Using Probabilistic Model Checking for Dynamic Power Management

Gethin Norman[1], David Parker[1], Marta Kwiatkowska[1],
Sandeep Shukla[2], and Rajesh Gupta[3]

[1] School of Computer Science, University of Birmingham
Birmingham B15 2TT, United Kingdom
[2] Bradley School of Electrical and Computer Engineering, Virginia Tech
Blacksburg, VA 24060, USA
[3] Department of Computer Science and Engineering,
University of California at San Diego, La Jolla, CA, USA

Abstract. We present an approach to deriving stochastic Dynamic Power Management (DPM) strategies that enables us to design both discrete time and continuous time Markov chain based strategies, in a formal and uniform framework. This is a novel application of formal model checking of probabilistic systems in the area of system design. This approach allows us to obtain expected performance measures of the derived strategies by automated analytical means without expensive simulations. Moreover, one can formally prove various probabilistically quantified properties pertaining to buffer sizes, delays, energy usage etc., for each derived strategy. Comparison of various strategies under various stochastic behavioral assumptions can also be done formally without simulation. In this paper, we illustrate how we have implemented our uniform approach in the PRISM model checker framework and present results from realistic DPM scenarios based on a disk-drive example with multiple power management states.

1 Introduction

Power Management is an important area of research [1, 3, 15, 18, 9] because of an increasing trend in the usage of portable, mobile, and hand-held electronic devices. These devices usually run on batteries and any savings in power usage translate to extended battery life. Various approaches to low power design have been explored, at device, circuit and micro-architecture level. System level power management exploits application characteristics and manages various system devices for power optimization. System components such as network interface cards, disk drives, and DRAM, etc., are manufactured with a number of power modes which can be changed by an operating system through standard APIs such as ACPI [8] and power-aware API [13]. However, in order to take advantage of these power modes and APIs, the power management strategies need to be implemented at the O/S level. In this context, Dynamic Power Management (DPM) strategies refer to strategies that attempt to make power mode changing decisions based on information about their usage pattern available at runtime. The objective of such strategies is to minimize power consumption, while minimizing effect on performance.

Among the various methodologies for deriving DPM strategies, Stochastic Modelling has been extensively studied in the literature [16, 17, 1, 3, 21]. Stochastic Modelling of device behaviours, input characteristics and inherent delays in the system, have led to stochastic modelling based techniques including: *Discrete Time Markov Chains* [2, 12], *Continuous Time Markov Chains* [15–17, 11], and Renewal Theory based methods [21] to derive stochastic DPM strategies. These existing methodologies depend on formulating a stochastic optimization problem to minimize the average energy consumption and the average delay in system response. In this paper, we show how the probabilistic model checker PRISM [10, 14], can be used as a uniform framework to derive stochastic strategies. This paper substantially complements our previous work on Continuous-Time Markov Chain based strategies in [11] and conclusively shows that our PRISM based framework provides a uniform methodology for deriving, analyzing, and validating stochastic DPM strategies. In particular, instead of restricting the derivation of the strategies based on optimizing average cases, one can parameterize the strategies on required extremal properties such as maximal delay bounds, probabilistically quantified delay bounds, and maximal buffer size etc. On another note, the existing research literature relied heavily on expensive trace based simulations to analyse variational effects such as varying the arrival rate of requests from one standard distribution to another, whereas the PRISM framework can perform such analysis quite easily without simulation. Moreover, one can formally prove various properties pertaining to buffer sizes, delays, energy usage etc, for any given strategy. In summary, this paper presents a uniform framework for stochastic Dynamic Power Management with additional advantages, by utilising a probabilistic model checker.

In the next section, we introduce the basic concepts of probabilistic model checking and the PRISM tool. We also provide background information on stochastic modeling based power management and existing literature on the topic. In Section 3, we describe how we use the PRISM framework to derive DPM strategies and parameterize the approach to obtain various strategies based on varying stochastic assumptions on the environment. We also present experimental results in this section. Section 4 contains discussion and future work.

2 Background

In this section, we briefly outline concepts from system level power management, probabilistic model checking, the tool PRISM, and stochastic modeling for DPM.

2.1 System Level Power Management

Power management in embedded computing systems is achieved by actively changing the power consumption profile of the system by putting its components into power/energy states which are sufficient to meet functionality requirements. For example, an idling component (such as a disk drive) can be put into a slow-down or shutdown state. Of course, bringing such a component into an active state may then require additional energy and/or latency to service the tasks. The DPM problem can be defined as:

> *Given a power managed component, such as a disk drive or a network interface card, derive a randomized power management strategy which minimizes average power dissipation, under the constraint that the average delay suffered by requests coming for service by the component is bounded by an a priori constant.*

We also seek to develop probabilistic guarantees on the worst case and best case scenarios, that enables us to quantify the effectiveness of the DPM strategy. These measures include the worst case delay, worst and best power consumption. We also want to obtain information on buffer sizes and other design space parameters (in a probabilistic sense). In other words, we want to know the values of these parameters as a function of probabilities of a certain delay or power consumption in the system.

Dynamic Power Management (DPM) attempts to make optimal decisions (usually under the control of the operating system) at runtime based on dynamically changing system state, functionality and delay requirements [6, 22, 3, 20, 19, 5, 9]. A survey of the DPM techniques can be found in [1]. The authors in [1] classify DPM strategies into two main groups: *predictive schemes*, and *stochastic optimum control* schemes. Predictive schemes attempt to predict the timing of future input to the system and, based on such prediction, schedule shutdown (usually to a single lower power state) of the system. Stochastic optimum control is a well-researched area [1, 21, 2, 5, 15, 17]. The chief characteristic of these approaches is construction (and validation) of a mathematical model of the system that lends itself to a formulation of a stochastic optimization problem, and then creation of strategies to guide the system power profile that achieve the highest power savings in the presence of the uncertainty related to system inputs.

While several useful and practical techniques have been developed using predictive and stochastic optimum control schemes, as of now, it is difficult to develop bounds on the quality of these results without extensive simulations and/or model justification.

Stochastic control oriented dynamic power management work [2, 15] has relied on modelling inter-arrival times using an exponential distribution. In practice, such stochastic modelling seems to work well for specific kinds of applications. However, the approaches varied in the model of time; for example in [2], the arrival process and service process are all modelled as discrete time Markov chains, whereas in [15] these are modelled with continuous time Markov chains. In more recent work, such as in [21], extended models to incorporate more general stochastic processes for modelling event arrivals have been considered. In this paper, we focus on discrete-time Markov chains following [2].

2.2 Probabilistic Model Checking

At a high level of abstraction, a model can often be simplified, for instance by replacing determinism by nondeterminism. However, complete nondeterminism often leads to an inability to prove any useful property of such systems. As a result, probabilities are often used to abstract some of the low level disregarded information, and a quantified nondeterminism can be represented in a probabilistic model. For example, if one looks at the the service time by a disk-drive per request, if one models the functionality of the disk-drive, the queues, the environment, the device drivers and the operating system and the architecture, one may be able to predict exactly for each request how much service time is required. However, often it is inconvenient, and certainly it is not easily amenable to formal analysis, to model disk-drive behaviour in such detail. However, by observing its behaviour for a sufficiently long time, one can infer that the disk-drive exhibits probabilistic behaviour with certain parameters. As an example, the service time per request is often modelled as an exponential distribution with a mean of 3 ms. Such information is then useful in modelling and analysing this behaviour and also in devising probabilistic algorithms for managing such devices.

2.3 Probabilistic Model Checking and PRISM

Probabilistic model checking refers to a state space analysis technique for probabilistic finite state systems. The system is usually specified as state transition systems, with probability measures on the rate of transition, and a probabilistic model checker applies algorithmic techniques to analyse the state space, and calculate probabilities of reaching different states etc. Given probabilistic assertions about the system, probabilistic model checkers can prove or disprove such assertions by means of algorithmic techniques [4].

We use PRISM [10, 14], a probabilistic model checker developed at the University of Birmingham. It supports analysis of three types of probabilistic models: discrete-time Markov chains, continuous-time Markov chains and Markov decision processes. These models are described in a high-level language based on guarded commands with probabilistic information attached to them. Properties of the models to be analysed are specified in the probabilistic temporal logics PCTL and CSL. This allows us to express various probabilistic properties such as “some event happens with probability 1”, and “the probability of cost exceeding C is 95%”. The model checker then analyses the model and checks if the property holds in each state.

2.4 Stochastic Modelling for DPM

Stochastic modelling based approaches to DPM have been based on the framework of stationary discrete-time Markov chains [12], continuous-time Markov Chains [15] or their variants [21].

Irrespective of whether they are based on stationary discrete-time Markov chains, continuous-time Markov chains and their variants, existing methodologies depend on modelling the input arrival process and the behaviour of power managed components by creating the stochastic matrices or generator matrices for these processes by hand, and then creating and solving optimization problems from those to optimize the average case. One novelty of this work is that we express the behaviour of the input generator and power managed component, as well as the power manager, in a high level probabilistic language for expressing stochastic state machines. This allows automatic generation of the matrices; the rest of the required computation for designing strategies is then carried out in the model checking framework. In [16], the power manager and managed components are modelled using stochastic Petri nets. This allows automatic generation of the stochastic matrices and the formulation of the optimization problems. These *exact* optimization problems are meant to optimize the *average* energy usage while minimizing *average* delay. They are usually validated by simulation to check for the soundness of the modelling assumptions, and effectiveness of the strategies in practice [15, 12]. Since probabilistic model checking is inherently exhaustive in its search among all possible scenarios, more useful information can be obtained about the design space than using simulation. For example, optimal buffer sizes, average delays, probabilities of various corner case scenarios etc, and probability based comparisons between various delay-cost possibilities (obtainable by competing DPM strategies) can easily be predicted.

3 Experimental Results

We obtained all the data used by the authors of [2] directly from the authors, and used PRISM to model the DTMC based strategy derivation problem. In this section we describe the model of the disk drive as in [2], and briefly describe how we modelled the discrete case in PRISM, and what other measures we obtained for the derived strategies, and describe some of the properties proven.

3.1 Multiple Power States

Device and component manufacturers provide multiple power states which can be controlled under the operating system through these standardized APIs. Table 1-2 shows the data from [2] for a 5-state device (a commercially available hard disk drive [7]). Note that, it is only in state *active* that the device can perform data reads and writes. In state *idle* the disk is spinning while some of the electronic components of the disk drive have been switched off. The state *idlelp* (idle low power) is similar except that it has a lower power dissipation, the states *stby* and *sleep* correspond to the disk being spun down.

3.2 The System Model

We consider the system model as illustrated in Figure 1. The model consists of a Service Requester(SR), a Service Provider(SP), Service Request Queue(SRQ),

	sleep	*stby*	*idlelp*	*idle*	*active*
Power (W)	0.1	0.3	0.8	1.5	2.5
Service Time (ms)	0	0	0	0	1

Table 1. Average power consumption and service times for the managed device [2]

	active	*idle*	*idlelp*	*stby*	*sleep*
active	–	1ms	5ms	2.2sec	6sec
idle	1ms	–	5ms	2.2sec	6sec
idlelp	5ms	–	–	2.2sec	6sec
stby	2.2sec	–	–	–	6sec
sleep	6sec	–	–	–	–

Table 2. Average transition times for state transition of the managed device [2]

and the power manager(PM) modules. In both discrete-time as well as continuous-time models, this level of description is the same. However, they differ in how time is represented. In this paper time is considered to be discrete, whereas in [11] we consider continuous time.

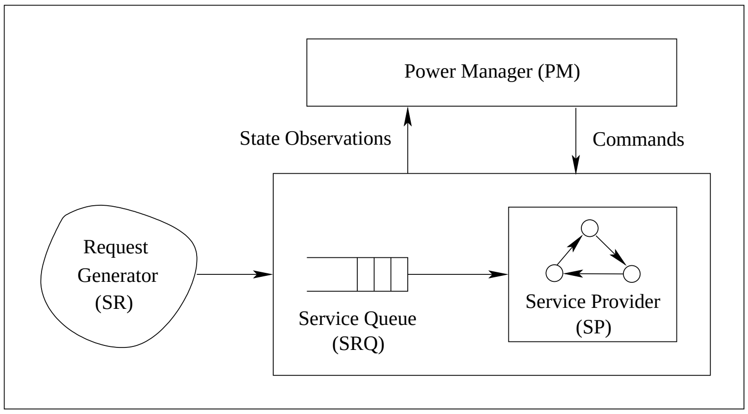

Fig. 1. The System Model.

3.3 Modeling in PRISM

To model the system we have chosen a time resolution of 1ms based on the fastest possible transition performed by the SP.

To model the PM deciding which state SP should move to at the start of each time step, we split each time step of the system into two parts: in the first the PM (instantaneously) decides what the SP should do next (based on the current state), and in the second the system makes a transition (with the SP's move based on the choice made by the PM). To achieve this we include the CLOCK module given below.

```
module CLOCK

    c : [0..1] init 0;

    [tick1] c = 0  →  c′ = 1;
    [tick2] c = 1  →  c′ = 0;

endmodule
```

The PM is then constructed to synchronize with the CLOCK on tick1, while the remaining components are constructed to synchronize with the CLOCK on tick2.

Modelling the PM As mentioned above, the PM synchronises with the clock on tick1 and bases its choices on the current state of the system. A generic PM has the following form:

```
module PM

    pm : [0..4];
    // 0 − go to busy, 1 − go to idle, 2 − go to idlelp
    // 3 − go to standby and 4 − go to sleep

    [tick1] cond1  →  p0_1 : sp′=0 + p1_1 : sp′=1+
                      p2_1 : sp′=2 + p3_1 : sp′=1 + p4_1 : sp′=4;
    [tick1] cond2  →  p0_2 : sp′=0 + p1_2 : sp′=1+
                      p2_2 : sp′=2 + p3_2 : sp′=1 + p4_2 : sp′=4;
                      ⋮

endmodule
```

For example, if a state of the system satisfies cond_1 then the PM decides that with probability $\mathrm{p0}_1$ the SP moves to *active*, with probability $\mathrm{p1}_1$ the SP moves to *idle*, with $\mathrm{p2}_1$ to *idlelp*, $\mathrm{p3}_1$ to *standby*, and $\mathrm{p4}_1$ to *sleep*.

Modelling the SP Recall that the SP synchronises with the clock on tick2 and the behaviour of the SP depends on the PM. Furthermore, since a time resolution of 1ms has been chosen, to correctly model transitions with delays longer than this time resolution *transient* states are introduced. For example, the transient state *active_idlelp* is used to model the non-unitary time transition from *active* to *idlelp*. Note that we suppose that the power dissipation in these transient states is high (2.5W). The module representing the SP is given below.

```
module SP

    sp : [0..10] init 9;
    // 0=active, 1=idle, 2=active_idlelp, 3=idlelp, 4=idlelp_active
    // 5=active_stby, 6=stby, 7=stby_active, 8=active_sleep
    // 9=sleep, 10=sleep_active

    // states where PM has no control (transient states)
    [tick2] sp=2  →  0.75 : (sp′=2) + 0.25 : (sp′=3);
    [tick2] sp=4  →  0.25 : (sp′=0) + 0.75 : (sp′=4);
    [tick2] sp=5  →  0.995 : (sp′=5) + 0.005 : (sp′=6);
    [tick2] sp=7  →  0.005 : (sp′=0) + 0.995 : (sp′=7);
    [tick2] sp=8  →  0.9983 : (sp′=8) + 0.0017 : (sp′=9);
    [tick2] sp=10  →  0.0017 : (sp′=0) + 0.9983 : (sp′=10);
    // PM: goto active
    [tick2] pm=0 ∧ (sp=0 ∨ sp=1) → sp′=0;
    [tick2] pm=0 ∧ sp=3 → sp′=4;
    [tick2] pm=0 ∧ sp=6 → sp′=7;
    [tick2] pm=0 ∧ sp=9 → sp′=10;
    // PM: goto idle
    [tick2] pm=1 ∧ (sp=0 ∨ sp=1) → sp′=1;
    [tick2] pm=1 ∧ (sp=3 ∨ sp=6 ∨ sp=9) → sp′=sp;
    // PM: goto idlelp
    [tick2] pm=2 ∧ (sp=0 ∨ sp=1) → sp′=2;
    [tick2] pm=2 ∧ (sp=3 ∨ sp=6 ∨ sp=9) → sp′=sp;
    // PM: goto stby
    [tick2] pm=3 ∧ (sp=0 ∨ sp=1 ∨ sp=3) → sp′=5;
    [tick2] pm=3 ∧ (sp=6 ∨ sp=9) → sp′=sp;
    // PM: goto sleep
    [tick2] pm=4 ∧ (sp=0 ∨ sp=1 ∨ sp=3 ∨ sp=6) → sp′=8;
    [tick2] pm=4 ∧ sp=9 → sp′=9;

endmodule
```

Modelling the SR and SRQ As mentioned above, both the SRQ and the SR will synchronise with the clock on tick2. The SR has two states: *idle* where no requests are generated and *1req* where one request is generated per time step (1ms). The transitions between these states is based on time-stamped traces of disk access measured on real machines [2]. The module of the SR is given by:

```
module SR

    sr : [0..1] init 0;
    // 0 - idle and 1 - 1req

    [tick2] sr=0 → 0.898 : (sr'=0) + 0.102 : (sr'=1);
    [tick2] sr=1 → 0.454 : (sr'=0) + 0.546 : (sr'=1);

endmodule
```

In the case of the SRQ, to model the arrival and service of requests the transitions of the SRQ are dependent on the state of both the SR and SR. Since either the SR is in state *idle* and no requests arrive, or in state *1req* and one request arrives, and on the other hand the SP can only serve requests when it is in state *active*, the module of the SRQ is presented below.

```
const QMAX = 2; //maximum size of the queue

module SRQ

    q : [0..QMAX] init 0; // size of queue

    // SP is active
    [tick2] sr = 0 ∧ sp = 0 → q' = max(q − 1, 0);
    [tick2] sr = 1 ∧ sp = 0 → q' = q;
    // SP is not active
    [tick2] sr = 0 ∧ sp > 0 → q' = q;
    [tick2] sr = 1 ∧ sp > 0 → q' = min(q + 1, QMAX);

endmodule
```

Modelling a Finite Time Horizon The policies we have calculated are the optimum policies for minimizing power consumption under different performance constraints, where these constraints concern the average size of the queue. In addition, we suppose that there is a time horizon of one million time steps. To model this horizon we include an additional module representing a battery which has an expected life span of 1 million time steps.

```
module BATTERY

  bat : [0..1] init 1;
  // 0 - battery off and 1 - battery on

  [tick2] bat = 1 → 0.999999 : (bat'=1) + 0.000001 : (bat'=0);

endmodule
```

Note that, once the battery reaches state 0, it cannot perform the action tick2 which prevents the system from performing the action tick2, and hence prevents the rest of the system from moving (i.e. the states where bat = 0 act as sink states).

3.4 Interpretation of Results

We have designed generic models of the the Power Management system in PRISM s input description language, for the disk drive model of [2]. Then, using PRISM, we are able to construct the full stochastic matrix of a system, and hence construct the optimization problem whose solution is the optimal policy. Moreover, once the optimal policy is found, by using the generic description we can construct a model of the power manager module corresponding to this policy and investigate its performance. Varying the average queue length, we obtain different stochastic policies where the calculation is done by generating the matrices in PRISM and formulating and solving the linear optimization problem in MAPLE symbolic solver.

The PRISM representation consists of a number of modules representing the different components of the system, namely the power manager (PM), service provider (SP), service requester (SR) service request queue (SRQ). In [2], the authors created the SR model using disk usage traces, and we have used their model verbatim. In order to parameterize the results on various alternative strategies, we had to represent different stochastic strategies. For that, one needs only to modify the PM module. This way we analyse performance under varied constraints and Figure 2 illustrates the power consumption versus performance both in terms of the average queue size and the average number of lost requests for the case study. The computed strategies correspond to optimizing the power consumption under a constraint on the average queue size.

We now examine representative results that demonstrate the utility and power of our methodology. A representative selection of results are depicted graphically in Figures 2–5. The delay measure is given by the queue length (which indicates how much delay a new incoming request will suffer). We also show the graphical representation of the probabilities of losing N requests by a certain time, and probabilities of being served by a certain time T. The graphs are shown for different strategies, where the strategies are derived under various average queue length constraints. We note that probabilistic properties proven

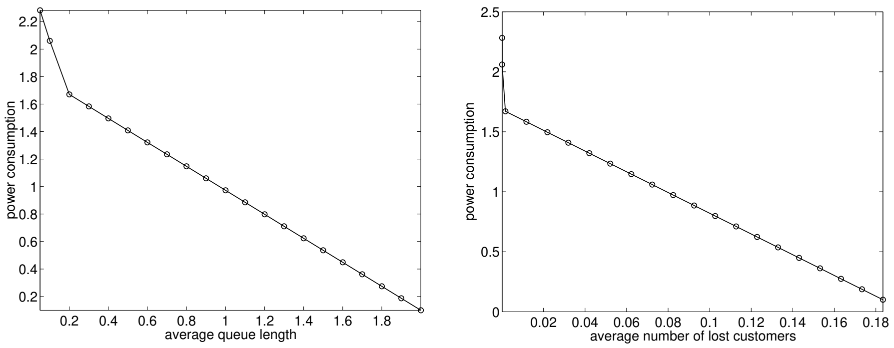

Fig. 2. Power versus performance for the disk-drive

about the strategies, or the variational effects of replacing the assumptions on various probability distributions are not shown here. In [11] we have shown how those can be done for Continuous-Time models, and their extension to Discrete-Time cases is similar and omitted here for brevity. The results (obtained using PRISM) presented in Figure 3 show the expected power consumption by time T, the expected queue size at time T, and expected number of lost customers by time T for a number of different values of T. Next, in Figure 4 we depict the probability that a request is served by time T, given that it arrived into a certain position in the queue. In Figure 4 the results are presented for a number of performance constraints and values of T, both in the case when the request arrives into the first and also second position of the queue. Finally, Figure 5 shows, from the initial state, the probability that N requests get lost by time T for different performance constraints.

These results show that strategies which consume less power have in general larger queue sizes (both in the long run and at time T), and a larger number of lost requests (both on average and by time T). Furthermore, the graphs show that the probability that requests get served or lost within a time bound increases for those strategies that consume less power. These results are to be expected, since to reduce power the strategies must force the service provider to spend more time in low power states, that is, those states which cannot service requests (for example, sleep and standby). Moreover, the fact that the results for the expected queue size at time T initially increase and then decrease, follows from the fact that the strategies wait for the queue to become full before switching the SP on. Finally, we note that the results concerning the average power consumption and average queue size given in Figure 2 coincides with the results presented in [2].

4 Conclusion and Future Work

We show that probabilistic model checking can provide a uniform framework for deriving, analyzing and validating DPM strategies. This is accomplished by virtue of derivation and analysis of strategies when the system is modelled

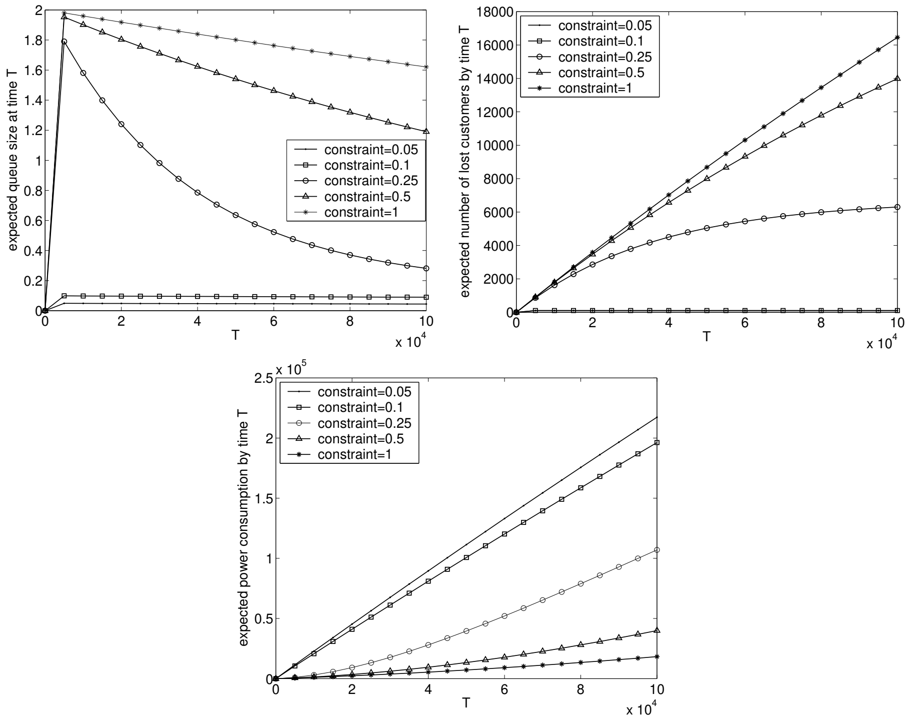

Fig. 3. Power and performance by time T (ms)

using discrete-time Markov chain as in [2], and by virtue of our previous results for the continuous time Markov chain models [11]. As opposed to deriving the strategies to minimize average case power, and delays as done in the the recent literature, we naturally obtain the power management policies under varied delay constraints (rather than only the one that minimizes average delay). The ongoing work focuses on building an analytical framework that derives and analyses strategies for more general probabilistic assumptions.

Acknowledgments

This work was supported by NSF grant CCR-0098335, EPSRC grant GR/N22960, QinetiQ, SRC, and DARPA/ITO supported PADS project under the PAC/C program. Part of the work was possible by a visiting fellowship from University of Birmingham.

We would also like to thank the authors of [2] for providing us with the details specification of their hard-drive specification.

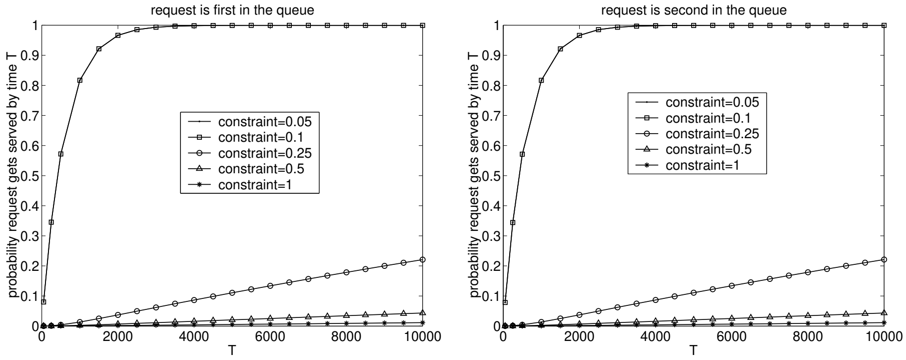

Fig. 4. Probability that a request is served by time T (ms)

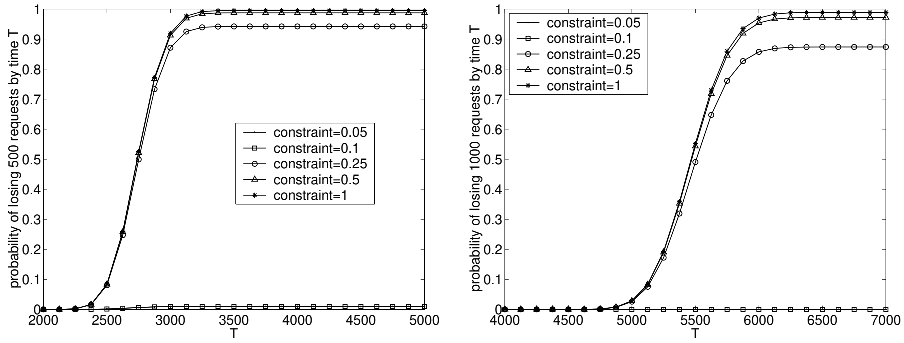

Fig. 5. Probability that N requests gets lost by time T (ms)

References

1. Benini, L., Bogliolo, A., and Micheli, G. D. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (TVLSI) Systems*, 8(3):299–316, 2000.
2. Benini, L., Bogliolo, A., Paleologo, G., and Micheli, G. D. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, 1999.
3. Benini, L., De Micheli, G., and Macii, E. Designing Low-power Circuits: Practical Recipes. *IEEE Circuits and Systems Magazine*, 1(1):6–25, 2001.
4. Bianco, A. and de Alfaro, L. Model checking of probabilistic and nondeterministic systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.
5. Chung, E. Y., Benini, L., Bogliolo, A., and Micheli, G. D. Dynamic Power Management for Non-Stationary Service Requests. In *Proceedings of the Design Automation and Test Europe*, 1999.
6. Hwang, C.-H., Allen, C., and Wu, H. A Predictive System Shutdown Method For Energy Saving of Event-Driven Computation. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 28–32, 1996.

7. Technical specifications of hard drive IBM Travelstar VP 2.5inch. `http://www.storage.ibm.com/storage/oem/data/travvp.htm`.
8. Intel and Microsoft and Toshiba. Advanced Configuration and Power Interface Specification. Website, 1996.
9. Irani, S., Shukla, S., and Gupta, R. Competitive Analysis of Dynamic Power Management Strategies for Systems with Multiple Power Saving States. In *Proceedings of the Design Automation and Test Europe Conference*, 2002.
10. Kwiatkowska, M., Norman, G., and Parker, D. PRISM: Probabilistic symbolic model checker. In Field, T., Harrison, P., Bradley, J., and Harder, U., editors, *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.
11. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S., and Gupta, R. Formal analysis and validation of continuous time Markov chain based system level power management strategies. In Rosenstiel, W., editor, *Proc. 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02)*, pages 45–50. OmniPress, 2002.
12. Paleologo, G. A., Benini, L., Bogliolo, A., and Micheli, G. D. Policy Optimization for Dynamic Power Management. In *Proceedings of Design Automation Conference*, pages 182–187. ACM Press, 1998.
13. Pereira, C., Gupta, R., Spanos, P., and Srivastava, M. *A Power Aware API*, chapter 8 - Power-Aware API for Embedded and Portable Systems. Kluwer Academic Publishers, 2002.
14. PRISM web page. `http://www.cs.bham.ac.uk/~dxp/prism/`.
15. Q. Qiu and M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In *Proceedings of Design Automation Conference*, pages 555–561. ACM Press, 1999.
16. Q. Qiu and Q. Wu and M. Pedram. Dynamic power management of complex systems using generalized stochastic petri nets. In *Proceedings of Design Automation Conference*, pages 352–356. ACM Press, 2000.
17. Q. Qiu, Q. Wu and M. Pedram. Stochastic Modeling of a Power-Managed System: Construction and Optimization. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 194–199, 1999.
18. Ramanathan, D., Irani, S., and Gupta, R. An Analysis of System Level Power Management Algorithms and their effects on Latency. *IEEE Trans. on Computer Aided Design*, 21(3):291–305, 2002.
19. Ramanathan, D., Irani, S., and Gupta, R. K. Latency Effects of System Level Power Management Algorithms. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 350–356, 2000.
20. S. Shukla and R. Gupta. A Model Checking Approach to Evaluating System Level Power Management for Embedded Systems. In *Proceedings of IEEE Workshop on High Level Design Validation and Test (HLDVT01)*, pages 53–57. IEEE Press, 2001.
21. Simunic, T., Benini, L., and Micheli, G. D. Event Driven Power Management of Portable Systems. In *In the Proceedings of International Symposium on System Synthesis*, pages 18–23, 1999.
22. Srivastava, M. B., Chandrakasan, A. P., and Broderson, R. W. Predictive Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation. *IEEE Trans. on VLSI Systems*, 4(1):42–54, 1996.