Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 13 (14): 2682-2687, 2013
ISSN 1812-5654 / DOI: 10.3923/jas.2013.2682.2687

Reinforcement Learning Integration in Dynamic Power Management

[1]Fa-Gui Liu, [1]Jin-Biao Lin, [1]Xiao-Yong Xing, [1]Bin Wang and [2]Jun Lin
[1]School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China
[2]China Great Wall Computer Shenzhen Corpration Ltd, Rm501, Greatwall Building, Kefa Road, Science and Industry Park Nanshan District, Shenzhen, People's Republic of China

Abstract: Policy is the core of system-level Dynamic Power Management (DPM). All traditional policies have their own shortcomings and heavy dependency on specific workload model in terms of effectiveness. In order to solve the problem in the partially observable environment where the workload model can hardly be predicted, this paper proposes an online-learning policy based on reinforcement learning. The policy produces a workload sequence L with the historical workload and next interval workload predicted by prediction tree. And we estimate the value of all pairs composed of the sequence and the time threshold (L-T) with Q-value based on the improved Q-learning algorithm. The smallest Q-value is chosen for each sequence and the corresponding time threshold is used to be the time threshold to perform time-out policy for the DPM system. Experimental results show that the proposed policy saves more power consumption range from 3.8% to 31.4%, comparing with traditional policies, while keeping acceptable performance.

Key words: Dynamic power management, reinforcement learning, online learning policy

INTRODUCTION

DPM policy is the research emphasis of system-level dynamic power management. It determines when spare parts can turn off or switch to a corresponding low power state to suit system performance need. The performance plays a key role in energy utilization. Traditional DPM policies (Zhao *et al*., 2008) mainly compose of timeout, predictive and stochastic model, which rely heavily on a specific workload model. However, workload is often unpredictable for a complex system due to its dependency on properties, input data and user context.

Timeout (Li *et al*., 1994; Douglis *et al*., 1995) is the simplest and most widely used policy, with its obvious disadvantage slow response in low power state and workload-model dependency in terms of determination on delay of the decision-making process. Currently, there are two main methods of prediction policy: One is the predicted wake-up technology that activates the components in advance according to system component' idle time predicted by strategy (Wang and Wu, 1997); the other is the predicted turnoff technology which turns down the components based on the predicted result (Srivastava *et al*., 1996). This kind of strategy relies heavily on the workload model and needs off-line calculation. Stochastic model policy, using the theory of probability, abstracts DPM into a stochastic optimization problem (Norman *et al*., 2002; Liu *et al*., 2009). Workload optimal results, however, can't be ensured by stochastic model policy for solving DPM optimization problems. The complexity of runtime environment makes it difficult to establish the precise workload model; hence, the solution of DPM problem is an approximate value. Moreover, because the models are too complex, we can't calculate it online.

In this paper, based on Reinforcement Learning (RL) (Sutton and Barto, 1998), we present an online learning policy to solve the problem that workload model can't be established precisely. Finally, the method is proved to be correct and effective.

RL-BASED POLICY MODEL

As shown in Fig. 1, it is a diagram showing a model of an online DPM policy. System runtime environment is abstracted into Service Requestor (SR), Service Queue (SQ) and Service Provider (SP). Service Requestor (SR), the workload source of system components, creates instructions operating on system hardware to generate a service request when the user or system does some sending and executing operation. This service does some sending and executing operation. These service requests

Corresponding Author: Fa-Gui Liu, School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China

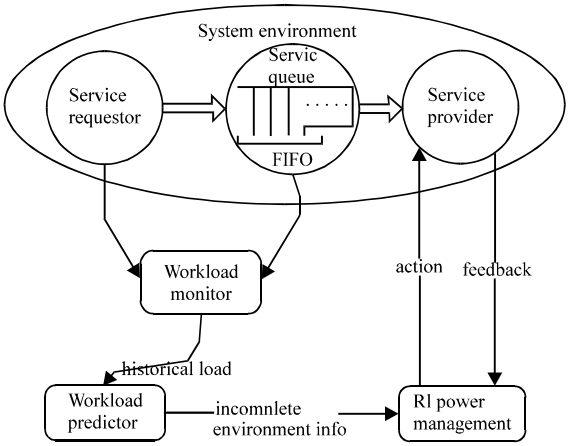

Fig. 1: Model of online policy

will be sent to a processing queue designed as a FIFO (first in first out) service queue for system components and will be cached through services information way. The service processing logic of system components is abstracted into a service provider. Then, the provider will get the request of the cache from the service queue and deal with it. Therefore, the system runtime environment can be represented as a collection of triples E = (SR, SQ, SP).

Workload monitor can observe the workload information in running time and record the information inside the monitor. Workload information observed by monitor is only a partial result of the system environment. Workload predictor will receive the discrete workload information from the monitor to predict the workload in the subsequent moment (Qiu *et al*., 2007). After that, workload sequence will be updated according to the workload we predicted. The action space of RL is a series of time values. In every decision-making time, RL power management chooses a time value from the time set on the basis of workload sequence. The value will serve as a threshold value for timeout policy which is executed by the policy enforcement module to control the energy consumption of service provider (system components). Then RL power management receives feedback information from the execution results and implements online updating policy.

WORKLOAD PREDICTION

Workload sequence: A finite set of N elements $L = \{L_1, \ldots, L_n\}$ is used to represent the historical workload of the system. We denote historical workload sequence by L and denote L_i the system workload at epoch I.

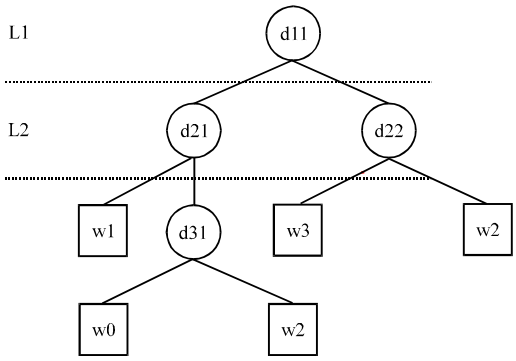

Fig. 2: Prediction tree

Suppose T_i denotes the system idle time at epoch i and then L_i can be given in the form of Eq. 1 ($1 \le i \le n$, T_{be} is threshold):

$$L_i = \begin{cases} 0, \text{ when } T_i < T_{be} \\ 1, \text{ when } T_i \ge T_{be} \end{cases} \quad (1)$$

For the running workload of system components, workload has similarities over a period of time. That is the workload of components in the next moment is largely depends on the latest workload status. Consequently, when the new workload is predicted, workload sequence will perform shift and update operation: from i = 1, successively make $L_{i+1} = L_i$ until i = n-1; then store the new workload prediction in L_1.

Fundamental structures: In the online policy model, workload predictor provides incomplete environment information to RL power management for RL model to achieve decision making and updating. This paper uses a workload-prediction tree based on learning tree algorithm (Bshouty *et al*., 1998). The tree is used for implementing the workload predictor in RL-based online policy model. We can predict the workload status of components in the next moment by means of switching the historical workload of system components monitored by workload monitor. Figur 2 shows a prediction tree generated by historical workload sequence $L = \{L_1, L_2\}$.

The significance of each element in Fig. 2 is as follows:

- **Circle:** Decision nodes, a path-matching branch point of prediction tree, denoted with d_{ik}, where i correspond to workload point L_i in historical workload sequence and k indicates that the layer has k decision points

- **Rectangle:** Weight node has a weighted value that represents the probabilities of which a branch is selected. We denote it with w_j and j indicates the j_{th} weight status
- **Solid line:** History branches are used for predicting path-matching
- **Dotted line:** It is used for predicting branches and represents the workload classification of system in the next moment. A weight node is associated with it.

Prediction tree is a hierarchical structure and the height of the tree does not exceed more than doubling the length of historical workload sequence. In addition, the i_{th} layer is corresponding with the workload L_i in historical workload sequence. Each weight node represents the probabilities of its associated prediction branch which is being selected. In other words, it's the workload prediction probabilities forecasted by prediction-tree in the next moment. Each weight node in the tree has a weighted value and tree nodes connect with each other through branches. However, weight nodes have no branch and the branches of decision nodes can have only two types: historical branch and prediction branch. When $L_i = 0$, value of L_i is corresponding with the left branch of decision node. Otherwise, it is corresponding with right branch when $L_i = 1$.

Prediction process: When entering a historical workload sequence L, a workload prediction process can be accomplished through matching a path that can reach the weight nodes. Suppose i indicates the i_{th} element in historical workload sequence L; n indicates the length of the sequence and k indicates the k_{th} decision node in i_{th} level of the prediction-tree. For $L = \{L_1, L_2, \ldots\ldots L_n\}$, the prediction process is as follows:

- Let i = 1, k = 1
- Matching historical workload L_i from decision node d_{ik}
- If $L_i = 0$, goto step (4); if $L_i = 1$, goto step (5)

If the left branch of decision node d_{ik} is a history branch, then we denote the root decision node which is corresponding with the left subtree of decision node d_{ik} by d_{pq}, let i = p, k = q, goto step (2); If the left branch of decision node d_{ik} is a prediction branch, then stop matching and choose the left prediction branch and predicted result of the system in the next moment is 0.

If the right branch of decision node d_{ik} is a history branch, then we denote the root decision node which is corresponding with the right subtree of decision node d_{ik} by d_{pq}, let i = p, k = q, goto step (2); If the right branch of decision node d_{ik} is a prediction branch, then stop matching and choose the right prediction branch and predicted result of the system in the next moment is 1.

Update process: Online policy model has the capacity for on-line updating when the system is running and the update is particularly important in workload prediction. According to the information when the system is running, by adjusting the value of weight nodes and splitting the nodes, we can adjust or update the prediction tree structure to improve the prediction accuracy.

Weight value in prediction tree represents the probability that a prediction branch is selected in the process of path matching prediction. When the system workload prediction is correct, with the purpose of improving the consistency of the same workload sequence next time, obviously we need increase the weight value of the selected prediction branches.

Weight node updating: With regard to the update of weight value, we use a finite state machine for modeling. The definition of weight finite state machine is a five-tuple, as shown in Eq. 2:

$$M = (Q, S, d, q_0, F) \quad (2)$$

Among them:

$Q = \{w_0 \ldots w_n\}$ and weight value $w_i > w_{i-1}$, $1 = i = n$

Σ = {"Successful prediction", "Unsuccessful prediction"}

$$\delta(w_i) = \begin{cases} W0, \text{ unsuccessful and } i = 0 \\ w_{i-1}, \text{ unsuccessful and } 0 < i \le n \\ w_{i+1}, \text{ successful and } 0 \le i < n \\ w_n \text{ successful and } i = n \end{cases}$$

$q_0 \in Q$

$F = \varnothing$

Figure 3 is a state transition diagram for four types of weight state machine.

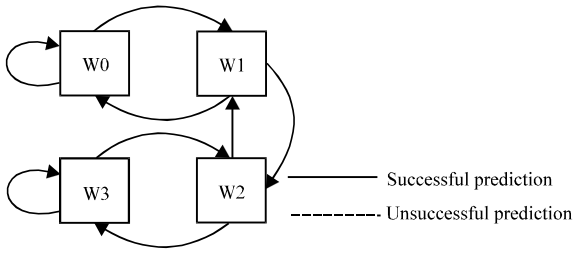

Fig. 3: Weight FSM

Weight node splitting: When the workload prediction result is faulty, we need to do a downward adjustment for mispredicted branch weight value. Besides, upward adjustment and split operation for correctly predicted branches are needed. The specific operation is as follows:

- Do the downward adjustment for mispredicted branch value. Adjust the mispredicted branch weight nodes according to the state transition function defined in Eq. 2
- Determine whether the last decision node in the predicted path can be split or not. If path length is equal to the length of the workload sequence, then goto step (3) without splitting the prediction tree. If path length is less than the length of the workload sequence, then goto step (4) with node splitting
- Do the upward adjustment for correctly predicted branches weight node. According to the state transition function defined in Eq. 2, turn the correct weight nodes into a higher state
- Increase the path length. Replace the real predicted weight node with a new decision node. The decision node is set as default weight node and the weight value of it is the state q_0 defined in Eq. 2
- Adjust the weight value of correctly predicted branches. For path matching, upward adjustment is needed for the weight value of correct branches. Then the operations end

REINFORCEMENT LEARNING ALGORITHM

By means of improving the Q-learning algorithm and combining timeout policy, online policy model implements the RL power management, which can make updates in real-time.

In the Q-learning algorithm, for each "action-status" pair in reinforcement learning model, there is an associated Q corresponding to it. Q-learning algorithm doesn't need environment model and we can get an adaptive policy by optimizing an iterative Q function. Reference (Watkins and Dayan, 1992) has firstly proposed the Q-learning algorithm and the Q value is defined in Eq. 3:

$$Q(s_t, a_t) = r_t + ?\max_{a_t}\{Q(s_{t+1}, a_t) \mid a_t \in A\} \quad (3)$$

Equation 3 shows the totalize reinforcement value when the model is executed according to the optimal action sequence in state s_t. r_t represents the value system environment feed back to the learning system at epoch t and ã is a policy discount coefficient. The updating of Q depend on the action selected by the system.

Traditional Q-learning algorithm does not require a priori system information. However, in the proposed online policy model, the workload predictor can provide historical workload information. Besides, we can know the status of the service provider model and the timeout threshold sequence which is preestablished. Obviously, online policy model can provide part of the system environment information. Based on this information, this section has improved Q-learning algorithm, to make sure it can be applied to solve the problem of dynamic power management.

For RL-based online policy model, the state space of model is all possible workload sequence L, while the action space is a set of timeout threshold T. Thus we get a set of pairs composed of each sequence and each timeout threshold. Each composed pair has a Q value which will be updated during decision process. A two-dimension Q table is formed with all composed pairs and the corresponding Q value. At each decision-making time, RL power management update the workload sequence L and selects a timeout threshold T, of which the Q value in "L-T" pairs is the minimum corresponding to the specific sequence L, as the threshold for timeout policy. After performing timeout policy with the selected timeout value T, we will update the Q value according to Eq. 4 below:

$$\begin{aligned} Q(L_t, T_t; ß) \leftarrow\ & Q(L_t, T_t; ß) + \\ & a\left[r(L_t, T_t; ß) + ?Q(L_{t+1}, T_{t+1}; ß) - Q(L_t, T_t; ß)\right] \end{aligned} \quad (4)$$

where, β represents the consumption and performance ratio; L_t represents the workload sequence generated by predictor at time t; T_t represents the timeout threshold used at time t and á represents the learning rate.

Equation 4 shows that the updating of Q for " L_t-T_t " at time t is under the influence of the value of Q for " L_{t+1}-T_{t+1}" at time t+1.

DPM strategy should not be overly aggressive and we need to make decisions under the condition of minimizing the loss of system performance so that the energy loss can reach the minimum. Therefore the feedback value of Q-learning algorithm should contain the evaluation of system energy consumption and performance. And the online policy feedback value is defined in Eq. 5:

$$R(L, T; \beta) = \beta c(L, T) + (1-\beta)\, p(L, T) \quad (5)$$

In Eq. 5, c (L, T) represents the energy consumption when T is used as the time threshold. Analogously, p (L, T) shown in Eq. 6 represents the performance loss in the same situation.

$$P(L, T) = q \times (1 + T_{tran}) \quad (6)$$

In Eq. 6, q represents the number of pending request in the service queue and T_{tran} represents the average conversion time in state transition by using T as the time threshold.

The article defines the feedback value of Q-learning algorithm by using a linear combination of the system energy loss and pending requests in the service queue. It is also reasonable feedback estimation. According to the experimental results, it shows that the average number of pending request in service queue is proportionate to each request latency time initiated by service requestor. The request latency time consist of the time requested for service queue and execution. Based on that fact, each Q in "L-T" pair, a weighted value, is the combination of the consumption of system components and delay request.

MODEL DETAIL

Decision epoch: Before implementing the DPM policy, we first define a series of decision epochs, when the system makes decisions according to different environment. The decision epochs are as following:

- The SP is in idle state and the SQ is empty
- The SP is in idle state and the SQ is not empty
- When the SP is entering the standby state, some requests from SR has just arrived (The SQ changes from empty state to not-empty state)
- When the SP is in standby state, some requests from SQ arrive and they are cached in SQ

At any decision epoch, the PM finds itself in one of the four conditions. Then it will make a decision according to different decision epochs and transform the SP to a corresponding state through the interfaces provided by device driver. In case (1), the PM will adopt the time-out policy based on improved Q-learning (described below); In case (2), the PM will activate the SP to handle the requests cached in SQ, in order to response the SR quickly. But the PM does not update the Q-learning algorithm. Both in case (3) and (4), the PM will also transform the SP to the active state and update the Q-learning algorithm. During the whole running time of system, the workload predictor caches the workload information at each decision epoch.

Policy detail: When the PM is in case (1), the workload predictor transforms the historical workload into the corresponding workload sequence L and use L to predict the next interval workload with the help of prediction tree. Then we get a new workload sequence with the predicted workload. The PM queries the Q table for the current workload sequence to choose the corresponding action, which is a timeout threshold, with minimum Q value. The PM performs timeout policy based on the timeout threshold to manage the SP. The timeout policy based on improved Q-learning in case (1) is as follows:

- Transform the historical workload into workload sequence L according to equation (1)
- Predict the next interval workload with prediction tree and update the sequence L to gain a new workload sequence
- Query the Q table for the sequence and choose the corresponding action with minimum Q value, which is a timeout threshold T
- Perform the timeout policy with timeout value T to transform the SP into the standby state when the idle time exceeds T

When the PM is in case (3) or in case (4), the SP will be activated to handle the coming requests. Meanwhile, the prediction tree and the Q value corresponding to last action will both be updated according to the time during which the SP stays standby state. The update algorithm is as below:

- If the time during which the SP stays standby state is larger than the time threshold T_{be}, the prediction is correct; Otherwise, prediction is wrong
- Adjust the branch weight value of prediction tree according to the update rule of prediction tree in part 3 (workload prediction)
- Calculate the feedback value of last action with Eq. 5 and Eq. 6
- Update the Q value with Eq. 4 produced by modified Q-learning algorithm

EXPERIMENTAL RESULTS

We implement this proposed online learning policy base on the DPM framework in (Liu *et al*., 2007) and performed our experiments using the IDE Hard Disk of the Hitachi Travel Star 4K40 Laptop. Meanwhile, we compare our proposed policy with other policies, such as time-out policy, predictive policy and stochastic policy. The experimental results are shown by table 1 below.

Here are some parameters used to compare the performance of different policies: P (W) is the average power consumption and the less the better. TS (s) is the average time keeping standby status and the more the better. TB (s) is the average time keeping idle status

Table 1: Experimental results comparing with other policies

Policy	P (W)	TS (s)	TB (s)	N_t	N_{wt}
Fixed timeout (Li *et al.* (1994)	1.59	0.221	0.316	80	17
Adaptive timeout (Douglis *et al.* (1995)	1.13	0.550	0.900	32	2
Exponent Average Algorithm (Wang, C.H. and Wu, A. (1997)	1.17	0.418	0.909	31	9
Renewal process Model (Liu *et al.* (2009)	1.44	0.390	0.328	62	19
RL-based	1.09	0.833	0.533	56	1

before transforming to standby status and the less the better. N_t is the total transform times. $N_{wt}$is the wrong transform times and the less the better.

The experimental results show that our proposed algorithm saves more power consumption range from 3.8-31.4%, comparing with traditional policies, while keeping acceptable performance. In addition, the RL-based policy has least wrong transformation and quicker response.

CONCLUSION

This study proposes an online-learning policy based on reinforcement learning. The policy produces a workload sequence L with the historical workload and next interval workload predicted by prediction tree. The reinforcement learning is used to update the Q value of all pairs composed of the sequence L and timeout threshold (L-T) with Q-learning algorithm. During the interaction with the system, our policy can find out a proper timeout value and use it to perform timeout policy for every workload sequence L. Experimental results show that the proposed policy outperforms the traditional policies in saving power consumption while keeping acceptable performance.

ACKNOWLEDGEMENTS

This study is supported by the university-industry cooperation major project of the ministry of education of Guangdong province: Independent Trusted Cloud Computing Platform and Cloud Terminal Development for E-Government Affairs, No. 2012B091000109; the university-industry cooperation project of Shenzhen: Cloud Terminal Product Development based on The Domestic CPU, No. CXY201107010263A.

REFERENCES

Bshouty, N.H., C. Tamon and D.K. Wilson, 1998. On learning decision trees with large output domains. Algorithmica, 20: 77-100.

Douglis, F., P. Krishnan and B. Bershad, 1995. Adaptive disk sp in2 down policies for mobile computers. Proceedings of the 2nd USN IX Symposium on Mobile and Location Inde2 Pendent Computing, (SMLIPC '95), USEN IX Association, Berkeley, pp: 12-37.

Li, K., R. Kumpf, P. Horton and T. Anderson, 1994. A quantitative analysis of disk drive power management in portable computers. Proceedings of the International Conference on USENIX Winter 1994 Technical, (ICUWT'94), USENIX Association Berkeley, CA, USA., Pages: 22-22.

Liu, F.G., W.P. Mai and K.Y. Huang, 2007. Extension and realization of dynamic power management architecture. J. South China Univ. Technol., 5: 60-64.

Liu, F., Z. Wu, W. Mai, 2009. Renewal-theory-based hard disk power management strategy optimization. Proceedings of the 5th International Joint Conference on INC, IMS and IDC, August 25-27, 2009, pp: 511-515.

Norman, G., D. Parker, M. Kwiatkowska, S.K. Shukla and R.K. Gupta, 2002. Formal analysis and validation of continuous time Markov chain based system level power management strategies. Proceedings of the 7th International High-Level Design Validation and Test Workshop, (IHLDVTW'02), Computer Society Washington, DC., USA., pp: 45-50.

Qiu, Q., Y. Tan and Q. Wu, 2007. Stochastic modeling and optimization for robust power management in a partially observable system. Proceedings of the International Conference on Design, Automation and Test in Europe and Exhibition, April 16-20, 2007, Nice, pp: 1-6.

Srivastava, M., A.P. Chandrakasan and R.W. Brodersen, 1996. Predictive system shutdown and other architectural techniques for energy efficient programmable computation. Very Large Scale Integration Syst. Trans., 4: 42-55.

Sutton, R.S. and A.G. Barto, 1998. Reinforcement learning: An introduction. MIT Press, Cambridge, MA.

Hwang, C.H. and A.C.H. Wu, 1997. A predictive system shutdown method for energy saving of event-driven computation. Proceedings of the International Conference on Computer Aided Design, November 9-13, 1997, San Jose, CA, USA., pp: 28-32.

Watkins, C.J.C.H. and P. Dayan, 1992. Q-learning. Machine learning, 8: 279-292.

Zhao, X., X.Q. Chen, Y. Guo and F.Q. Ya, 2008. A survey on operating system power management. J. Comput. Res. Dev., 6: 817-824.