
Yuan Tian, Chuang Lin, Min Yao
Computer Science and Technology Department, Tsinghua University
Beijing, China, 100084
{tianyuan, clin, yaomin}@csnet1.cs.tsinghua.edu.cn

ABSTRACT
Server farms are playing an important role in the Internet infrastructure today. However, the increasing power consumption of server farms makes them expensive to operate. Thus, how to reduce the power consumed by server farms has become a important research topic. Power can be thought as a resource of system, just like traditional resources, and we can manage power via improved resource management strategies. In recent studies on power management, the system is attached with multiple states of different power consumption, and by switching among these states, power consumption can be made proportional to the work load. As different job scheduling policies will result in different performance and power consumption, an optimized policy with power as a factor can achieve a better tradeoff between performance and power consumption. In this paper, we summarize some familiar power management policies and propose a novel model using Stochastic Reward Nets(SRN). Based on this model, we analyze the performance and power consumption of different power management policies, and propose a novel cost-aware job scheduling algorithm.

Categories and Subject Descriptors
C.4 [PERFORMANCE OF SYSTEMS]: Modeling techniques; G.1.6 [NUMERICAL ANALYSIS]: Optimization

General Terms
Algorithms, Performance, Theory

Keywords
speed scaling, job scheduling, power consumption, petri nets

1. INTRODUCTION
Traditional computer system design aims to improve performance, but today we are no longer concerned with performance alone. For example, with the development of large-scale computing infrastructures, such as clusters and data centers, the power consumption has kept increasing every year, and we are forced to consider the price of performance [5]. If the power consumption continues to grow rapidly, the cost of the power consumed by a server during its lifetime will exceed the hardware cost. It was estimated that data centers in the US consumed about 61 billion KWh of electricity, which incurred a cost of about 4.5 billion dollars [1]. Thus, the reduction of power consumption in data centers is not only a critical problem concerned in academy, but also a key issue in the industry, because of economical, environmental and marketing reasons. Since server farms are ubiquitous in data centers, in this paper we will mainly focus on how to reduce power consumption in server farms.

Due to over-provisioning and redundancy design, only 20-30% of the total capacity of servers in a typical data center are utilized on average and the idle servers consume as much as 60% of their peak power even when they are doing nothing [6]. This goes against of the goal of the green networking and results in huge energy waste. While, on the other hand, it provides us the opportunities to save power, for example, by strategically turning on/off servers. Recently, a number of schemes [8] have been proposed to address the problem using power management techniques, e.g., dynamically provisioning servers to right-size the data centers [22], scaling speed to adapt the system speed to the workload [27] [4], and load concentration or unbalancing job scheduling policies [25] [12], [2]. By treating the power consumption as another metric besides response time and throughput, traditional job scheduling and resource allocation designs should be modified. In this paper we study the problem of jobs scheduling when speed scaling is taken into consideration.

Unlike traditional load balancing strategies, energy-saving job scheduling requires jobs be dispatched to fewer nodes under unbalanced operations, so as to make more chances for turning off nodes in server farms. Instead of turning system off, speed scaling can adapt the system speed so as to balance the performance and power metrics. Speed scaling designs can be classified into dynamic speed scaling, which adapts the speed at all times based on the current state, and gated-static speed scaling, which runs at a static speed chosen for balancing power and performance, except when idle. Speed scaling is interacted with job scheduling, which has been studied by some literatures. Here we study the tradeoff between performance and power consumption in job scheduling with speed scaling, to gain insights into issues of: i) How does system perform under job scheduling interacted with speed scaling in terms of traditional performance metrics and power consumption? ii) How to design optimal scheduling policy with speed scaling concern?

To address the first question, we propose a formal performance evaluation framework using the Stochastic Reward Nets (SRN). SRN can be used as a formal, graphical, executable method which is particularly well-suited to model parallel system resource manage-
2. RELATED WORK

A number of researches have been done on green data center, and they focus on power-efficient mechanisms and policies. Different policies use different algorithms for power management, and their impact on power efficiency and system performance are different. So it is important to qualify and quantify these policies, and comparison among multiple algorithms may be more favorable to make improvements. The evaluation of these policies are mainly dependent on measurement of real systems or analysis using mathematical models. The former approach, which is straightforward, obtains the performance metrics by running particular benchmarks on real systems. The latter uses mathematical models such as Queueing Theory [10] [14] [17], Markov models [14] [19], etc. However, the models used in recent researches are not enough nor intuitive to reflect the relationship among multiple factors.

A.Gandhi et al. [17] used Queueing Theory to address the optimal energy-performance tradeoff in server farms under a very small natural class of server farm management policies. B.Guenter et al. [19] employed Markov model to predict idleness and developed power state transitions and server provisioning methods to remove idle servers. By solving stochastic optimization problems, this model can quickly identify the optimal assignment of servers. Both of the above studies utilize low power states (off, sleep) instead of speed scaling. When the number of power states increases, these models can be quite complicated for computing the optimal strategy. Furthermore, these solutions lack extensibility, and thus it is hard to be used in large-scale systems, where significant re-computation might be needed when statistics change. In this paper, we use a different approach that can adapt to these challenges. Our approach is based on the Stochastic Petri Nets model which can characterize these power states and transitions in quite an intuitive way. Furthermore, it has a better extensibility which allow us to characterize the speed scaling with a larger range of power states involved.

3. OVERVIEW OF POWER MANAGEMENT IN SERVER FARMS

Power is another resource in server system like other traditional resources [11]. The traditional resources such as processor, bandwidth are utilized attached with power consumption, so we can manage power via the improvement of traditional resource management. The power management mechanisms breaks the constraint that a system has only two states(on and off), instead, giving some additional states to the system through hardware or software approaches, for example, the different speeds of the servers. Each state has different power level and the states can transfer among each other. By managing these multiple power states, we can make the power consumption proportional to the utilization of system resources which is defined as a system attribute: the energy proportionality [7]. In server farms, the processing capacity of servers occupies a large proportion, so we mainly take the speed of the processor in servers as the primary system resource.

To achieve the energy proportionality, the system designer must strike a balance between power and performance by carefully selecting the state at which the system will run. For example, before putting an idle server to sleep to save power, we must consider the setup time to awake the sleeping server which will incur a significant cost. The setup cost takes the form of both a time delay and an energy penalty. Optimizing this tradeoff is a hard problem, since there are an infinite number of possible server farm management policies. Our work is built on the speed scaling technology [26] which allows load aware adaptation of the speed at which the system is run, in order to save power.

Fundamentally, the important questions in server farms include the scaling of speed and the job scheduling policy. To optimize the power-performance tradeoff, the scaling of speed must decide (1)At what speed should the servers run? (2)When should servers be turned off/p to sleep/put to sleep/put to sleep/ turned on/switched speed? And the job scheduling policy must decide the policy. (3)How to assign jobs to servers? In this section, we will present several frequently-used policies including both the scaling of speed and job scheduling with respect to the three issues above.

3.1 Data Center Architecture

Figure 1 illustrates an abstract data center architecture. The end users send their requests to the cloud and fetch what they want. Incoming jobs first arrive at the front-end proxy server which controls the scheduling of all incoming jobs globally and dispatches an incoming job to any node according to the dispatching policy. At the middle-tier nodes, each web app node will provide service with a job waiting queue. A capacity is set for each waiting queue to limit the maximum number of waiting jobs. Each web app node will access the data storage nodes which are located in different domains at the back-end for dynamic data request. Popular multi-tier workloads include social networking, streaming media, web search, etc. However, not all the workloads are multi-tiered, such as compute-intensive workload, static personal home page, etc. In this paper we focus on the single-tier architecture in data center for simplification.

3.2 The Scaling of Speed

The scaling of speed controls the power of a server by means of controlling the service speed of the server, the relationship between server speed and power consumption will describe in section 5. It will choose one speed from the speed set \( S = \{ s_n | n = 1, 2, ..., N \} \) which includes finite discrete states between a particular speed interval such as \([0, \text{MAX\_SPEED}]\). For the limit of today’s technology, the speed set \( S \) is a limited set of possibilities speed which is
not consecutive. When the service speed is 0, the server is at sleep state or off state. For each speed \( s_i \), the associated power values are \( P_i \), and satisfies if \( s_i < s_j \) then \( P_i < P_j \).

### 3.2.1 Power-down Policy

The idle time of various time scale exists in different parts of the system during the running [23], which make the chance to put the idle system resources into low-power level states. We consider a specific set of power-down policies for managing a single server as shown in Table 1 [16]. In ON/OFF policy, one could immediately turn it off when the server goes idle. While a server in sleep mode consumes more power than an off server, the setup cost for a sleeping server is lower than that for an off server, so in SLEEP/WAKE policy on could move the server to a specific sleep state. Base on the hardware support, there may be several sleep states with different depths [20], the deeper the sleep, the lesser the power consumption and the longer wake-up time cost. We can regard OFF state as the deepest sleep state and the ON/OFF policy is a special case of SLEEP/WAKE policy.

Random policy uses a stochastic factor to turn an idle server off with some probability \( p \), and leave it idle with probability \( (1-p) \). In time-out policy, a timer is defined as the minimum amount of time allowed between two state transitions. The server can adapt the time-out policy when transitions is needed from one state to another. For example, the server could power down at the expiration of an inactivity timer, which is started when the system is detected to be idle (i.e., there are no active jobs). Also when jobs arrive, one could power up the server until the minimum hold time has expired, which will cause a certain number of jobs have accumulated in the queue.

### 3.2.2 Speed Scaling Policy

When servers go to low-power state in power-down policies, they cannot work normally to process jobs which is not appropriate for the case that the server has a continuous light load for some period. Thus except from the sleep and awake states, we need a more general speed scaling policy to allow that the speed of server can vary in a wider range with different load. In speed scaling policy, the speed set \( S \) is larger than that of power-down policy, attached with different energy consumption correspondingly. In light load, the utilization of the server is low, slowing down to process jobs can reduce the power consumption with little loss in performance, on the contrary, in heavy load, the utilization is high, the server must speed up to maintain a good performance. From another perspective, the power-down policy is the special case of speed scaling policies.

The policy design can be highly sophisticated - using the optimization theory [9] to adapt the speed at all times with the current server state to balance power and performance measures. For simplicity, in this paper we will use the static speed scaling which chooses a special speed \( s_i \) of server \( i \) for a given arrival rate \( \lambda_i \), especially, when \( \lambda_i \) is zero, the server speed is zero correspondingly. The speed \( s_i \) can be obtained by many mathematical methods and we will apply one of these methods in Section 5. We will also consider the buffer-threshold policy which is widely used in many devices. In buffer-threshold policy, the buffer occupancy or queue length is relative to the server utilization, according to which the speed transfer from one state to another.

### 3.3 Job Scheduling Policy

Job scheduling is one of the major mechanisms of power managements which can solve the resource competition and sharing. Scheduling policy determined the rule to assign jobs, based on which different jobs are assigned to different queues. Load balance policy assigns jobs equally to the servers they connected, it uses the max number of servers and the load is spread among all the servers running. On the contrary, load concentration policy assigns jobs to a few servers that meet performance demand, so the load is concentrated in a few of servers and the other servers can be put into sleep or shut down to save power. The two policies have different power consumption and performance trade off and are appropriate for different scenarios. The load balance policy can avoid local overheating when some servers are used frequently, and the load concentration policy is appropriate for the basic power consumption (the power consumption when server has no load) occupy a high proportion in the whole power consumption. Andrew et al. studied how the speed scaling interact with scheduling and indicates that the speed scaling can be decoupled form the

<table>
<thead>
<tr>
<th>Table 1: Power down Policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>ON/OFF</td>
</tr>
<tr>
<td>SLEEP/WAKE</td>
</tr>
<tr>
<td>Random policy</td>
</tr>
<tr>
<td>Time-out policy</td>
</tr>
</tbody>
</table>

Figure 1: Abstract server farm workload architecture
4. SYSTEM MODELING USING STOCHASTIC PETRI NETS

In this section we will briefly recall the basic concepts of the Generalized Reward Nets (SRN) and introduce the basic model and power model for power management.

4.1 SRN Introduction

Petri Nets [15] is a graphical and mathematical tool used for modeling and analyzing systems with properties such as concurrent, asynchronous, distributed and parallel. The formal definition concerning Petri Nets is given below: A Petri Net (PN) is a 6-tuple \((P, T, F, K, W, M_0)\). \(P = \{p_1, p_2, \ldots, p_k\}\) is a finite set of places. \(T = \{t_1, t_2, \ldots, t_m\}\) is a finite set of transitions \((P \cap T) = \emptyset\). \(F \subseteq ((P \times T) \cup (T \times P))\) is a set of arcs. \(K\) is the capacity of a place and \(W\) is the value of arc. \(M_0 = \{m_{01}, m_{02}, \ldots, m_{0u}\}\) is an initial marking. A Stochastic Petri Net (SPN) is obtained from a PN in which the transition firing time is an exponentially distributed random variable. \(R = \{r_1, r_2, \ldots, r_m\}\) is a set of firing times such that \(r_i\) is the rate of exponential distribution associated with the firing time of transition \(t_i\).

SRN is an extension of Stochastic Petri Nets (SPN), which is associated each marking with cost. It can be used to model various scheduling algorithms in power management policies. In the rest of this section, we respectively introduce the basic model of resource scheduling and the power model. Then, the whole model is constructed by combining the two models above.

4.2 Basic Model

In basic model, the power saving policies are not considered which indicates that all the servers are running at full speed without speed switch. And we assume that our system comprises \(m\) computing nodes, i.e. \(m\) servers \(S_i (i = 1, 2, \ldots, m)\) and \(n\) global schedulers \(GS_j (j = 1, 2, \ldots, n)\). Global schedulers are connected to all or some of the nodes and each global scheduler and node have a buffer for waiting jobs. A threshold is set for each buffer to restrict the maximum number of waiting jobs. For analytic tractability, we assume the jobs arrival are Poisson distribution with mean rate \(\lambda_j\) for global scheduler \(GS_j\) and the service time are exponentially distributed. The meaning of the transitions and places are shown in table 1.

Figure 2 shows the model of resource scheduling architecture in figure 1. In this architecture, jobs are submitted to global scheduler independently which are represented by timed transitions associated with exponentially distributed firing time. Global schedulers dispatch the jobs they received to one of the servers according to some scheduling policy. The dispatch procedure are represented by a free-choice conflict transition with zero firing, all the jobs to the same servers are joint into the local queue \(LQ_j\) waiting for execution. The local scheduler allocates the jobs in the local queue according to some local allocation policy. The allocation of server \(S_j\) is modeled by instantaneous transitions \(g_{si}\). The processing of jobs on server \(S_j\) is modeled by time transition \(p_{ij}\).

Different global scheduling policies and local allocation policies are modeled by enabling functions and firing probabilities of \(gs_{ij}\) and \(l_{si}\) respectively, which will be described in section 4.

4.3 Power Model

In power model, we add the concept of power management into consider. The core ideas of power management are putting servers into different power states and enabling states transfer from each other to make the power consumption proportional to the utilization of system resources. For general condition we assume each server has a speed set which contains three states which are sleep state, low speed state and high speed state. We can extends our model to a larger speed set which can be more complex and wider, but this is out of the scope of this paper.

The power model is depicted in figure 3. We assume there is no job initially and server \(S_j (j = 1, 2, \ldots, m)\) is in sleep denoted by putting a token in place \(SLEEP\). When jobs arrive, according to

![Figure 2: Basic model using petri nets](image-url)

Table 2: Model parameters explanations

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Distribution and Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_i)</td>
<td>Timed transitions, job arrival with poisson distribution, the arrival rate is (\lambda_i)</td>
</tr>
<tr>
<td>(GQ_i)</td>
<td>Global queue for waiting jobs to be scheduled by global scheduler</td>
</tr>
<tr>
<td>(g_{si})</td>
<td>Free-choice conflict transition, global scheduling which schedule jobs to the server they connect</td>
</tr>
<tr>
<td>(TQ_{ij})</td>
<td>A queue of jobs waiting for transportation</td>
</tr>
<tr>
<td>(t_{ij})</td>
<td>Transportation of jobs from global scheduler (GS_i) to site (S_j)</td>
</tr>
<tr>
<td>(LQ_i)</td>
<td>Local waiting queue for a single server (S_j)</td>
</tr>
<tr>
<td>(l_{si})</td>
<td>Local scheduling with FCFS policy</td>
</tr>
<tr>
<td>(PQ_{ij})</td>
<td>A queue of jobs waiting for being processed by processor (S_j)</td>
</tr>
<tr>
<td>(p_{ij})</td>
<td>The processing of jobs on server (S_j)</td>
</tr>
<tr>
<td>(r_{si})</td>
<td>Free choice conflict transition for deciding server states</td>
</tr>
<tr>
<td>(SLEEP_{j},HIGHT_{j},LOW_{j})</td>
<td>Indicate server (S_j) in sleep state, high speed states and low speed states</td>
</tr>
<tr>
<td>(D_{m2n}(m,n \in (s,H,L)))</td>
<td>Guard for states transfer</td>
</tr>
<tr>
<td>(T_{m2n}(m,n \in (s,H,L)))</td>
<td>Process for states transfer with power and time cost</td>
</tr>
</tbody>
</table>
some policy of speed scaling, it maybe transfer to low speed state or high speed state denoted by place HIGH and place LOW. When no job to process, it may go to sleep state again. In sleep state the servers are unable to process until be waked up under some condition. When load is light, it may transfer from high speed state to low speed state vice versa. The transition among these states are modeled by instantaneous transition with enabling functions as a guard based on power management policies and timed transitions will attach the transition cost including time and power consumption as penalty.

### 4.4 System Modeling

The whole model can be constructed by combining basic model and power model shown in figure 5 using enabling functions of free-choice conflict transition $r_{sij}$ and instantaneous transition $g_{sij}$ as a guard. The enabling functions are different according to speed scaling and job scheduling policies for power management which will be described in the next section.

## 5. SCHEDULING POLICIES AND SPEED SCALING

As depicted in section 2, for global scheduling policies we mainly focus on the load balance and load concentration scheduling and the two global policies can be specified by assigning different enabling functions and firing probabilities for transition $g_{sij}(i = 1, 2, ..., n; j = 1, 2, ..., m)$.

### 5.1 Global Scheduling Policies

We consider the load balance and load concentration scheduling policies for global scheduler $G_S(i = 1, 2, ..., n)$ to decide how to schedule jobs to each server $S_i$. The enabling functions and firing probabilities for transition $g_{sij}(i = 1, 2, ..., n; j = 1, 2, ..., m)$ are specified as follows:

- **Load Balance**: Selecting the server that has the shortest waiting queue length and the queue is not full, to make all the servers has a balance load. Enabling functions of $gs_{ij}$ is  
  $$F(gs_{ij}) = (M(TQ_{ij}) < C(TQ_{ij})) \land (M(LQ_i) \leq M(LQ_{j}))$$

  Where $C(TQ_{ij})$ denotes the threshold of places $TQ_{ij}$ and $M(LQ_{j}) = \min_{j=1}^{m}(LQ_j|M(LQ_j)) < C(LQ_i)$

- **Load Concentration**: Selecting the server that has the longest waiting queue length and the queue is not full, to make all the other server which has no jobs to process go to sleep or be shut off in order to save power.

  Enabling functions of $gs_{ij}$ is  
  $$F(gs_{ij}) = (M(TQ_{ij}) < C(TQ_{ij})) \land (M(LQ_i) \geq M(LQ_{j}))$$

  Where $C(TQ_{ij})$ denotes the threshold of places $TQ_{ij}$ and $M(LQ_{j}) = \max_{j=1}^{m}(LQ_j|M(LQ_j)) < C(LQ_i)$

  for both of the two global scheduling policies, firing probability of $gs_{ij}$ is

  $$P_{gs_{ij}} = \begin{cases} 1/\|U_i\| & \text{if } gs_{ij} \in U_i \\ 0 & \text{otherwise} \end{cases}$$

  Where $U_i = gs_{ij}|F(gs_{ij}) = TRUE, j = 1, 2, ..., m)(i = 1, 2, ..., n)$.

### 5.2 Speed Scaling

#### 5.2.1 Power Down Policy

As depicted in Table 1, the first two policies can be specified by figure 4, when a server decide whether to go to sleep, the enabling function of transition $D_{12a}$ and $D_{12s}$ is:

$$F(D_{12a}) = F(D_{12s}) = (M(LQ_i) = 0)$$

and the enabling function of transition $D_{2a}$ is:

$$F(D_{2a}) = (M(LQ_i) > 0)$$

In random policy, the firing probability of transition $D_{a}$ in figure 4 is $P(D_{a}) = a(0 \leq a \leq 1)$ and the firing probability of transition $D_{2a}$ is $P(D_{2a}) = b(0 \leq b \leq 1)$

Time-out policy can be modeled by adding a determined time transition $T_{sleep}$ to figure 4 which denoting the delay of going to sleep state, the time value of $T_{sleep}$ is $\alpha(\alpha \geq 0)$, likewise, the determined time transition $T_{wake}$ denote the delay of waking a sleep server, the time value of $T_{wake}$ is $\beta(\beta \geq 0)$. The enabling function of transition $D_{a}$ is:

$$F(D_{a}) = (M(LQ_i) = 0)$$

and the enabling function of transition $D_{2a}$ is:

$$F(D_{2a}) = (M(LQ_i) > 0)$$

#### 5.2.2 Speed Scaling Policy

For static speed scaling, we can set firing rate for each time transition which present the service rate corresponding to different arrival rates. And in buffer-threshold policy, the buffer occupancy threshold is defined as $k(0 \leq k \leq MAX\ CAPACITY)$. 

![Figure 3: Three states switch](image)

![Figure 4: Sleep and wake switch](image)

![Figure 5: The system model of power management](image)
The enabling function of $rs_{i,j}$ and $r_{s_{i,j}}$ in figure 5 is $F(r_{s_{i,j}}) = (0 \leq M(LQ_j) < k) \Rightarrow F(rs_{i,j}) = (k \leq M(LQ_j) \leq C(LQ_j))$. Correspondingly, the enabling function of $D_{1/2H}$ and $D_{H/2L}$ is $F(D_{1/2H}) = (M(WQ_{1/2}) > 0) \Rightarrow F(D_{H/2L}) = (M(WQ_{H/2}) > 0).

The guard $gd_{1,j}$ and $gd_{2,j}$ is $F(gd_{1,j}) = (M(LOW) > 0), F(gd_{2,j}) = (M(HIGH) > 0).

6. POLICY OPTIMIZATION

To make a tradeoff between the load balance and load concentration scheduling policies, we introduce an optimal global scheduling policy, the cost-aware optimal scheduling, interacted with speed scaling in this section. The model we developed is simple but general that captures the major issues that effect the design of scheduling policy. And we propose a distributed cost-aware scheduling algorithms to achieve the optima.

6.1 Cost Model

We consider the architecture shown in Figure 1, with a set of $m$ servers and a poisson arrival rate of $\lambda$. Each server has a service rate $s_i$. We focus on the homogeneous structure in server farms, and model the cost of a server by a weighted sum of response time costs and power costs which is the same for all servers

$$M_i = E[P_i(s_i)] + \beta_i\lambda_i T_i$$

(1)

where $E[T_i]$ is the traditional performance metric which the end users concern and the $E[P_i(s_i)]$ is the power cost of server $i$, $\beta_i > 0$ is used to characterize the relative weight of the power cost and the response time cost. The mean response time with service rate $s_i$ can be modeled using an M/Gi/1 PS queue which takes the form

$$E[T_i] = \frac{1}{s_i - \lambda_i}$$

(2)

The power consumption of a single server can be estimated based on the current server speed, $s_i$, and the modeling of the power function $P_i(s_i)$ is an open topic, many researches show it can take different forms depending on specific system. The low-order polynomial form

$$P_i(s_i) = k_1 s_i^{\alpha_i} + k_2$$

(3)

provides a good approximation, which is the sum of the dynamic power consumption and the static power consumption. $k_1$, $k_2$ and $\alpha$ are parameters associated with the device involved (While $\alpha$ has been usually assumed to be around 3) [21]. The combination of the models above gives the cost for server $i$

$$f_i(\lambda_i, s_i) = \frac{\lambda_i}{s_i (k_1 s_i^{\alpha_i} + k_2) + \beta_i \lambda_i - s_i}$$

(4)

6.2 Policy Optimization by Decomposition Theory and Convex Optimization

Given the cost model above, we focus on two important decision-s for the server farms: (i) the global schedule algorithm to dispatch jobs to servers, i.e., determining $\lambda_i$, the arrival rate to server $i$ and $\sum_{i=1}^{m} \lambda_i = \lambda$, and (ii) Given arrival rate $\lambda_i$, each server $i$ choose a speed $s_i$ to minimize the cost in formula(4). Speed scaling can adapt the service rate to the work load. In our model, we assumed static speed scaling which will choose a service rate $s_i$ for a given arrival rate $\lambda_i$. The choice depends on the cost function $f_i(\lambda_i, s_i)$. When $\lambda_i$ is zero, the service rate is zero correspondingly which means putting the server to sleep or shut down. So we can acquire how many servers are needed to handle the arrival jobs which is denoted by $n^*$ and at what service rate each server should run which is denoted by $\lambda_i$. To minimize the cost, assume the problem $min f_i$ has a unique solution $s_i(\lambda_i)$ and it will satisfy [13]

$$\frac{\partial f_i(\lambda_i, s_i)}{\partial s_i} = 0$$

(5)

Given the speed scaling $s_i(\lambda_i)$, the goal of our model is to determining $\lambda_i$ to minimize the total cost of the server farms, which is captured by the following optimization

$$\min \lambda_i \sum_i^{m} \frac{\lambda_i}{s_i(\lambda_i)} (k_1 s_i^{\alpha_i}(\lambda_i) + k_2) + \beta_i \frac{\lambda_i}{s_i(\lambda_i) - \lambda_i}$$

(6)

s.t. $\sum_i^{m} \lambda_i = \lambda$

where $s_i(\lambda_i)$ satisfies

$$\frac{\beta_i}{s_i(\lambda_i)} = k_1 (\alpha_i - 1) s_i^{\alpha_i - 2}$$

(7)

By Solving $s_i(\lambda_i)$ first, the above problem reduces to

$$\min \lambda_i \ h_i(\lambda_i)$$

s.t. $\sum_i^{m} \lambda_i = \lambda$ (8)

We can decompose the original problem into distributively solvable subproblems which are then coordinated by a high-level master problem. In this paper, we apply dual decomposition theory [24] which is based on decomposing the Lagrangian dual problem. Lagrangian duality theory links the original minimization problem (8), termed primal problem, with a dual maximization problem, which sometimes readily presents decomposition possibilities. The Lagrangian duality can relax the original problem (8) by transferring the constraints to the objective in the form of a weighted sum. The Lagrangian of (8) is defined as

$$\min \lambda_i \ h_i(\lambda_i) + \sum_i^{m} \nu_i (\sum_i^{m} \lambda_i - \lambda)$$

(9)

Where $\nu_i$ is the Lagrange multiplier. Such that the optimization is separated into two levels of optimization. At the lower level, we have the subproblems in which for each $i$ (9) decouples

$$\min \ g_i(\nu_i) = h_i(\lambda_i) + \nu_i \lambda_i$$

(10)

At the higher level, we have the master dual problem in charge of updating the dual variable $\nu$ by solving the dual problem

$$\max \ g(\nu) = \sum_i g_i(\nu) - \sum_i \nu_i \lambda_i$$

(11)

where $g_i(\nu)$ is the dual function obtained as the maximum value of the Lagrangian solved in (9) for a given $\nu$. The master dual problem is always a convex optimization problem even if the original problem is not convex. In general, the function $g(\nu)$ may or may not be differentiable, so we can solve the master dual problem in (11) with a subgradient method. Given the Lagrange multiplier $\nu$, we can solve the subproblem by

$$\lambda_i^* = \arg \min [h_i(\lambda_i) + \nu_i \lambda_i]$$

(12)

The dual function is differentiable and the following gradient method can be used

$$\nu_i(t + 1) = [\nu_i(t) - \alpha(t) (\lambda - \lambda_i^*(\nu_i(t)))]$$

(13)

where $t$ is the iteration index, $\alpha$ is a sufficiently small positive step-size. The dual variable $\nu(t)$ will converge to the dual optimal $\nu^*$.
as \( t \to \infty \) and the primal variable \( \lambda^*(\nu(t)) \) will also converge to the primal optimal variable \( \lambda^* \).

We can apply a distributed algorithms to solve the problem (10)-(11).

Standard Dual Algorithm to Solve the Problem (8)

- **Parameters:** each server needs its function \( h_i \) and total arrival rate \( \lambda \).
- **Initialization:** set \( t = 0 \) and \( \nu_i(0) \) equal to some nonnegative value for all \( i \).
  1) Each source locally solves its problem by computing (12) and then gets the solution \( \lambda^*(\nu(t)) \).
  2) Each link updates its prices with the subgradient iterate (13) and gets the new \( \nu_i(t+1) \).
  3) Set \( t \to t + 1 \) and go to step 1 (until satisfying termination criterion).

7. MODEL ANALYSIS

Based on the SRN model described in section 4 and 5, we can make performance evaluation and power consumption analysis for different policies. In the rest of this section we investigate the impact of several factors on system performance and power consumption, specifically, the job scheduling, and make the verification and comparison for the cost-aware scheduling using optimization theory.

7.1 Average Power Consumption

Different server speed states have different power consumption, we can obtain the whole power consumption by associating each place and transition which denoting the server states and state switch-es with power cost. We assume the relationship between server place and transition which denoting the server states and state switch-es, we can obtain the whole power consumption by associating each server.

For an arbitrary policy, the steady state probability for the place \( s \) which contains \( i \) tokens can be obtained by

\[
Pr\{M(s) = i\} = \sum_j Pr\{M_j\} \tag{14}
\]

Where \( M_j \in [M_0 > and M_j(s) = i] \).

Furthermore, let \( P_t \) represents the power consumption when server runs at state \( s_j \) \( i \in \text{speed set } S \), then for an arbitrary power management policy, the average power consumption of each server is:

\[
P = \sum_{j \in S} P_j \times Pr\{M(s) = 1\} \tag{15}
\]

7.2 Average Response Time

The average response time can be obtained by computing the average round-trip time of one token in a close loop net. First we calculate the average number of tokens in the sub-system Ps which eliminates the state place:

\[
\bar{N}_k = \sum_{p_k \in Ps} \bar{u}_k \tag{16}
\]

where \( \bar{u}_k \):

\[
\bar{u}_k = \sum_m m \times Pr\{M(p_k) = m\} \tag{17}
\]

Then we calculate the flow rate of transition into the sub-system Ps:

\[
R(t, s) = W(t, s) \times U(t) \times \lambda \tag{18}
\]

where \( \lambda \) is the average fire rate of transition \( t \) and \( U(t) \) is the sum of probability of stability of all markings which enable transition \( t \):

\[
U(t) = \sum_{M \in E} Pr\{M\} \tag{19}
\]

\( E \) is the reachable marking set enabling transition \( t \). Finally, the flow rate in and out of the subsystem are the same, so we use Little formula obtain the average response time:

\[
T_{Response} = \bar{N}/r \tag{20}
\]

where \( r \) is the flow rate of transition into the sub-system.

7.3 Cost Measures

The cost model for optimization is described in Section 5, which takes the form of the weighted sum of the power costs and response time costs. So the costs for a server farm whose capacity is \( N \) servers are

\[
Cost_N = \sum_{i=1}^{N} \sum_{j \in S} P_{ij} \times Pr\{M(s_j) = 1\} + \beta_i \times \bar{N}_i/r_i \tag{21}
\]

where \( \beta_i > 0 \) is used to characterize the relative weight of the above two costs.

8. NUMERICAL EXAMPLES

In this section, we give some numerical examples of analysis in section 7, mainly focus on the average power consumptions and response time. For petri nets, TimeNET (Timed Petri Net Evaluation Tool) [18] is used to calculate the analytic solutions, which contains a new JAVA-based user interface, and the latest version 4.02 for windows is used. We consider a system with 10 servers with static speed scaling, each server will choose a single service rate for a given arrival. We can scale the system up to more servers, however, the time cost by simulation may increase correspondingly. We assume the main parameters in power function \( P(s_i) \) are \( k_1 = 4 \), \( k_2 = 60 \) and \( \alpha = 2 \), which may vary in different systems.

We first show power consumption and response time individually for the balance and concentration scheduling in figure 6, we can see that when the time interval of job arrival is lower than about 0.2, which means the arrival rate, the reciprocal of time interval, is larger than 5, the balance scheduling is better than load concentration scheduling. When arrival rate is lower than 5, the load concentration scheduling is better. For response time, when arrival rate is at the interval from 5 down to 1.25 , the balance scheduling is better. Thus, we conclude that when the incoming load drops below a certain threshold, we might apply load concentration policy among fewer servers and turn off the unused servers to save power.

Either of two scheduling policies is not better than the other one all the time. Since the environment of network is diverse and complicated, we cannot deploy either single policy for maintaining the optimal tradeoff between performance and power consumption. To better solve the problem, we employ the cost-aware scheduling policy described in section 6 in the same petri nets structure and modify the enable functions of each transition according to optimal server provisioning. We assume the weighted factor \( \beta = 8 \).

When the arrival rate increases, the optimal \( n^* \) is changing which indicates provisioning \( n^* \) servers for the workload and putting the
other servers out of work. We found that the optimal scheduling among these $n^*$ servers is load balance, in another words, the arrival rates assigned to each server in $n^*$ are the same, which is $\lambda/n^*$. The other $(N - n^*)$ servers are shut down according to the speed scaling policy. The costs for the load balance, load concentration and cost-aware scheduling are illustrated by figure 7, when $\lambda$ is low, fewer servers are needed, thus, the load balance scheduling gains the worst cost. Since the $n^*$ servers which cost-aware scheduling used are the same as the load concentration scheduling when the $\lambda$ goes low enough, the two curves are tend to be almost approximated. To the contrary, more servers are needed when $\lambda$ goes high and the load concentration scheduling gains the worst cost. When the $\lambda$ goes high enough, the curves of load balance and cost-aware scheduling are tend to be almost approximated.

9. CONCLUSION AND FUTURE WORK

We present several frequently-used power management policies including both speed scaling and job scheduling. These policies have different power and performance tradeoffs. In order to evaluate different policies we propose a novel modeling technique for power management policies in server farms using SRN. The whole model can be completed by basic model and power model with enable function is used for connecting them. Based on this model, we analyze the load balance and concentration scheduling policies interacted with speed scaling policies, and apply the optimization theory to our model to acquire a better performance and power consumption tradeoff. Through the solution of SRN, we present the formal formula of average power consumption and response time measures and give some numerical results of comparing our cost-aware scheduling costs with balance and concentration scheduling policies under different job arrivals.

In future work, we will make effort in two aspects. The first is investigating arbitrary power function for optimization which is an open question under research. Second, petri nets can easily model the homogeneous and heterogeneous structure. But for optimization, heterogeneous structure is more complex than the homogeneous structure. How to extend our optimal model to the heterogeneous structure in server farms is the other aspect we will focus on.

10. ACKNOWLEDGMENTS

This work is supported by the National Grand Fundamental Research 973 Program of China (No. 2010CB328105, No. 2009CB320504), and the National Natural Science Foundation of China (No. 60932003, No.61020106002).

11. REFERENCES


