm, s is set to s' and the process repeats.

```
Initialize $Q(s, a)$;
Repeat (for each episode): Initialize $s$;
Choose $a$ from $s$ using policy derived
from $Q$;
Repeat (for each step of episode):
Take action $a$, observe $r, s'$; Choose $a'$
from $s'$ using policy derived from $Q$
$Q(s, a) \longleftarrow Q(s, a)$
    $+\alpha(r + \gamma^* Q(s', a) - Q(s, a))0$
$s \longleftarrow s', a \longleftarrow a'$.
Until $s$ is terminal.
$\alpha$, is the learning constants and $\gamma^*$ is the
discount factor.
```

Algorithm 2

4. SYSTEM MODEL

Agent

The aim of the proposed system is to select and adopt the best system-level power management policy. The agent is the learner. The agent in our system is responsible for learning through the desired RL scheme, updating the reward table, and issuing the action, that is, declaring the winner policy. This action is fed to the environment. Thus, the agent can be assumed to have three important parts: (1) reinforcement learner that implements the desired RL algorithm, (2) reward table (for SARSA Q-table) that gets updated by reinforcement learner and (3) action generator which selects the winner policy with the help of reward table. In short, the agent constitutes the brain of the system.

Environment

The environment constitutes the part that the agent cannot control, that is, the incoming traffic. It monitors the incoming user requests and decides whether the current policy, that is, the action generated by the agent is successful or not. If successful, it issues a command to increase the reward of the current policy; otherwise it issues a signal to punish the current policy. During the idle time, it puts the system in the lower modes according to the winning policy issued by the agent. These policies are then evaluated with the duration of the current idle period to decide whether they are successful or not. The two important parts of the environment can be termed as (1) the decision and implementation module, (2) the servicing module (Figure 3). The latter module services the requests till the requester queue remains un emptied. The decision and implementation module starts when the queue becomes empty and issues requisite command to implement the winner policy according to the action (i.e., the winner policy) selected by the agent. Thus, it puts the system to its optimal state according to the winner policy. The decision module makes use of the cost function for system-level policies to evaluate the energy for the current idle period.

The cost (energy) computation for different policies is indicated in Table 1.

Table 1: Cost computation for different policies.

Policy	Cost (energy)
Always on	$C_{\text{AP}} = (P_a * T_a)$, P_a-active power, T_a-active time
Greedy	$C_{\text{GP}} = P_a * T_a + P_i * T_i + e_i + e_L P_i$-idle power, e_i-startup energy, T_i-idle time
Time out	$C_{TP} = P_a * T_a + P_i * (\tau) + e_i + e_L$ L-latency, τ-threshold time
Stochastic	$C_{\text{DPM}} = P_a * T_a + P_i * T_r(n+1) + e_i +$ $L(T_i)$ $T_r(n+1)$-predicted idle time based on previous idle time

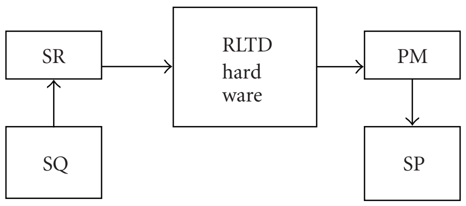


Figure 1: Structure of DPM with RLTD hardware block.

5. HARDWARE ARCHITECTURE

The basic model of a DPM has a Power Manager which issues commands to the service provider based on the input request and the queue, using a defined policy. The Power Manager could be activated by a hardware whose output is a winner policy. The winner policy would guide the Power Manager and switch the service provider to the sleep states optimally as shown in Figure 1.

The SARSA algorithm is converted into an equivalent hardware block by modeling the algorithm using a VHDL model.

The hardware architecture consisting of various blocks is as shown in Figure 2. It receives clock as one of the inputs and active signal as another input. When the active signal is high (low), it implies that the system is in Active state (idle state).

Idle time calculation unit

The input to this unit is the clk and the active input. The output of this unit is the idle time and active time value, which is fed to compute the cost or energy for different policies used.

Cost evaluation unit

with active and idle time duration as input, the cost (energy consumption) for all policies is calculated as per Table 1.

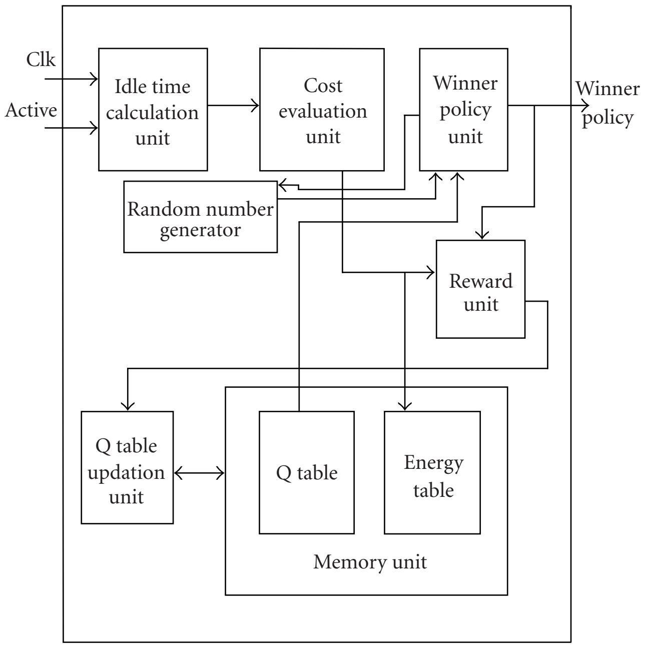


Figure 2: Architecture for SARSA algorithm.

Q-table updating unit

The input to this unit is the output of reward unit. For every idle time based on the reward or punishment a policy receives, the Q-table is updated using the Q-updating formula

$$\begin{aligned}\text{Update} :=&\ q\text{table}(0) + \text{alpha}\\ &* (\text{reward} + \text{gamma} * q\text{table}(1) - q\text{table}(0)).\end{aligned} \tag{6}$$

This Q-updating is carried out for all the polices.

Memory unit

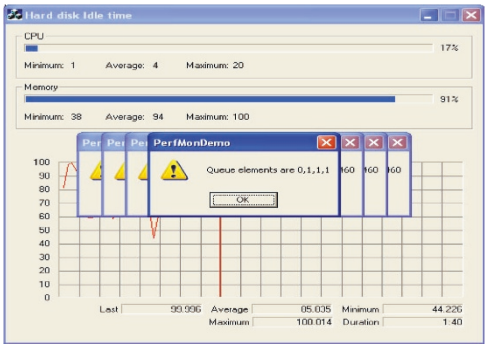
Internally, it was divided into two blocks, namely, Q-table and Energy table. Energy table receives input from the cost evaluation unit and Q-table receives input from Q-table updating unit. The purpose of this memory unit is to keep a store of the computed energy values of the three policies. To get a better accuracy, 32 bit output is chosen for the computed energy values. The Q-table stored in the memory helps in giving the appropriate values for Q-updating as previous Q-values are needed for current Q-computation.

Winner policy unit

This unit compares the computed Q-values for all policies and declares as output the policy which has maximum Q as the winner policy.

Reward unit

This unit receives input from cost evaluation unit and winner policy unit. If the winner policy has the least cost (i.e., energy), then the winner policy is rewarded by giving a weightage of +1, else the policy is given a negative weightage of −1.



```
CPU                                            17%
Minimum: 1     Average: 4     Maximum: 20
Memory                                         91%
Minimum: 38    Average: 94    Maximum: 100
Queue elements are 0,1,1,1
Last 99.996   Average 05.035   Minimum 44.226
              Maximum 100.014  Duration 1:40
```

Figure 3: Workload trace capture.

6. EXPERIMENTAL RESULTS

The system was modeled in VHDL (modelsim), simulated, and then synthesized using Xilinx (device-type Spartan 2E). The input workload traces were derived by capturing real time input generated by opening different applications on a desktop system, and the way the capturing was done is as shown in Figure 3.

This capture is done using Visual C++. It is a powerful tool to explore the system resources of windows operating system effectively. WinAPI functions are used for exploring the system resources. Here, mode programming is used. The system resources are explored by using PDH interfaces, which is available in Pdh.h and Pdhmsg.h header files. By using PdhAddCounter, the percentage idle time of the hard disk is captured. Active state is represented by 0 and idle state by 1.

The trace shows how real data captured has been buffered and stacked in a queue. This captured queue value is the active signal fed into the idle trace calculation unit to compute the idle period with clock time as reference.

The real time plot, when processor and hard disk are busy is shown in Figure 4.

For simulation purpose embedded devices with estimated active, sleep, idle, and wakeup powers were used. Policy switching takes place based on the dynamic traffic arrival rate. The experiment was carried out for different time durations that have been termed as episodes. Figure 5 shows how the policy transition takes place for a 4-episode case. Here policy 1 is timeout policy, policy 2 is greedy policy, policy 3 is predictive policy and policy 4 is always on policy. The positive and negative transitions indicate if the selected policy got a reward or a punishment at that instant of time. This concludes that policy switching takes place with incoming dynamic incoming traffic and further increase in learning time lead to less punishment or penalty in comparison to the rewards by a particular policy.

The experiment was carried out with a variety of policies and the energy savings obtained was observed. It was

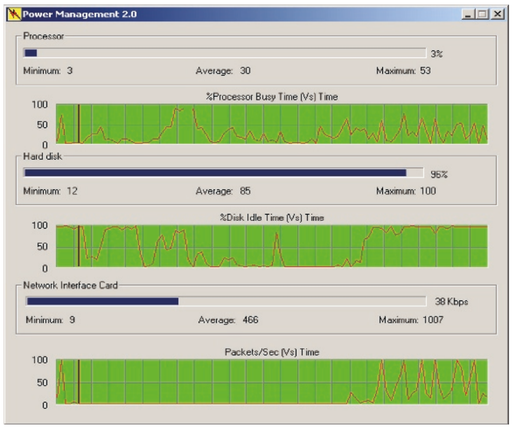


Figure 4: Real time capture plot when processor and hard disk are busy.

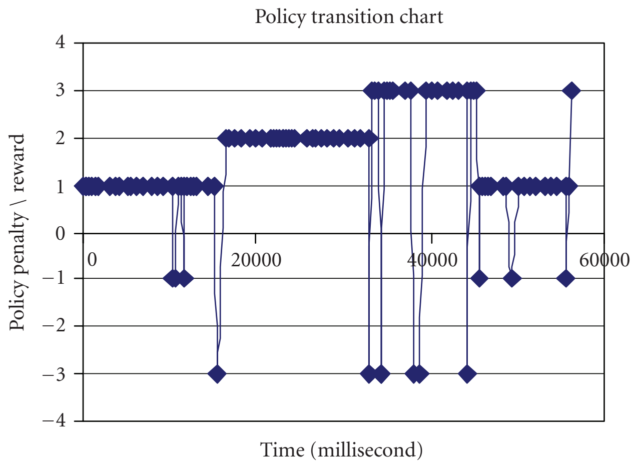


Figure 5: Policy transition for 4 episodes.

Table 2: Energy savings using RLTD.

Energy savings% for traces	IBM	Fujitsu	WLAN	HP
Trace1	19.92	12.71	25.65	10.75
Trace2	23.24	13.87	27.86	9.65
Trace3	21.54	15.65	24.67	12.87

observed that reinforcement learning with temporal difference has significant advantage over other policies as it dynamically settles on the best policy for any given workload. Table 2 shows the percentage energy savings achieved by reinforcement learning TD DPM using traces captured as workload. The energy savings was computed by running any single policy such as greedy, always on, timeout, deterministic Markov stationary policy and reinforcement learning TD.

7. IMPROVEMENT IN ENERGY SAVINGS

Temporal Difference Reinforcement Learning DPM has proved that it outperforms other DPM methods. The major advantage of this method over other methods is that it is able to exploit the advantages of individual policies. Real time workloads are highly random and nonstationary in nature, and hence any single policy fails at some point of time. OPBA (Online Probability-Based Algorithm) like policies works well when the probability distributions that help in determining the threshold point of state transition are highly clustered. RL method performance improves with time, and policy convergence takes place quickly and effectively.

The hardware solution suggested can be introduced in the ACPI (Advanced Configuration Power Interface), which links the application and the Power Manager. The output of the block winner policy guides the Power Manager to move the service provider to the appropriate low power state determined by the policy.

8. CONCLUSION

Dynamic power management is a powerful design methodology aiming at controlling performance and power levels of digital circuits and embedded systems, with the goal of extending the autonomous operation time of battery-powered systems.

In this work, Temporal Difference Reinforcement Learning-based intelligent dynamic power management (IDPM) approaches to find an optimal policy from a policy table, that is, precomputed. Hardware architecture has been proposed. The proposed approach deals effectively with highly nonstationary workloads. The results have been verified using the evolved hardware in FPGA. It concludes that Temporal Difference Reinforcement Learning is an effective scheme as the power saving is appreciable.

REFERENCES

[1] S. Irani, S. Shukala, and R. Gupta, "Competitive analysis of dynamic power management strategies for systems with multiple power savings states," Tech. Rep. 01-50, University of Irvine, Irvine, Calif, USA, September 2001.

[2] L. Benini, A. Bogliolo, G. A. Paleologo, and G. de Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813–833, 1999.

[3] Y.-H. Lu, T. Simunic, and G. de Micheli, "Software controlled power management," in *Proceedings of the 7th International Workshop on Hardware/Software Codesign (CODES '99)*, pp. 157–161, Rome, Italy, May 1999.

[4] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," in *Proceedings of the 36th Annual Design Automation Conference (DAC '99)*, pp. 555–561, New Orleans, La, USA, June 1999.

[5] Y.-H. Lu and G. de Micheli, "Comparing system-level power management policies," *IEEE Design and Test of Computers*, vol. 18, no. 2, pp. 10–19, 2001.

[6] S. K. Shukla and R. K. Gupta, "A model checking approach to evaluating system level dynamic power management policies

for embedded systems," in *Proceedings of the 6th IEEE International High-Level Design Validation and Test Workshop*, pp. 53–57, Monterey, Calif, USA, September 2001.

[7] C. Watts and R. Ambatipudi, "Dynamic energy management in embedded systems," *Computing & Control Engineering*, vol. 14, no. 5, pp. 36–40, 2003.

[8] E.-Y. Chung, L. Benini, and G. de Micheli, "Dynamic power management using adaptive learning tree," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '99)*, pp. 274–279, San Jose, Calif, USA, November 1999.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, UK, 1998.

[10] C. H. C. Ribeiro, "A tutorial on reinforcement learning techniques," in *Proceedings of International Conference on Neural Networks*, INNS Press, Washington, DC, USA, July 1999.

[11] R. A. Johnson, *Probability and Statistics for Engineers*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2001.