Formal Methods for Dynamic Power Management

Rajesh K. Gupta
Computer Science & Engineering
UC San Diego, La Jolla, CA

Sandy Irani
School of Information & Computer Science
UC Irvine, Irvine, CA

Sandeep K. Shukla
FERMAT Lab, Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA

ABSTRACT

Dynamic Power Management or DPM refers to the problem of judicious application of various low power techniques based on runtime conditions in an embedded system to minimize the total energy consumption. To be effective, often such decisions take into account the operating conditions and the system-level design goals. DPM has been a subject of intense research in the past decade driven by the need for low power in modern embedded devices. We present an overview of the formal methods that have been explored in solving the system-level DPM problem. We show how formal reasoning frameworks can potentially unify apparently disparate DPM techniques.

1. INTRODUCTION

1.1 Dynamic Power Management

Minimization of power consumption is rapidly becoming the chief optimization criterion in system design for a range of systems from general purpose computing to embedded, mobile computing devices. To be useful, such optimizations must often be done against other competing criteria, such as functionality delivery within performance and timing constraints. Often a balance is sought between the amount of computing (as in local processing) verus the amount of communication that would be needed as computation is reduced [43, 37, 41].

Our focus in this paper is on system-level dynamic power management that can be implemented in the operating system. These power saving measures allow for observation and incorporation of application behavior [8, 9, 6] in a *Power Manager* (PM). The PM can change the power consumption of a device through selection of shutdown/sleep/wakeup states for the device, or by changing its speed through voltage or frequency scaling. For historical reasons, system level DPM generally refers to the techniques that save energy in devices by turning these on and off under operating system control. From an OS point of view, shutdown/wakeup remains a key decision in effective power management even as the effectiveness of speed-scaling is sometimes called into question because of the process technology effects such as dominance of leakage [27]. DPM has been studied by several research groups [15, 44, 9, 8, 40, 36, 11, 39], as well as concerted industry efforts such as Microsoft's OnNow [26] and ACPI [16].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2003 ACM 0-89791-88-6/97/05 ...$5.00.

1.2 Previous Survey

A survey of the DPM techniques developed prior to 2000 can be found in [6]. In this extensive review, the solution approaches to DPM have been classified into *predictive schemes* and *stochastic optimum control* schemes. Predictive schemes attempt to predict a device's usage behavior in the future, usually based on the past history of usage patterns, and decide to change power states of the device accordingly. The chief parameter of interest here is the idleness threshold, i.e., the time period for a device to transition from an active state to a sleep state. Work on prediction based dynamic power management can be categorized into two groups: *adaptive* and *non-adaptive*. Non-adaptive strategies set the idleness thresholds for the algorithm once and for all and do not alter them based on observed input patterns. Adaptive strategies, on the other hand, use the history of idle periods to guide the decisions of the algorithm for future idle periods. There have been a number of adaptive strategies proposed in the literature [15, 19, 7, 11, 44]. Stochastic approaches make probabilistic assumptions (based on observations) about usage patterns and exploit the nature of the probability distribution to formulate an optimization problem, the solution to which drives the DPM strategy. Examples are in [7, 33]. Until very recently predictive schemes have been mostly based on devices with two power saving states (e.g., standby and sleep). In case of multiple states, the predictive schemes can be extended to use a sequence of idleness thresholds to determine when to transition to the next power state. By comparison, multi-state systems are naturally modeled in most stochastic optimum control approaches [6, 42, 7, 11, 33, 35]. Examples of session clustering and prediction strategies are in [24], on-line strategies are in [36], and adaptive learning based strategies are in [12]. Lu *et. al.* in [23] provide a quantitative comparison of various power management strategies. Most adaptive dynamic power management strategies [15, 44, 36, 19, 11, 24] use a sequence of past idle period lengths to predict the length of the next idle period. These strategies typically describe their prediction for the next idle period with a single value. Given this prediction, they transition to the power state that is optimal for this specific idle period length. In case the prediction is wrong, they transition to the lowest power state if the idle period extends beyond a fixed threshold value. For the sake of comparison with other approaches, we shall call these predictive DPM schemes *Single-Value Prediction*

schemes (SVP). Among SVPs, of particular interest is [12] that addresses multiple idle state systems using a prediction scheme, based on adaptive learning trees, that improves the hit ratio of the predicted interval significantly.

1.3 The System Model

Consider a single peripheral device whose power state is managed by the operating system. The device can be in one of the n power states denoted by $\{s_1, ..., s_n\}$. The power consumption for state i is denoted by α_i. Without loss of generality, we assume that the states are ordered so that $\alpha_i > \alpha_j$ for $i < j$. Thus, state s_1 is the *ready* state which is the highest power consumption state. (As an example, the ACPI [16] standard specifies the different devices classes and their recommended power states in an Intel PC platform.) In addition to the states, we are also given (typically from device manufacturer's specification) the transition power p_{ij}, and transition times t_{ij}, to move from state s_i to s_j. Typically, the power needed and time spent to go from a higher (power consumption) state to a lower state is negligible, while the converse is not true. In this model, predictive schemes often consider the transition power and transition time numbers as deterministic [9, 11, 12, 15, 39, 18, 36], whereas in stochastic approaches these numbers are used as parameters to the probability distributions assumed. In some cases, these numbers are experimentally determined [24, 23]. Another characteristic of predictive schemes is that they generally transition to the ready state when powering up and not to an intermediate (higher powered) state. Schemes using predictive wake-up [23, 12] are a notable exception and beyond the scope of this paper. However, stochastic strategies often have probabilistic predictive wakeup built into DPM algorithm. As a result, when discussing deterministic DPM we only need the time and total energy consumed (β_i) in transitioning up from each state i to the ready state.

We note that, in cases where the time and energy used in transitioning to lower power consumption states is non-negligible, these can also be incorporated in the model by folding them into the corresponding power-up parameters. This can be done as long as the time and energy used in transitioning down is additive. That is, we require that for $i < j < k$, the cost to go from i to j and then from j to k is the same as the cost of going from i directly down to k. Recently, DPM analysis has been extended to the case where additivity assumptions do not hold [2].

The input to the PM is a sequence of requests for service that arrive over time. With each request, the PM notes the time of its arrival and the length of time it will take to satisfy the request. If the device is busy when a new request arrives, it enters a queue and is served on a first-come-first-serve basis. In this case, there is no idle period and the device remains active through the time that the request is finished. Thus, the number of idle periods is less than the number of requests serviced. Whenever a request terminates and there are no outstanding requests waiting in the system, an idle period begins. In these situations, the PM determines the power consumption states the device should transition and at what times.

If the device is not busy when a new request arrives, it will immediately transition to the ready state to serve the new request if it is not already there. In the case where the device is not already in the ready state, the request can not be serviced immediately, but will have to incur some latency in waiting for the transition to complete. This delay will cause future idle periods to be shorter. In fact, if a request is delayed, some idle periods may disappear. Thus, the behavior of the algorithm affects future inputs (idle period lengths) given to the algorithm. Similarly, note that without performance constraints, delaying the servicing of a request will tend to lower the power usage. Consider the extreme case where the power manager remains in the deepest sleep state while it waits for all the requests to arrive and then processes all of them consecutively. This extreme case is not allowed to happen in our model since we require that the strategy transition to the ready state as soon as any request appears. However, it illustrates the natural trade-off which occurs between power consumption and latency. See [36] for a more extensive discussion of this trade-off.

1.4 Formal Methods for DPM

A common method for prediction of the next idle period is to use some form of regression equation over the previous idle periods, and/or use of interpolation or learning-based techniques. In contrast to these *ad hoc* techniques, the stochastic DPM literature tends to be more formal in the sense that assumptions are made about the characteristics of the probability distribution of idle periods, device response times etc. These are then used to formulate the optimization problems. Much of the stochastic DPM strategy literature uses Markov models, based on assumptions about how and when requests can arrive (whether at certain time points or at any time). For example, discrete-time and continuous-time Markov chains have been used.

Our focus on formal methods is from the point of view of developing DPM strategies that attempt to ensure bounds on the efficiency of achievable power reduction and power-latency tradeoffs without the need for time consuming simulation techniques. We seek methods that can determine these bounds either in the deterministic or probabilistic sense.

The remainder of this tutorial paper is organized as follows. Section 2 focuses mostly on predictive schemes. Due to space limitations, this presentation complements survey in [6] by focusing on the more recent work in the area. We introduce the basic concepts of on-line algorithms and competitive analysis in the context of DPM. Section 3 considers the stochastic approaches to DPM. Section 4 describes the most recent approaches based on probabilistic model checking. Finally, Section 5 summarizes the tutorial.

2. DPM AS AN ON-LINE PROBLEM

Dynamic power management is an inherently *online* problem, in that the power manager must make decisions about the expenditure of resources before all the input to the system is available [1]. The input here is the length of an upcoming idle period and the decision to be made is whether to transition to a lower power dissipation state while the system is idle. A short idleness threshold will lead to higher power-up costs, whereas a large threshold would lead to sub-optimal power usage. Analytical solutions to such online problems are often best characterized in terms of a *competitive ratio* [32] that compares the cost of an online algorithm to the optimal offline solution which knows the input in advance (and thus chooses the best assignment of power states). Earliest work on competitive analysis of dynamic power management strategies presents bounds on the quality of various DPM solutions [36, 19, 20].

2.1 Competitve Analysis of Deterministic DPM

An algorithm is $c-$*competitive* if, for any input, the cost of the online algorithm is bounded by c times the cost of the optimal offline algorithm for that input. The *competitive*

ratio of an algorithm is the infimum over all c such that the algorithm is c-competitive. It has been known for some time that 2 is the optimal competitive ratio that can be achieved for any two-state system by a deterministic algorithm [32]. We will sketch the idea behind this result in order to illustrate how competitive analysis works. Since the system has only two states, we call these the active and the sleep state. Let β be the energy cost to transition from the sleep to the active state. Let α be the power dissipation rate in the active state. Without loss of generality, we assume that the power dissipation in the sleep state is zero.

The optimal offline algorithm is assumed to know the length T of the idle period in advance. Thus, it chooses the best state for this idle period length and stays in that state for the duration of the idle period. Staying in the active state costs αT. Transitioning immediately to the sleep state costs β because the algorithm must transition back to the active state at the end of the idle period. This means that the cost for the optimal offline algorithm for an idle period of length T is $\min\{\alpha T, \beta\}$.

Since the online algorithm does not know the length of the idle period in advance, it selects a threshold τ and stays in the active state for time τ, after which it transitions to the sleep state if the system is still idle. If the idle period length T is less than τ, its cost is αT. If the idle period is longer than τ, its cost is $\beta + \alpha\tau$. The online algorithm seeks to minimize the ratio of its cost to the cost of the optimal offline algorithm for all T. It can be shown that if $\tau = \beta/\alpha$, this ratio is never more than two. The worst case for the online algorithm is if the idle period ends immediately after it transitions to the sleep state. This puts a tight bound of 2 for the competitive ratio of any deterministic algorithm.

For multi-state systems, the situation is a bit more complex in that the optimal competitive ratio will, in general, depend on the parameters of the system (e.g. the number of states, power dissipation rates, start-up costs, etc.). In [18] a generalization of the 2-competitive algorithm for two-state systems is given for multi-state systems that also achieves a competitive ratio of 2. In general, this bound is not tight because it may be possible to attain a better competitive ratio for specific systems. The bound holds under the assumption that the cost to power-down is negligible or that the state transition costs are additive. The algorithm is non-adaptive since it does not use any information about the arrival sequence of jobs to the device.

More recently, [2] have developed competitive algorithms for multi-state systems that work for arbitrary transition costs on the states. The authors give an online algorithm that obtains a competitive ratio of 8. This can be improved to 5.828 under the very reasonable assumption that $p_{ij} > p_{il}$ for any $i < l < j$. They also develop a *meta-algorithm* (i.e., a DPM algorithm generator) that takes as input the parameters of a system and produces a DPM strategy (sequence of states and threshold times). The strategy they produce is guaranteed to achieve a competitive ratio that is within an arbitrary ϵ of the best possible competitive ratio for that system. The running time of the meta-algorithm is polynomial in the number of states and $1/\epsilon$. An interesting open question is if similar techniques can be used when the idle period length is generated by a known probability distribution.

Another direction that has recently been undertaken is to combine DPM strategies with Dynamic Voltage Scaling for devices that have both the ability to run at varying speeds and the ability to shut down when idle [17]. Combining these two problems of DSS and DPM, introduces challenges which do not appear in either of the original problems. In DPM, the lengths of the idle intervals are given as part of the input whereas in the combined problem they are created by the scheduler which decides when and how fast to perform the tasks. In DVS, it is always in the best interest of the scheduler to run jobs as slowly as possible within the constraints of the arrival times and deadlines due to the convexity of the power function. By contrast in the combined problem, it may be beneficial to speed up a task in order to create an idle period in which the system can sleep. An offline algorithm is described that is within a factor of three of the optimal algorithm as well as an online algorithm with a constant competitive ratio. Some of the same issues are dealt with in [22] in which process schedulers have some latitude in scheduling the execution of tasks so as to maximize the benefit of dynamic power scheduling.

2.2 Probabilistic Analysis

As discussed above, competitive analysis often gives overly pessimistic bounds for the behavior of algorithms. This is because competitive analysis is a worst-case analysis. In many applications there is structure in the input sequence that can be utilized to fine tune online strategies and improve their performance. Indeed, important earlier works in this area [7, 33] have relied on modeling the distribution governing inter-arrival times as an exponential distribution. In practice, such stochastic modeling seems to hold well for specific kinds of applications. However, these assumptions have led to complications in other settings due to phenomena such as the non-stationary nature of the arrival process, clustering, and the lack of independence between subsequent events. These problems have been addressed to some extent in [24, 12].

In [18], we introduced an approach that models the upcoming input sequence by a probability distribution that is *learnt* based on historical data. One of the strengths of this method is that it makes no assumptions about the form of this distribution. Once the distribution is learnt, we can automatically generate a probability-based DPM strategy that minimizes the *expected* power dissipation given that the input is generated according to that distribution based on the notion of a *probabilistic competitive ratio* [39].

Optimizing Power Based on a Probability Distribution

Let us suppose that the length of the idle interval is generated by a fixed, known distribution whose density function is π. Let us consider systems with two states. As before, let β be the start-up energy of the sleep state and α the power dissipation of the active state. Suppose that the online algorithm uses τ as the threshold at which time it will transition from the active state to the sleep state if the system is still idle. In this case, the *expected* energy cost for the algorithm for a single idle period is given as:

$$\int_0^\tau \pi(t)(\alpha t)dt + \int_\tau^\infty \pi(t)[\alpha\tau + \beta]dt.$$

The best online algorithm will select a value for τ which minimizes this expression. On the other hand, the offline optimal algorithm which knows the actual length of an upcoming idle period will have an expected cost of:

$$\int_0^{\beta/\alpha} \pi(t)(\alpha t)dt + \int_{\beta/\alpha}^\infty \pi(t)\beta dt.$$

It has been shown that for the 2-state case, the online algo-

rithm can pick its threshold τ so that the ratio of its expected cost to the expected cost of the optimal algorithm is at most $e/(e-1) = 1.58$ [19]. A generalization of this algorithm to the multi-state case is given in [39]. The generalized algorithm is called the Probabilistic Lower Envelope Algorithm (or PLEA) and the authors have shown that for any distribution, the expected cost of PLEA is within a factor of $\frac{e}{e-1}$ of the expected cost for the optimal offline algorithm. Thus a knowledge of the input pattern and its use can help bridge the gap between the performance of online strategy and that of the optimal offline strategy. Results show the the worst case competitive ratio can be improved by 21%, with respect to the deterministic case [18].

Learning the Probability Distribution

The algorithm PLEA above assumes perfect knowledge of the probability distribution governing the length of the idle period. Rather than assuming such a distribution, it can be learnt based on recent history. For instance, a learning scheme in conjunction with PLEA is called the Online Probability-Based Algorithm (OPBA). The probability estimator works as follows: a window size w is chosen in advance and is used throughout the execution of the algorithm. The algorithm keeps track of the last w idle period lengths and summarizes this information in a histogram. Periodically, the histogram is used to generate a new power management strategy.

The set of all possible idle period lengths $(0, \infty)$ is partitioned into n intervals, where n is the number of bins in the histogram. Let r_i be the left endpoint of the i^{th} interval. The i^{th} bin has a counter c_i which indicates the number of idle periods among that last w idle periods whose length fell in the range $[r_i, r_{i+1})$. Instead of using the continuous probability distribution π with PLEA as described in the previous section, we use a discrete distribution, where the probability the idle period has length r_i is c_i/w. A similar approach was taken for a two state system in the context of determining virtual circuit holding time policies in IP-over-ATM Networks [20].

Efficient implementation of such an algorithm is important to ensure overall gains in power reduction. In [18], we present an implementation for finding the $m-1$ thresholds in time $O(mn)$, where m is the number of states and n is the number of bins in the histogram. Two important factors which determine the cost (in time expenditure) of implementing our method is the frequency with which the thresholds are updated and the number of bins in the histogram. Selecting the right granularity for the histogram is an important consideration since there is a tradeoff between efficiency and accuracy. The algorithm employs non-uniform bin sizes so as to have a high degree of accuracy in critical regions. The reader is referred to [18] for a description of how system parameters are used to select bin sizes.

3. STOCHASTIC APPROACHES TO DPM

We now discuss the stochastic version of the DPM problem. The problem basically requires one to devise a strategy (policy) which is *probabilistic*, in the sense that the actions to be taken by the strategy have probabilities attached to them. Unlike deterministic strategies, where a particular state of the system will lead the strategy to take a deterministic action, here, the strategy can choose between multiple actions with pre-designated probabilities.

In recent years, several approaches for designing stochastic DPM strategies have been proposed [30, 7, 6, 11, 33, 35, 34, 42]. These methodologies are based on a stochastic model of the DPM problem, which incorporates the probabilistic characteristics of request arrivals to the device, the device response time distribution, the power consumption by the device in various states and the distribution of energy consumption in changing states. From this stochastic model, an exact optimization problem is formulated, the solution to which is the required optimal stochastic DPM policy. The strategy devised must ensure that power savings are not achieved at an undue cost in performance. For example, a new request should be always served in a reasonable time. The constructed policy optimizes the *average* energy usage while minimizing *average* delay. The policies are usually validated by simulation to check for the soundness of the modeling assumptions, and the effectiveness of the strategies in practice [33, 30].

The stochastic models which have been used in the literature are discrete-time Markov chains [30, 7], continuous-time Markov chains [33, 35, 34] or their variants [42]. The approaches vary in the model of time. In the continuous-time case, *mode* switching commands can be issued at any time, and events can happen at any time. In the discrete-time case, all events and actions occur at certain discrete time points. The continuous-time assumption makes the formulation of the problem easier. In practice, such stochastic modeling seems to work well for specific kinds of applications. Generally, the stochastic matrices for these models are created manually. In [34], stochastic Petri nets are used, which allows automatic generation of the stochastic matrices and formulation of the optimization problems.

3.1 Analysis using Model Checking

Probabilistic Model Checking (PMC) offers a promising way to verify stochastic approaches to DPM as shown in [28, 29]. The idea is to construct a probabilistic model of the system under study. As in the deterministic case, this is usually a labeled transition system which defines the set of all possible states and the transitions between these states. In PMC, the model is augmented with information about the likelihood that each transition will take place. Examples of such models are discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs). The properties to be verified, are specified typically in probabilistic extensions of temporal logic. These allow specification of properties such as: "shutdown occurs with probability at most 0.01"; or "the video frame will be delivered within 5ms with probability at least 0.97." The properties can be verified with a probabilistic model checker either as graph-based analysis and solution of linear equation systems or linear optimization problems [31].

Like the conventional, non-probabilistic case, probabilistic model checking usually constitutes verifying whether or not some temporal logic formula is satisfied by a model. The two most common temporal logics for this purpose are PCTL [14, 10] and CSL [3, 5], both extensions of the logic CTL. PCTL is used to specify properties for DTMCs and MDPs and CSL is used for CTMCs. One common feature of the two logics is the probabilistic $\mathcal{P}$ operator, which allows one to reason about the probability that executions of the system satisfy some property. For example, the formula $\mathcal{P}_{\geq 1}[\Diamond\ terminate]$ states that with probability 1, the system will eventually terminate. On the other hand, the formula $\mathcal{P}_{\geq 0.95}[\neg repair\ U^{\leq 200}\ terminate]$ asserts that with probability 0.95 or greater, the system will terminate within 200 time steps and without requiring any repairs. These prop-

erties can be seen as analogues of the non-probabilistic case, where a formula would typically state that *all* executions satisfy a particular property, or that *there exists* an execution which satisfies it. CSL also provides the $\mathcal{S}$ operator to reason about steady-state (long-run) behavior. The formula $\mathcal{S}_{<0.01}[queue\_size = max]$, for example, states that in the long-run, the probability that a queue is full is strictly less than 0.01. Further properties can be analyzed by introducing the notion of *costs* (or, conversely, *rewards*). If each state of the probabilistic model is assigned a real-valued cost, one can compute properties such as the expected cost to reach a certain states, the expected accumulated cost over some time period, or the expected cost at a particular time instant. Such properties can also be expressed concisely and unambiguously in temporal logic [13, 4].

4. DPM ANALYSIS USING PRISM

PRISM [31, 28, 29] is a probabilistic model checker developed at the University of Birmingham in England. In [28, 29] PRISM was used for deriving stochastic DPM policies for disk-drives, and was shown to be a uniform framework in which DPM policies can be derived and evaluated. The basic approach is to build a probabilistic model of the DPM system from which, for a given constraint, an optimization problem is constructed. The solution to this problem is the optimum randomized power management policy satisfying this constraint.

Once an optimal power management policy has been constructed, it must be validated to ensure it performs as intended. Possible approaches are to use trace-based simulation or to actually implement the schemes in device drivers. The advantage of PMC is that it allows one to validate and analyze the policies statically leading to a wide range of useful information about the policy to be generated.

Modeling DPM in PRISM

While PMC has been applied to both DTMC of [30, 7], as well as CTMC of [33, 35, 34], we focus on the former here in view of the limited space. The approach is described through the example of the IBM TravelStar VP disk-drive [45]. The device has 5 power states, labelled *sleep*, *stby*, *idle*, *idlelp* and *active*. It is only in the state *active* that the drive can perform data read and write operations. 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. Based on the fastest possible transition performed by system, one can choose a time resolution of 1ms for the model, i.e., each discrete-time step of the DTMC will correspond to 1ms.

The system model shown in Figure 1 consists of: a Service Provider (SP), which represents the device under power management control; a Service Requester (SR), which issues requests to the device; a Service Request Queue (SRQ), which stores requests that are not serviced immediately; and the Power Manager (PM), which issues commands to the SP, based on observations of the system and a stochastic DPM policy. Each component is represented by an individual PRISM module, which we now consider in turn. Due to lack of space, we only provide examples of important components for illustration purposes.

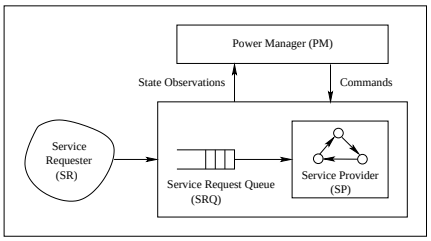

Figure 1: The System Model

Modeling the Power Manager(PM), Service Requester (SR) and Queue(SRQ)

The PM decides to which state the SP should move at each time step. To model this, each step is split 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). These steps are synchronized with other components using two synchronization actions $tick1$ and $tick2$, described in a CLOCK module (not shown here). Figure 3 shows an example PM in PRISM.

Both the SRQ and the SR will synchronize 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 [7]. 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
```

The SRQ models queue of service requests. It responds to the arrival of requests from the SR and the service of requests by the SP. The queue size will only decrease when the SR and SP are in states *idle* and *active*, respectively. Similarly, it will only increase when the SR is in state *1req* and the SP is not *active*. The PRISM code is as follows:

```
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
```

4.1 Policy Construction and Analysis

Using the PRISM language description shown in the previous section, the PRISM model checking tool can be used to construct a generic model of the power management system. From the transition matrix of this system, the linear optimization problem (whose solution is the optimal policy) can be formulated, as described in [30, 7]. This optimization problem is then passed to the MAPLE symbolic solver. Figure 2 shows policies constructed in this way for a range of constraints on the average size of the service request

constraint	optimum policy
≤ 2	remain sleeping
≤ 1.5	SP active and queue not full: goto idle SR in state 0, SP sleeping and queue full: remain sleeping with probability 0.99999953 go to active with probability 0.00000047 SR in state 1, SP idle: goto active
≤ 0.5	SP active and queue not full: goto idle SR in state 0, SP sleeping and queue full: remain sleeping with probability 0.99999418 go to active with probability 0.00000582 SR in state 1 and SP idle: goto active
≤ 0.05	SP active, SR in state 0 and queue empty: remain active with probability 0.63683933 go to idle with probability 0.36316067 SP idle: go to active SP sleeping: go to active

Figure 2: Optimum policies under varying constraints on the average queue size

queue. The first column lists the constraint; the second column summarizes the corresponding policy.

Once a policy has been constructed, its performance can be investigated using probabilistic model checking. The generic power manager PRISM module is modified to represent a specific policy. Figure 3 shows an example of this for the constraint "queue size is less than 0.05". This can be seen to correspond to the policy in the 6th row of the table in Figure 2. PRISM is then used to construct and analyze the DTMC for this policy.

```
module PM

    // policy when constraint on queue size equals 0.05
    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] sr=0 ∧ sp=0 ∧ q=0 →
                              0.63683933 : pm′=0 //active
                            + 0.36316067 : pm′=1; //idle
    [tick1] sp=1 → pm′=0; //active
    [tick1] sp=9 → pm′=0; //active
    [tick1] ¬(sp=9 ∨ sp=1 ∨ (sr=0 ∧ sp=0 ∧ q=0)) →
                                                pm′=pm;

endmodule
```

Figure 3: Example input to PRISM for a derived Policy under performance constraint = 0.05

From the analysis, we can see that the average power consumption of a policy decreases as the constraint on queue length used to construct it is relaxed (i.e. the queue size is larger). One can also validate the policy by confirming that the expected size of the queue matches the value in the constraint which was used to construct it. Finally, we see that a side-effect of this is that the average number of requests lost is also increased.

In Figure 4, we show results for a range of policies from [29]. Using the same assignments of model states to costs as discussed above, we compute and plot, for a range of values of T: "expected power consumption by time T", "expected queue size at time T", and "expected number of lost customers by time T". The first and third properties are determined by computing expected cost cumulated up until time T; the second by computing the instantaneous cost at time T. Again, we see that policies which consume less power have larger queue sizes and are more likely to lose requests. Here, though, we can get a much clearer view of how these properties change over time. We see, for example, that the expected queue size at time T initially increases and then decreases. This follows from the fact that the strategies wait for the queue to become full before switching the SP on.

Figure 5 shows the probability that a request is served

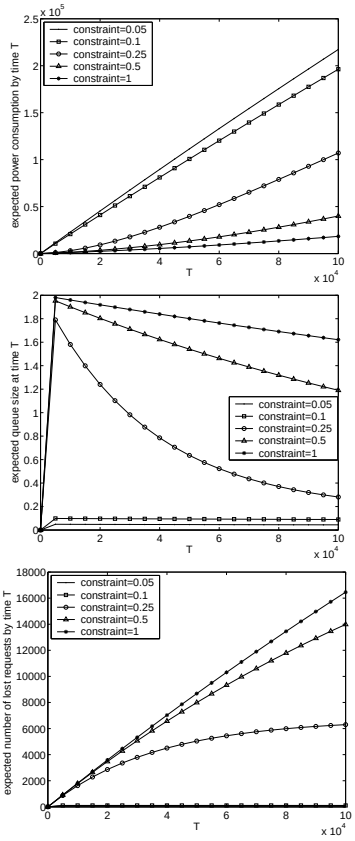

Figure 4: Power and performance by time T (ms)

by time T, given that it arrived into a certain position in the queue. Figure 6 shows the probability that N requests get lost by time T for $N = 500$ and $N = 1000$. Again this information has been computed for a range of policies and for a range of values of T. These properties are computed by adding additional state variables to the PRISM model. For those in Figure 6, for example, we add a variable which is initially zero and is increased each time a customer is lost (up to a maximum on N). We then calculate the probability of reaching any state where this variable's value is equal to N.

The graphs show that the probability of requests being lost within a certain time bound increases more quickly 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 which cannot service requests, e.g., *sleep* and *standby*.

Probabilistic model checking has also been applied [28] to the stochastic optimum control approach of [33, 35, 34], which is based on CTMCs rather than DTMCs. Since the model is a CTMC, components change state according to exponentially distributed delays and the PM acts when such a state transition occurs. The construction of optimum policies from the PRISM model follows the approach of [33, 35, 34] but is essentially the same overall process. For analysis of policies, one can consider similar properties to the DTMC case. The main differences are that the logic CSL is used as opposed to the logic PCTL, and that the time bound T used in the properties is now a real-value as opposed to a number of discrete steps. In addition, in this case, using

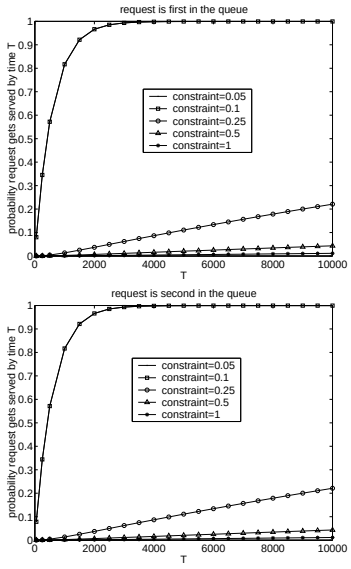

Figure 5: Probability that a request is served by time T (ms)

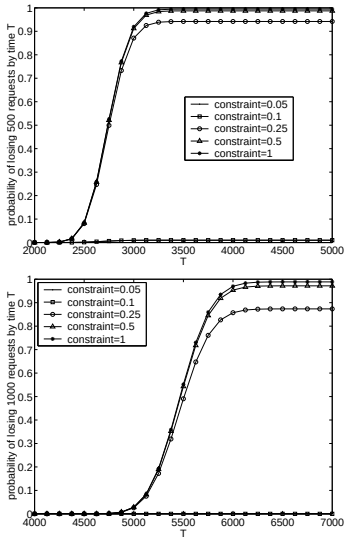

Figure 6: Probability that N requests gets lost by time T (ms)

the approach of [21] one can also analyze the policies for alternative inter-arrival distributions, to give a more realistic model of the arrival of service requests. For example, Figure 7 shows the performance (average power consumption, average queue size and average number of lost requests) for optimum policies under five different inter-arrival distributions. All the chosen distributions have the same mean and it can be seen that, with the exception of the Pareto distribution, the long-run performance and costs are reasonably close to those of the exponential arrival process. For the Pareto distribution, the average queue size is generally much smaller. This is due to the Pareto distribution's *heavy tail*: in the long run, many requests will not arrive for a very long time, in which case the service provider (SP) will serve all pending requests, leaving the queue empty.

5. SUMMARY

In this tutorial, we focused on techniques for power management that rely on formal techniques for evaluation of the effectiveness of DPM algorithms. For deterministic models of the system, competitive analysis along with learning techniques provide a reaonsable framework for their analyses. Stochastic optimization approaches to DPM can be analyzed using advances in probabilistic model checking techniques.

We showed (from [28, 29]) how probabilistic model checking allows generation of a wide range of performance measures for the analysis of DPM policies. Statistics such as power consumption, service queue length and the number of requests lost can be computed both in the average case and for particular time instances over a given range. Furthermore, the policies' behavior can be examined under alternative service request inter-arrival distributions such as Erlang and Pareto. In addition to the exhaustic analysis (including corner-case scenarios), probabilistic model checking presents an attractive unified framework for automated construction, validation and analysis of DPM policies.

Due to lack of space we have not been able to cover some new efforts in power management in the context of power aware ad-hoc network protocols. Notable among these are the use of an economics-based model for networking protocols in [41] and power aware source routing in [25]. We also did not discuss DPM models that consider the battery model which is not considered in any of the approaches discussed above. Rong and Pedram in [38] provide a stochastic model that takes into account the current discharge rates by the batteries in formulating stochastic DPM strategies.

6. REFERENCES

[1] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[2] J. Augustine and C. Swamy. Dynamic power management with non-additive state transitions. Submitted for publication. Available as Technical Report, Information and Computer Science, UC Irvine.

[3] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Proc. Conference on Computer-Aided Verification*, July 1996.

[4] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *Proc. ICALP'00*, 2000.

[5] C. Baier, J.-P. Katoen, and H. Hermanns. Approximative symbolic model checking of continuous-time Markov chains. In *Proc. CONCUR'99*, Eindhoven, August 1999.

[6] L. Benini, A. Bogliolo, and G. D. Micheli. 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.

[7] L. Benini, A. Bogliolo, G. Paleologo, and G. D. Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, 1999.

[8] L. Benini, G. De Micheli, and E. Macii. Designing Low-power Circuits: Practical Recipes. *IEEE Circuits and Systems Magazine*, 1(1):6–25, Mar. 2001.

[9] L. Benini and G. D. Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Publications, 1998.

[10] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. FSTTCS*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[11] E. Y. Chung, L. Benini, A. Bogliolo, and G. D. Micheli. Dynamic Power Management for Non-Stationary Service Requests. In *Proceedings of the Design Automation and Test Europe*, 1999.

[12] E.-Y. Chung, L. Benini, and G. D. Micheli. Dynamic Power Management Using Adaptive Learning Trees. In *Proceedings of ICCAD*, 1999.

[13] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

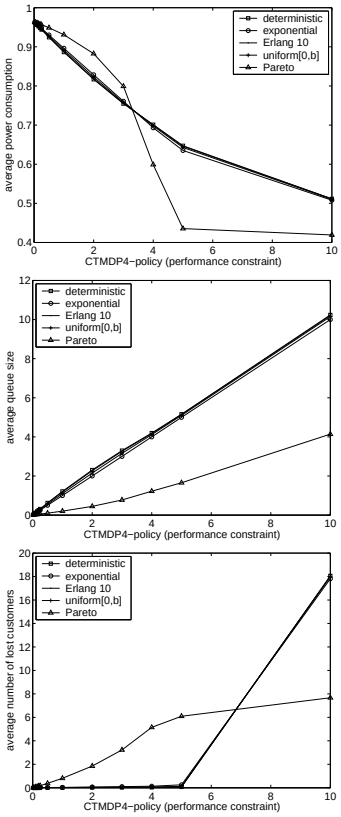

Figure 7: Analysis of the CTMC case for a variety of inter-arrival distributions

[14] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.

[15] C.-H. Hwang, C. Allen, and H. Wu. 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.

[16] Intel and Microsoft and Toshiba. Advanced Configuration and Power Interface Specification. Website, December 1996.

[17] S. Irani, S. Shukla, and R.Gupta. Algorithms for power savings. In *Proceedings of the 14th Symposium on Discrete Algorithms*, pages 37–46, 2003.

[18] S. Irani, S. Shukla, and R.Gupta. Dynamic power management of devices with multiple power saving states. *ACM Transactions on Embedded Systems*, Accepted for publication, 2003.

[19] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Randomized competitive algorithms for non-uniform problems. In *First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990.

[20] S. Keshav, C. Lund, S. Phillips, N. Reaingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in ip-over-atm networks. *IEEE Journal on Selected Areas in Communications*, 13:1371–1382, 1995.

[21] M. Kwiatkowska, G. Norman, and A. Pacheco. Model checking expected time and expected reward formulae with random time bounds. In *Proc. 2nd Euro-Japanese Workshop on Stochastic Risk Modelling for Finance, Insurance, Production and Reliability*, 2002.

[22] Y. Lu, L. Benini, and G. DeMicheli. Request-aware power reduction. In *Proceedings of the International Symposium on System Synthesis*, pages 18–23, 2000.

[23] Y. Lu, E. Chung, t. Simunic, L. Benini, and G. DeMicheli. Quantitative Comparison of Power Management Algorithms. In *Proceedings of the Design and Automation and Test in Europe*, 2000.

[24] Y. Lu and G. DeMicheli. Adaptive Hard Disk Power Management on Personal Computers. In *Proceedings of the Great Lakes Symposium on VLSI*, 1999.

[25] M. Maleki, K. Dantu, and M. Pedram. Power-aware source routing protocol for mobile ad hoc networks. In *Proceedings of ISLPED*, 2002.

[26] Microsoft. OnNow Power Management Architecture for Applications. Website, August 1997.

[27] R. Min, M. Bhardwaj, S.-H. Cho, N. Ickes, E. Shih, A. Sinha, A. Wang, and A. Chandrakasan. Energy-Centric Enabling Technologies for Wireless Sensor Networks. *IEEE Wireless Communications*, August 2002.

[28] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Formal analysis and validation of continuous time Markov chain based system level power management strategies. In *Proc. 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02)*, pages 45–50, 2002.

[29] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. In *Proc. 3rd Workshop on Automated Verification of Critical Systems (AVoCS'03)*, pages 202–215, April 2003.

[30] G. A. Paleologo, L. Benini, A. Bogliolo, and G. D. Micheli. Policy Optimization for Dynamic Power Management. In *Proceedings of Design Automation Conference*, 1998.

[31] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[32] S. Phillips and J. Westbrook. *chapter 10 of Algorithms and Theory of Computation Handbook*, chapter On-line algorithms: Competitive analysis and beyond. CRC Press, Boca Raton, 1999.

[33] Q. Qiu and M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In *Proceedings of Design Automation Conference*, pages 555–561, June 1999.

[34] 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, June 2000.

[35] 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*, 1999.

[36] D. Ramanathan, S. Irani, , and R. K. Gupta. Latency Effects of System Level Power Management Algorithms. In *Proc. ICCAD*, 2000.

[37] Z. Ren and B. Krogh. Sentry-based Power Management in Wireless Sensor Networks. In *Proceedings of IPSN*, 2003.

[38] P. Rong and M. Pedram. Extending the lifetime of a network of battery-powered mobile devices by remote processing: a Markovian decision-based approach. In *Proceedings of DAC*, 2003.

[39] S. Irani and S. Shukla and R. Gupta. 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.

[40] 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)*. IEEE Press, November 2001.

[41] L. Shang, R. Dick, and N. K. Jha. An economics-based power-aware protocol for computation distribution in mobile ad-hoc networks. In *Proceedings of 14th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2002.

[42] T. Simunic, L. Benini, and G. D. Micheli. Event Driven Power Management of Portable Systems. In *In the Proceedings of International Symposium on System Synthesis*, pages 18–23, 1999.

[43] M. Srivastava. *Chapter 11 of Power Aware Design Methodologies*, chapter Power-Aware Communication Systems. Kluwer Academic Press, 2002.

[44] M. B. Srivastava, A. P. Chandrakasan, and R. W. Broderson. Predictive Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation. *IEEE Trans. on VLSI Systems*, 4(1):42–54, march 1996.

[45] *Technical specifications of hard drive IBM Travelstar VP 2.5inch, available at.* http://www.storage.ibm.com/storage/oem/data/travvp.htm, 1996.