Dynamic Power Management of Complex Systems Using Generalized Stochastic Petri Nets*

Qinru Qiu, Qing Wu and Massoud Pedram

Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089
EMAIL: {qinru, qingwu, massoud}@zugros.usc.edu

Abstract

In this paper, we introduce a new technique for modeling and solving the dynamic power management (DPM) problem for systems with complex behavioral characteristics such as concurrency, synchronization, mutual exclusion and conflict. We model a power-managed distributed computing system as a controllable Generalized Stochastic Petri Net (GSPN) with cost. The obtained GSPN model is automatically converted to an equivalent continuous-time Markov decision process. Given the delay constraints, the optimal power management policy for system components as well as the optimal dispatch policy for requests are calculated by solving a linear programming problem based on the Markov decision process. Experimental results show that the proposed technique can achieve more than 20% power saving compared to other existing DPM techniques.

I. Introduction

With the rapid progress in the semiconductor technology, the chip density and operation frequency have increased, making the power consumption in battery-operated portable devices a major concern. High power consumption reduces the battery service life. The goal of low-power design [1]-[4] for battery-powered devices is thus to extend the battery service life while meeting performance requirements. Dynamic power management (DPM) [5] – which refers to selective shut-off or slow-down of system components that are idle or underutilized – has proven to be a particularly effective technique for reducing power dissipation in such systems. Incorporating a dynamic power management scheme in the design of an already-complex system is a difficult process that may require many design iterations and careful debugging and validation.

A simple and widely-used technique is the "time-out" policy [5], which turns on the component when it is to be used and turns off the component when it has not been used for some pre-specified length of time. Srivastava et al. [6] proposed a predictive power management strategy, which uses a regression equation based on the previous "on" and "off" times of the component to estimate the next "turn-on" time. In [7], Hwang and Wu have introduced a more complex predictive shut-down strategy that has a better performance. However, these heuristic techniques cannot handle components with more than two ("ON" and "OFF") power modes; they cannot handle complex system behaviors; and they cannot guarantee optimality.

As proposed in [8], a power-managed system can be modeled as a **discrete-time** Markov decision process by combining the stochastic models of each component. Once the model and its parameters are determined, an optimal power management policy for achieving the best power-delay trade-off in the system can be generated. In [9], the authors improved [8] by modeling the power managed system using a **continuous-time** Markov decision process (CTMDP). Further research results can be found in [10]-[13]. The DPM approaches based on Markov decision processes offer significant improvements over heuristic power management policies in terms of the theoretical framework and ability to apply strong mathematical optimization techniques [14]-[20]. However, previous works based on Markov decision processes only describe modeling and policy optimization techniques for a simple power managed system. Such a system contains one Service Provider (SP) that provides services (e.g. computing, file access, etc.), one Service Queue (SQ) that buffers the service requests for the SP, and one Service Requestor (SR) that generates the requests for SP. It is relatively easy to construct the stochastic models of the individual components because their behavior is rather simple. However a significant effort is required to construct the joint model of SP and SQ mostly because of the required synchronization between state transitions of SP and SQ. In this paper, we target more complex power-managed systems as shown in Figure 1. The example depicts a typical multi-server (distributed computing) system. Note that we are only interested in the system behavior as related to the power management. The system contains multiple SPs with their own Local SQs (LSQ). There is a SR that generates the tasks (requests) that need to be serviced. The Request Dispatcher (RD) makes decisions about which SP should service which request. Different SPs may have different power/performance parameters. In real applications, the RD and LSQs can be part of the operating system, while SPs can be multiple processors in a multi-processor computing system or number of networked computers of a distributed computing system.

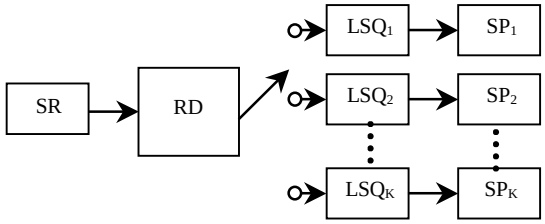


Figure 1 A multi-server/distributed-computing system.

The complexity of the modeling problem for the above system is high not only because of the increased number of components, but also because of the complex system behaviors that are present. For example, we need to consider the synchronization of LSQs and SPs, the synchronization of the SR and LSQs, the dispatch behavior of the RD, and so on. In this situation when complex system behaviors is a major part of the system model, the modeling techniques in [8]-[10] become powerless because they only offer stochastic models for individual components and require that global system behaviors be captured manually. Obviously, we need new DPM modeling techniques for large systems with complex behaviors.

In this work, we first present a methodology based on **Generalized Stochastic Petri Nets** (GSPN) [22] to model complex systems such as the one shown in Figure 1 and then

*This work was supported in part by SRC under contract No. 98-DJ-606 and NSF under contract No. MIP-9628999.

calculate the optimal power management policy (which includes the optimal dispatch policy for the RD and the optimal mode-switching policies for the SPs) under given performance (delay) constraints.

The paper is organized as follows. Sections II introduces the system modeling based on GSPN. Sections III and IV present experimental results and conclusions. Please refer to [22][23] for background on the GSPN and [17] for background on the continuous-time Markov decision process.

II. System modeling using GSPN

First, we give the definition of a *GSPN with cost* and a *controllable GSPN with cost.*

Definition: A GSPN with cost is a GSPN model with the addition of two types of cost: *impulse cost* associate with marking transitions and *rate cost* associated with places. Impulse cost occurs when the GSPN makes a transition from one marking to another. Rate cost is the cost per unit time when the GSPN stays in a certain marking.

Definition: A controllable GSPN with cost is a GSPN where all or part of the case probabilities of activities can be controlled by outside commands.

The objective of presenting a controllable GSPN with cost is to model the systems that can be controlled by outside commands to achieve different costs. In this work, our power-managed system is modeled using a controllable GSPN with cost. In this section, we will focus on how to construct such a controllable GSPN model with cost for a *unit server system* (*USS*), a *requestor generating system* (*RGS*) and the *multiple server system* (*MSS*). After constructing the GSPN model for the MSS, we reduce it into SPN and generate the corresponding CTMDP.

2.1 Unit Server System

First, we give the definition of *tangible place* and *vanishing place.*

Definition 3.1 A place is called a vanishing place if it is the only input place of an instantaneous activity; otherwise the place is called a tangible place.

A *USS* contains a SP and a SQ. The information needed for SP is its power consumption in each power state, its service speed in each power state, and the time and the energy needed to switch from one state to another state. The information needed for the SQ is its capacity. We use two cost metrics to describe the system, one is *c_pow* which denotes the power consumption of the system, the other is *c_delay* which denotes the performance of the system.

The GSPN model of a USS contains the following elements:

1. A set of vanishing places $\{P_{decision}(s)\}$. Each $P_{decision}$ is associated with a SP power state *s* and represents a short time period during which the SP is receiving commands from the power manager.
2. A set of tangible places $\{P_{SP\_status}\}$, which can be divided into three subsets: $\{P_{s2s'}\}$, $\{P_{work}(s)\}$ and $\{P_{idle}(s)\}$. The places in $\{P_{s2s'}\}$ represent the status of SP while it is switching from state *s* to *s′*. The places in $\{P_{work}(s)\}$ represent the status of SP while it is in power state *s* and servicing certain request. The places in $\{P_{idle}(s)\}$ represent the status of SP while it is in power state *s* and not doing any operation. For example, the SP can be idle when it is in active state while the SQ is empty; it can also be idle when it is in sleeping state while no new command is issued by PM. Whether or not a token is in place $P_x \in \{P_{SP\_status}\}$ indicates whether or not the SP is in status *x*. At any time, the SP should be in exactly one of these statuses. Therefore, the sum of the number of the tokens in the set of tangible places $\{P_{SP\_status}\}$ is 1. The rate cost of c_pow for place $P_x \in \{P_{SP\_states}\}$ is the power consumption of SP. The rate cost of c_delay for P_x is 0.
3. A tangible place P_{SQ}. The number of tokens in this place represents the number of waiting requests in the SQ. The rate cost of c_power of this place is 0, the rate cost of c_delay of this place equals the number of tokens in the place.
4. A vanishing place $P_{changing}$. It indicate the very short time interval when the state of the system is changed, therefore, it is the time instance at which PM issues a new command. If SP is idle at that instance, it will take the command, otherwise, it will ignore it. The procedure is modeled by using GSPN as follows: if the token of SP is in $P_{SP\_idle}(s)$, it will transfer to $P_{decision}(s)$, otherwise, no action is taken. In both cases, the token in $P_{changing}$ is vanished.
5. A set of instantaneous activities $\{T_{decision}(s)\}$. Each $T_{decision}(s)$ is associated with a SP power state *s*. The input place of $T_{decision}(s)$ is $P_{decision}(s)$. A $T_{decision}(s)$ has several cases, which are mutually exclusive. The case probability is policy- and marking-dependent. Given the policy and system marking, the probability for the case *i* of $T_{decision}(s)$ is the probability that action *i* is chosen when SP is in state *s*. The output place of case *i* of $T_{decision}(s)$ is $P_{s2s'}$, where *s′* is the destination state of action *i*. In order to make the resulting Markov decision process solvable, we set the constraint that when SQ is full, the probability for case *i* in $T_{decision}(s)$ is zero if the destination state of action *i* provides slower service or wakes up more slowly than state *s*.
6. A set of timed activities $\{T_{s2s'}\}$. The input place of $T_{s2s'}$ is $P_{s2s'}$. The output place of $T_{s2s'}$ is $P_{decision}(s')$. The time of the activity is the time that SP needs to switch from state *s* to state *s′*. The impulse cost of *c_pow* for $T_{s2s'}$ is the energy needed to switch from state *s* to state *s′*.
7. A set of timed activities $\{T_{process}(s)\}$. The input places of $T_{process}(s)$ are $P_{work}(s)$. The output place of $T_{process}(s)$ is $P_{decision}(s)$. The rate of the activity is the service rate of SP when it is in state *s*.
8. A set of instantanouse activities $\{T_{re\text{-}decision}(s)\}$. The input places of $T_{re\text{-}decision}(s)$ are $P_{changing}$ and $P_{idle}(s)$. Its output place is $P_{decision}(s)$.

An example GSPN model of a simple USS is given in Example 2.1.

Example 2.1 Assume that the SP in the processor has two power states: {*active, sleeping*}. In the *active* state, the SP provides services with an average service time of 5ms. The average time to switch from the *active* state to *sleeping* state is 0.66ms, the average time to switch from the *sleeping* state to *active* state is 6ms. The power consumption of SP is 2.3w when it is in the *active* state and 0.1w when it is in the *sleeping* state. The energy needed to switch from the *active* state to *sleeping* state is 2mJ, the energy needed to switch from the *sleeping* state to *active* state is 30mJ. Assume that the maximum length of SQ is 3. Figure 2 shows the GSPN model of the single processor system. The input gate $G_{capacity}$ sets the SQ capacity constraint.

The place P_{a2s} denotes the SP status when it is switching from the *active* state to *sleeping* state, the place P_{s2a} denotes the SP status when it is switching from the *sleeping* state to *active* state. Therefore, $\{P_{si2sj}\}=\{P_{a2s}, P_{s2a}\}$. The place $P_{idle}(a)/P_{idle}(s)$ denotes the SP status when it is idle and the power state is *active*/*sleeping*. The place $P_{work}(a)$ denotes the SP status when it is working and the power state is *active*. The SP will be in exactly one such status at any time. From the topology of the GSPN we know that the sum of tokens in places P_{a2s}, $P_{idle}(a)$, P_{work}, $P_{idle}(s)$, P_{s2a} is 1 at any time. The *c_pow* rate cost for P_{a2s}, $P_{idle}(a)$ and $P_{work}(a)$ are 2.3w. The c_pow rate cost for P_{s2a}, $P_{idle}(s)$ are 0.1w.

The number of tokens in P_{SQ} denotes the number of waiting requests in the SQ. The initial marking of $P_{idle}(a)$ is 1 while the initial marking of the other places is 0, which indicates that the initial state of the SP is idle and the initial state of SQ is empty.

The places $P_{decision}(a)$, $P_{decision}(s)$ are vanishing places. They indicate the very short period of time when the SP is taking command from PM and is in *active* or *sleeping* state. The place $P_{changing}$ is also a vanishing place. It is an auxiliary place that indicates that the state the system is changing so that it is time for the SP to receive the power management command if it is currently idle.

T_{a2s}, T_{s2a} are timed activities. They indicate the time needed to switch from the *active* state to *sleeping* state and the time needed to switch from the *sleeping* state to *active* state. $T_{processing}$ is also a timed activity, which indicates the time needed to process one

request. T_{input} denotes the time needed to generate the next request. It actually belongs to the GSPN model of the request generation system. $T_{decision}(a)$ and $T_{decision}(s)$ are instantaneous activities. They represent the process of randomized action issued by the power manager (PM). The two cases in $T_{decision}(a)$ or $T_{decision}(s)$ are mutually exclusive. The case probability equals the *action probability*, which is marking and policy dependent. If the policy is unknown, the GSPN is a controllable GSPN.

When the SP is idle and active (a token is in place $P_{idle}(a)$) and SQ is not empty, the instantaneous activity T_{start} is completed which indicates that the SP enters the busy state. When the SP is sleeping (a token is in place $P_{idle}(s)$) and the state of SQ is changing (a token is in place $P_{changing}$), the instantaneous activity $T_{re\text{-}decision}$ is completed which indicates that the SP returns to the action taking stage. If the SP is not sleeping and the state of SQ is changing, the instantaneous activity T_{vanish} is completed which indicates that the change is ignored.

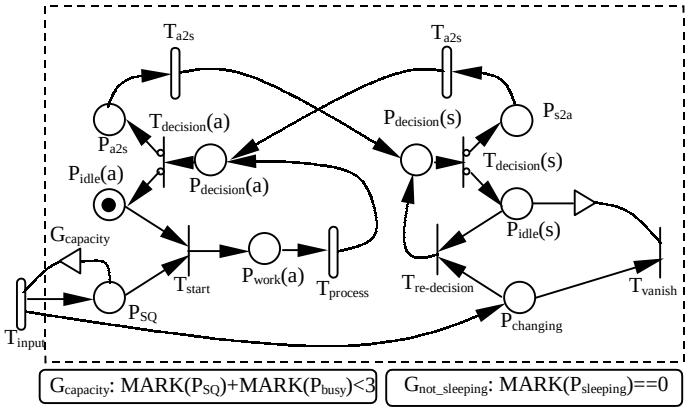


Figure 2 Example GSPN model of a USS.

2.2 Request Generation System

The RGS can generate various types of requests. The generation time of different types of request may be different. Some types of requests can be serviced by several different USS; whereas some other types of requests can be serviced by only a certain USS. There may exist correlations among the generation of different types of requests. If the SQ is full, the RGS will stop generating request. It will resume request generation when there is vacancy in the SQ.

The GSPN model of a USS contains the following elements:

1. A set of tangible places $\{P_{gen}(i)\}$. A place $P_{gen}(i)$ denotes the status of RGS when it is generating a request of type i.
2. A set of vanishing places $\{P_{switch}(i)\}$. A place $P_{switch}(i)$ denotes the very short period of time that one request has been issued but the next request generation has not started.
3. A set of timed activities $\{T_{gen}(i)\}$. An activity $T_{gen}(i)$ represents the time needed by RGS to generate request i. Its input place is $P_{gen}(i)$ whereas its output place is $P_{switch}(i)$.
4. A set of instantaneous activities $\{T_{switch}(i)\}$. An activity $T_{switch}(i)$ has several cases. The probability of case j denotes the probability that the next request will be of type j given the condition that current request is type i. Its input place is $P_{switch}(i)$, the output place of case j is $P_{gen}(j)$.

An example GSPN model of a simple RGS is given in Example 2.2.

Example 2.2 Assume that there are three types of requests, one is type A which can only be serviced by USS A, one is type B which can only be service by USS B, the other is type AB which can be serviced by both USS A and USS B. The correlations among these requests are given by matrix **P**, For example, from the matrix we know that the probability that a type AB request was issued after a type A request is 0.6. Figure 3 shows the GSPN model of this RSG. In this figure, the case probability of activities $T_{switch}(A)$, $T_{switch}(B)$ and $T_{switch}(AB)$ takes value from matrix **P**. The input gate G_{cap_A} represents the condition that the SQ in USS A is not full. The input gate G_{cap_B} represents the condition that the SQ in USS B is not full. The input gate G_{cap_AB} represents the condition that the SQ in USS A or the SQ in USS B is not full. Notice that the input places of these input gates belong to the GSPN model of USS. These input gates enable or disable the request generation. For example, if the condition given by G_{cap_A} is false, which means that SQ of USS_A is full, then the time activity $T_{gen}(A)$ is disabled, which means that request generation procedure of type A request pauses. $T_{gen}(A)$ will be enabled when the condition given by G_{cap_A} becomes true, which means that the request generation procedure resumes when there is a vacancy in the SQ.

$$\mathbf{P} = \begin{bmatrix} 0.2 & 0.2 & 0.6 \\ 0.1 & 0.8 & 0.1 \\ 0.8 & 0.1 & 0.1 \end{bmatrix}$$

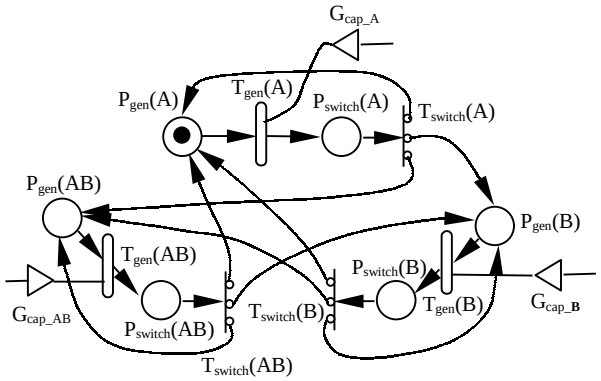


Figure 3 Example GSPN model for an RGS.

2.3 Multi-Server System

We next model a MSS, which is a complex system with several USS's, an RGS and interactions among the components. The requests generated by the RGS are sent to the USS_i with probability p_i through a dispatcher. If the request can only be serviced by USS_i, then p_i is 1. If the request cannot be serviced by USS_i then p_i is 0. In all other cases the probability p_i is controlled by the dispatcher. The optimal dispatch policy can be obtained by solving a Markov decision process.

The GSPN model of the MSS contains the following components:

1. The GSPN models of RGS and USS's.
2. A set of input gates $\{G_{cap\_i}\}$. The input place of a G_{cap_i} is the P_{SQ} of all USS's, which can provide service for request type i. The activity of G_{cap_i} is $T_{gen}(i)$. A gate G_{cap_i} indicates the condition that there are free positions in SQ to buffer the request.
3. Arcs from activity $T_{gen}(i)$ in RGS to place P_{SQ} in any USS that can provide service for request i.

An example GSPN model of a MSS is given in Example 3.3.

Example 2.3 Assume that the MSS contains two USS's as specified in Example 3.1 and one RGB as described in Example 3.2. Figure 4 shows the GSPN model of this MSS.

After generating the controllable GSPN model, we can reduce it into a SPN, which is the same as a GSPN except that SPN does not have instantaneous activities [22]. From the SPN we can find its reachability graph and hence generate the corresponding continuous time Markov decision process. The state s_i of the CTMDP corresponding to the marking M_i in the reachability set. The rate cost of s_i is the sum of the rate costs of places in M_i. The transition cost of the CTMDP from state s_i to s_j is the impulse costs of the completed activities when the GSPN is switching from marking M_i to M_j. The reader may refer to [22] for the procedure of reducing a GSPN to a SPN. The optimal policy is CTMDP is obtained by solving a set of linear programming.

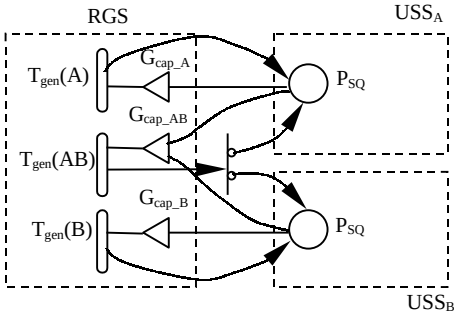
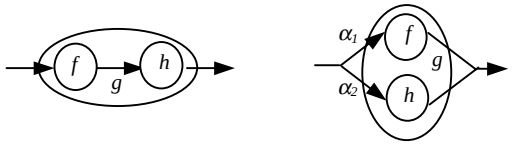


Figure 4 Example GSPN model of a MSS.

2.4 Non-exponential transition time

A GSPN can be converted to a CTMDP, hence it can be evaluated efficiently. However, the exponential distribution is not always an appropriate way to represent the transition time. If the transition time has a general distribution, the Markovian property will be destroyed. In this work, we use the *stage method* [20], which approximates the general distributions using the series or parallel combinations of exponential stages.

An example of a serial server is given in Figure 5 (a). The large oval *g* represents the entire server; the internal structure represents two series-connected stages of the service. A request is first processed in the first stage. After it leaves the first stage it immediately enters the second stage. The times it spends in the first stage and the second stage are independent random variables. We assume that they follow distribution $f(t)$ and $h(t)$, respectively. Only after the previous request departs from the second stage, may a new request enter the first stage. Assume that the entire service time follows distribution $g(t)$, and denote the Laplace transform of $f(t)$, $h(t)$ and $g(t)$ as $F(s)$, $H(s)$ and $G(s)$, we have the relation: $G(s)=F(s)\cdot H(s)$. If $f(t)$ and $h(t)$ are exponential distributions, $g(t)$ is called *Erlangian distribution* [20]. An example of parallel server is given in Figure 5 (b). When a request enters the server it will enter stage *f* with probability α_1 or enter stage *h* with probability α_2. Let the distribution of each stage be $f(t)$ and $h(t)$ and the distribution of overall service time be $g(t)$, we have the relation: $G(s)=\alpha_1\cdot F(s)+\alpha_2\cdot H(s)$. If $f(t)$ and $h(t)$ are exponential distribution, $g(t)$ is called *hyperexponential distribution* [20].



(a) Two stage serial server (b) Two stage parallel server

Figure 5 Series and parallel connection of two servers.

We know that the Laplace transform of an exponential distribution funciton, $f(x)=\mu e^{-\mu x}$, is: $F(s)=\frac{\mu}{s+\mu}$. Given a general distribution function g(t), if its Laplace transform can be written as:

$$G(s)=\alpha_1+\sum_{i=1}^{r}\beta_1\beta_2\cdots\beta_i\alpha_{i+1}\prod_{j=1}^{i}\left(\frac{\mu_j}{s+\mu_j}\right)$$

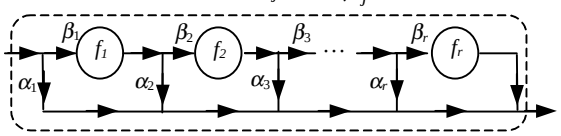


Figure 6 An illustration of the stage expansion method.

It can be represented using the stage-type server given in Figure 6. Other expansions of $G(s)$ and the corresponding stage-type representations are also available. To reduce the complexity, the first two terms of the expansion are used in practice.

III. Experimental results

It has been demonstrated [8]-[10] that DPM techniques based on Markov decision process outperform heuristic policies (e.g., time-out, greedy, etc.) when SP has more than two working modes. Therefore in this experimental setup, although we used a new modeling technique based on GSPN, we will not focus on substantiating previous conclusions. Instead, we demonstrate the importance of accurate modeling of a complex system, and the importance of having an optimal request dispatch policy as a part of optimal DPM policy for power saving in a distributed computing system.

Out target system is a simple distributed computing system. System details are as follows:

The MSS contains two USS's and an RGS.

The RGS generates a single type of requests, which can be serviced by both USS's.

The two USS have different power consumptions and different service speeds. The SP in both USS's has three power states: {*active*, *waiting*, *sleeping*}. However, when in active state, the USSA consumes less power and provide a slower service than the USSB. The average service time of the SP in USS_A is 5ms, the power consumptions are: p_{active}=2.3w, $p_{waiting}$=0.8w, $p_{sleeping}$=0.1w. The average service time of the SP in USS_B is 3ms, the power consumptions are: p_{active}=4.0w, $p_{waiting}$=0.8w, $p_{sleeping}$=0.1w. The time and energy needed for the SP in both USS's to switch from one state to another are the same. Matrix χ gives the transition rate, matrix $\boldsymbol{\varepsilon}$ gives the transition energy. The SQ capacity is 2.

$$\chi=\begin{bmatrix}\infty & 0.2 & 0 & 0\\ \infty & \infty & 1 & 0.5\\ 0 & 0.454 & \infty & 1.5\\ 0 & 0.166 & 1.5 & \infty\end{bmatrix},\quad \boldsymbol{\varepsilon}(s_i,s_j)=\begin{bmatrix}0 & 0 & \infty & \infty\\ 0 & 0 & 1mJ & 2mJ\\ \infty & 4.4mJ & 0 & 0.66mJ\\ \infty & 30mJ & 9mJ & 0\end{bmatrix}$$

Experimental results are obtained from *UltraSAN* simulation of the system [23].

3.1 Experimental setup 1

In this setup, we compare the following methods:

1. Time-out + heuristic dispatch: Time-out policy is used for the power management of each USS. Optimal time-out settings are used to achieve the lowest possible power. As for the dispatch policy, the requests from the RGS are assigned (dispatched) to USS_A or USS_B according to given probabilities P_A and P_B. Different power and delay values are obtained by setting different P_A and P_B values.
2. Greedy + heuristic dispatch: Same as previous one, except that a greedy policy is used for power management of each USS. The greedy policy turns on the SP when it is to be used and turns off the SP immediate when it is not used. Note that Methods 1 and 2 accept only two power modes of the SP, i.e. the "*Active*" and "*Sleeping*" modes.
3. GSPN-based method: In this setup, we only use two power modes for the SP ("*Active*" and "*Sleeping*") when building the GSPN model of the system. The reason for not using all three modes of SP is that we want to demonstrate the importance of the dispatch policy in a distributed computing system. Optimal power management policy is calculated based on the GSPN model. Notice that here the power management policy refers to not only the mode-switching policies for the USS_A and USS_B, but also to the request dispatch policy. The linear programming delay constraints are set such that the delays for our method match the ones of the previous two methods.

The comparison between Methods 1 and 3 are show in the Table 1. The comparison between Methods 2 and 3 are shown in Table 2.

Table 1 Experimental results for comparison between method 1 and our method.

METHOD 1			OUR METHOD	
P_A, P_B	Power	Delay	Power	Delay
0.2, 0.8	2.096	0.314	1.619	0.314
0.4, 0.6	2.095	0.321	1.594	0.322
0.5, 0.5	2.074	0.327	1.573	0.327
0.6, 0.4	2.008	0.335	1.546	0.335
0.8, 0.2	1.906	0.339	1.532	0.339

Table 2 Experimental results for comparison between method 2 and our method.

METHOD 2			OUR METHOD	
P_A, P_B	Power	Delay	Power	Delay
0.2, 0.8	1.802	0.330	1.559	0.330
0.4, 0.6	1.794	0.340	1.525	0.340
0.5, 0.5	1.776	0.346	1.494	0.346
0.6, 0.4	1.733	0.355	1.474	0.355
0.8, 0.2	1.670	0.365	1.442	0.365

3.2 Experimental setup 2

In this setup, we compare the following methods:

1. Optimal USS power management + heuristic dispatch policy: Optimal power management policies for USS_A and USS_B are obtained using the continuous-time Markov decision process model [10]. As for the dispatch policy, the requests from the RGS are assigned (dispatched) to USS_A or USS_B according to given probabilities P_A and P_B.
2. Our method: Same as the third method in Setup 1, except that we use all the three power modes of the SP in this setup to match the SP model used by the first method.

The comparison is shown in Table 3.

Table 3 Experimental results for comparison between Methods 1 and 2.

METHOD 1			OUR METHOD	
P_A, P_B	Power	Delay	Power	Delay
0.2, 0.8	1.162	0.336	0.856	0.336
0.4, 0.6	1.117	0.346	0.834	0.346
0.5, 0.5	1.104	0.353	0.810	0.353
0.6, 0.4	1.085	0.361	0.804	0.361
0.8, 0.2	1.771*	0.370	0.794	0.370

* This huge increase in power dissipation is caused by the inappropriate delay constraint settings when we set delay constraints for USS_A and USS_B without considering the dispatch policy. It illustrates the intrinsic shortcoming of the first method.

From the experimental results, we can see that our method (Method 2 in Setup 2) can save more than 20% power compared to the CTMDP-based method plus heuristic dispatch policy for same delay values. The improvement shows the importance of building an accurate system model and obtaining an optimal dispatch policy along with power management policy for the SP's for the target distributed computing system.

IV. Conclusion

We have introduced a new technique for modeling and solving the DPM problem for systems with complex behavioral characteristics such as concurrency, synchronization, mutual exclusion and conflict. We use controllable GSPN with cost to do the system modeling. The obtained GSPN model can be automatically converted to an isomorphic continuous-time Markov decision process. From the corresponding Markov decision process, we can calculate the optimal DPM policy, which achieves minimum power consumption for given delay constraints. We used the proposed technique to model and solve a power-managed distributed computing system. Experimental results show that, the proposed technique can achieve more than 20% power saving compared to other existing DPM techniques.

REFERENCES

[1] A. Chandrakasan, R. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, July 1995.

[2] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-Power Digital Design", *IEEE Symposium on Low Power Electronics*, pp.8-11, 1994.

[3] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data Driven Signal Processing: An Approach for Energy Efficient Computing", *1996 International Symposium on Low Power Electronics and Design*, pp. 347-352, Aug. 1996.

[4] J. Rabaey and M. Pedram, *Low Power Design Methodologies*, Kluwer Academic Publishers, 1996

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

[6] M. Srivastava, A. Chandrakasan. R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Transactions on VLSI Systems*, Vol. 4, No. 1 (1996), pp. 42-55.

[7] C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *Proc. of the Intl. Conference on Computer Aided Design*, pp. 28-32, November 1997.

[8] G. A. Paleologo, L. Benini, et.al, "Policy Optimization for Dynamic Power Management", *Proceedings of Design Automation Conference*, pp.182-187, Jun. 1998.

[9] Q. Qiu, M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes", *Proceedings of the Design Automation Conference*, pp. 555-561, Jun. 1999.

[10] Q. Qiu, Q. Wu, M. Pedram, "Stochastic Modeling of a Power-Managed System: Construction and Optimization", *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 194-199, Aug. 1999.

[11] L. Benini, A. Bogliolo, S. Cavallucci, B. Ricco, "Monitoring System Activity For OS-Directed Dynamic Power Management", *Proceedings of International Symposium of Low Power Electronics and Design Conference*, pp. 185-190, Aug. 1998.

[12] E. Chung, L. Benini and G. De Micheli, "Dynamic Power Management for Non-Stationary Service Requests", *Proceedings of DATE*, pp. 77-81, Mar. 1999.

[13] L. Benini, R. Hodgson, P. Siegel, "System-level Estimation And Optimization", *Proceedings of International Symposium of Low Power Electronics and Design Conference*, pp. 173-178, Aug. 1998.

[14] U. Narayan Bhat, "*Elements Of Applied Stochastic Processes*", John Wiley & Sons, Inc. 1984

[15] B. Miller, "Finite State Continuous Time Markov Decision Processes With an Finite Planning Horizon." *SIAM J. Control*, Vol. 5, No. 2, pp. 266-281, 1968.

[16] B. Miller, "Finite State Continuous Time Markov Decision Processes With an Infinite Planning Horizon". *J. Of Mathematical Analysis and Applications*, No. 22, pp. 552-569, 1968.

[17] R.A.Howard, *Dynamic Programming and Markov Processes*, Wiley, New York, 1960

[18] D. P. Heyman, M. J. Sobel, *Stochastic Models in Operations Research*, McGraw-Hill Book Company, 1982

[19] G. Bolch, S. Greiner, H. D. Meer and K. S. Trivedi, *Queueing Networks and Markov Chains*, John Wiley & Sons, Inc., 1998

[20] L. Kleinrock, *Queueing Systems. Volume I: Theory*, Wiley-Interscience, New York, 1981.

[21] J. F. Shapiro, *Mathematical Programming: Structures and Algorithms*, John Wiley & Sons, Inc, 1979.

[22] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franceschinis, *Modeling With Generalized Stochastic Petri Nets*, John Wiley & Sons, New York, 1995.

[23] *UltraSAN User's Manual*, Version 3.0, Center for Reliable and high-Performance Computing, Coordinated Science Laboratory, University of Illinois.