Vol.2009-DPS-140 No.2
2009/9/10

Energy-Efficient Computation Models for Cluster Systems

AILIXIER AIKEBAIER,[†1] YAN YANG,[†1] TOMOYA ENOKIDO[†2]
and MAKOTO TAKIZAWA [†1]

Information systems are composed of various types of computers interconnected in networks. In addition, information systems are being shifted from the traditional client-server model to the peer-to-peer (P2P) model. The P2P systems are scalable and fully distributed without any centralized coordination. It is getting more significant to discuss how to reduce the total electric power consumption of computers in information systems in addition to developing distributed algorithms to minimize the computation time. In this paper, we do not discuss the micro level like the hardware specification of each computer. We discuss a model to show the relation of the computation and the total power consumption of multiple peer computers to perform types of processes at macro level. We also discuss algorithms for allocating a process to a computer so that the deadline constraint is satisfied and the total power consumption is reduced.

1. Introduction

Information systems are getting scalable so that various types of computational devices like server computers and sensor nodes[1)] are interconnected in types of networks like wireless and wired networks. Various types of distributed algorithms[6)] are so far developed, e.g. for allocating computation resources to processes and synchronizing multiple conflicting processes are discussed to minimize the computation time and response time, maximize the throughput, and minimize the memory space. On the other hand, the $green\ IT$ technologies[4)] have to be realized in order to reduce the consumptions of natural resources like oil and resolve air pollution on the Earth. In information systems, total electric power consumption has to be reduced. Various hardware technologies like low-power consumption CPUs[2),3)] are now being developed. Biancini $et\ al.$[8)] discuss how to reduce the power consumption of a data center with a cluster of homogeneous server computers by turning off servers which are not required for executing a collection of web requests. Various types of algorithms to find required number of servers in homogeneous and heterogeneous servers are discussed[5),9)]. In wireless sensor networks[1)], routing algorithms to reduce the power consumption of the battery in a sensor node are discussed.

In this paper, we consider peer-to-peer (P2P) overlay networks[7)] where computers are in nature heterogeneous and cannot be turned off by other persons different from the owners. In addition, the P2P overlay networks are scalable and fully distributed with no centralized coordination. Each peer has to find peers which not only satisfy QoS requirement but also spend less electric power. First, we discuss a model for performing processes on a computer. Then, we measure how much electric power a type of computers spends to perform a Web application process. Next, we discuss *simple* and *multi-level* power consumption models for performing a process in a computer based on the experiments with servers and personal computers. In the simple model, each computer consumes maximally the electric power if at least one process is performed. Otherwise, the computer consumes minimum electric power. The simple model shows a personal computer with one CPU independently of the number of cores. In the multi-level model, the energy consumption of a computer depends on how many processes are concurrently performed. A server computer with multiple CPUs follows the multi-level model. A request to perform a process like a Web page request is allocated to one of the computers. We discuss allocation algorithms to reduce not only execution time but also power consumption in a collection of computers. In the algorithms, processes are allocated to computers so that the deadline constraints are satisfied based on the laxity concept.

In section 2, we present a systems model for performing a process on a computer. In section 3, we discuss a power consumption model obtained from the experiment. In section 4, we discuss how to allocate each process with a computer to reduce the power consumption. In section 5, we evaluate the algorithms.

2. Computation Model

2.1 Normalized computation rate

A system S is includes a set C of computers c_1, ..., c_n ($n \geq 1$) interconnected in reliable networks. A user issues a request to perform a process like a Web page request.

†1 Seikei University
†2 Rissho University

The process is performed on one computer. There are a set P of application processes p_1, ..., p_m ($m \geq 1$) which can be performed on any computer in C. A term *process* means an *application process* in this paper.

First, a user issues a request to perform a process p_s to a load balancer K. For example, a user issues a request to read a web page on a remote computer. The load balancer K selects one computer c_i in the set C for a process p_s and sends a request to the computer c_i. On receipt of the request, the process p_s is performed on the computer c_i and a reply, e.g. Web page is sent back to the requesting user.

Requests from multiple users are performed on a computer c_i. A process being performed at time t is *current*. A process which already terminates before time t is referred to as *previous*. Let $P_i(t)$ ($\subseteq P$) be a set of current processes on a computer c_i at time t. $N_i(t)$ shows the number of the current processes in the set $P_i(t)$, $N_i(t) = |P_i(t)|$. Let $P(t)$ be $\cup_{i=1,...,n} P_i(t)$.

Suppose a process p_s is performed on a computer c_i. Here, T_{is} is the total computation time of p_s on c_i and $minT_{is}$ shows the computation time T_{is} where a process p_s is exclusively performed on c_i, i.e. without any other process. Hence, $minT_{is} \leq T_{is}$ for every process p_i. Let $maxT_s$ and $minT_s$ be $max(minT_{1s}, ..., minT_{ns})$ and $min(minT_{1s}, ..., minT_{ns})$, respectively. If a process p_s is exclusively performed on the fastest computer c_i and the slowest computer c_j, $minT_s = minT_{is}$ and $maxT_s = minT_{js}$, respectively. A *time unit* (tu) shows the minimum time to perform a smallest process. We assume $1 \leq minT_s \leq maxT_s$.

The average *computation rate* (ACR) F_{is} of a process p_s on a computer c_i is defined as $F_{is} = 1 / T_{is}$ [1/ tu]. Here, $0 < F_{is} \leq 1 / minT_{is} \leq 1$. The maximum ACR $maxF_{is}$ is $1 / minT_{is}$. F_{is} shows how many percentages of the total amount of computation of a process p_s are performed for one time unit. Let $maxF_s$ and $minF_s$ be $max(maxF_{1s}, ..., maxF_{ns})$ and $min(maxF_{1s}, ..., maxF_{ns})$, respectively. $maxF_s$ and $minF_s$ show the maximum ACRs $maxF_{is}$ and $maxF_{js}$ for the fastest computer c_i and the slowest computer c_j, respectively.

The more number of processes are performed on a computer c_i, the longer it takes to perform each of the processes on c_i. Let $\alpha_i(t)$ indicate the *degradation rate* of a computer c_i at time t ($0 \leq \alpha_i(t) \leq 1$)[1/tu]. $\alpha_i(t_1) \leq \alpha_i(t_2) \leq 1$ if $N_i(t_1) \leq N_i(t_2)$ for every pair of different times t_1 and t_2. We assume $\alpha_i(t) = 1$ if $N_i(t) \leq 1$ and $\alpha_i(t) < 1$ if $N_i(t) > 1$. For example, suppose it takes 50 [msec] to exclusively perform a process p_s on a computer c_i. Here, $minT_{is}$ = 50. Here, $F_{is} = maxF_{is}$ = 1/50 [1/msec]. Suppose it takes 75 [msec] to concurrently perform the process p_s with other processes. Here, F_{is} = 1/75 [1/msec]. Hence, $\alpha_i(t)$ = 50/75 = 0.67 [1/msec].

We define the *normalized computation rate* (NCR) $f_{is}(t)$ of a process p_s on a computer c_i at time t as follows:

$$f_{is}(t) = \begin{cases} \alpha_i(t) \cdot maxF_{is}/maxF_s & [1/\mathtt{tu}] \\ \alpha_i(t) \cdot minT_s/minT_{is} & [1/\mathtt{tu}] \end{cases} \quad (1)$$

For the fastest computer c_i, $f_{is}(t)$ = 1 if $\alpha_i(t)$ = 1, i.e. $N_i(t)$ = 1. If a computer c_i is faster than c_j and the process p_s is exclusively performed on c_i and c_j at time t_i and t_j, respectively, $f_{is}(t_i) > f_{js}(t_j)$. If a process p_s is exclusively performed on c_i, $\alpha_{is}(t)$ = 1 and $f_{is}(t) = maxF_{is} / maxF_s$. The maximum NRC $maxf_{is}$ shows $maxF_{is} / maxF_s$. $0 \leq f_{is}(t) \leq maxf_{is} \leq 1$. $f_{is}(t)$ shows how many steps of a process p_s are performed on a computer c_i at time t. The average computation rate (ACR) F_{is} depends on the size of the process p_s while $f_{is}(t)$ depends on the speed of the computer c_i.

Next, suppose that a process p_s is started and terminated on a computer c_i at time st_{is} and et_{is}, respectively. Here, the total computation time T_{is} is et_{is} - st_{is}.

$$\int_{st_{is}}^{et_{is}} (f_{is}(t))\, dt = minT_s \cdot \int_{st_{is}}^{et_{is}} \frac{\alpha_i(t)}{minT_{is}} = minT_s \quad (2)$$

If there is no other process, i.e. $\alpha_i(t)$ = 1 on the computer c_i, $f_{is}(t) = maxF_{is} / maxF_s = minT_s / minT_{is}$. Hence, $T_{is} = et_{is} - st_{is} = minT_{is}$. If other processes are performed, $T_{is} = et_{is}$ - $st_{is} > minT_{is}$. Here, $minT_s$ shows the total amount of computation to be performed by the process p_s.

Figure 1 shows the NCRs $f_{is}(t)$ and $f_{js}(t)$ of a process p_s which are exclusively performed on a pair of computers c_i and c_j, respectively. Here, the computer c_i is the fastest in the computer set C. The NCR $f_{is}(t) = maxf_{is}$ = 1 for $st_{is} \leq t \leq et_{is}$ and $T_{is} = et_{is}$ - $st_{is} = minT_s$. On the slower computer c_j, $f_{js}(t) = maxf_{js} < 1$ and $T_{js} = et_{js}$ - $st_{js} > minT_s$. Here, $maxf_{is} \cdot minT_{is} = minT_s = maxf_{js} \cdot minT_{js}$ from the equation (2). The areas shown by $f_{is}(t)$ and $f_{js}(t)$ have the same size $minT_s$(= T_{is}).

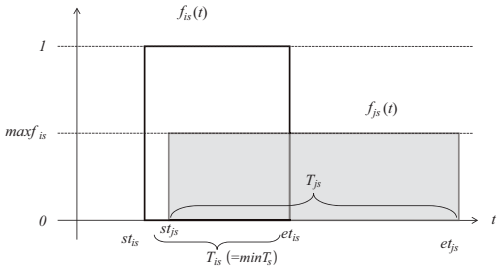

Fig. 1 Normalized computation rates (NCRs).

Next, we define the computation *laxity* $L_{is}(t)$ [tu] as follows:

$$L_{is}(t) = minT_s - \int_{st_{is}}^{t} (f_{is}(x))\, dx. \quad (3)$$

The laxity $L_{is}(t)$ shows how much computation the computer c_i has to spend to perform up a process p_s at time t. $L_{is}(st_{is})$ = $minT_s$ and $L_{is}(et_{is})$ = 0. If the process p_s would be exclusively performed on the computer c_i, the process p_s is expected to terminate at time et_{is} = t + $L_{is}(t)$.

2.2 Simple computation model

There are types of computers with respect to the performance. First, we consider a simple computation model. In the simple computation model, a computer c_i satisfies the following properties:

[Simple computation model]

1. $maxf_{is}$ = $maxf_{iu}$ for every pair of different processes p_s and p_u performed on a computer c_i.

2.
$$\sum_{p_s \in P_i(t)} f_{is}(t) = maxf_i. \quad (4)$$

The maximum NCR $maxf_i$ of a computer c_i is $maxf_{is}$ for any process p_s. This means, the computer c_i is working to perform any process with the maximum clock frequency. $P_i(t)$ shows a set of processes being performed on a computer c_i at time t. In the simple computation model, we assume the degradation factor $\alpha_i(t)$ = 1.

On a computer c_i, each process p_s starts at time st_{is} and terminates at time et_{is}. We would like to discuss how $f_{is}(t)$ of each process p_s changes in presence of multiple precesses on a computer c_i. A process p_s is referred to as *precedes* another process p_u on a computer c_i if $et_{is} < st_{iu}$. A process p_s is *interleaved* with another process p_u on a computer c_i iff $et_{iu} \geq et_{is} \geq st_{iu}$. The interleaving relation is symmetric but not transitive. A process p_s is referred to as *connected* with another process p_u iff (1) p_s is interleaved with p_u or (2) p_s is interleaved with some process p_v and p_v is connected with p_u. The connected relation is symmetric and transitive. A *schedule* sch_i of a computer c_i is a history of processes performed on the computer c_i. Processes in the schedule sch_i are partially ordered in the precedent relation and related in the connected relation. Here, let $K_i(p_s)$ be a closure subset of the processes in the schedule sch_i which are connected with a process p_s, i.e. $K_i(p_s)$ = $\{p_u \mid p_u$ is connected with $p_s\}$. $K_i(p_s)$ is an equivalent class with the connected relation, i.e. $K_i(p_s)$ = $K_i(p_u)$ for every process p_u in $K_i(p_s)$. $K_i(p_s)$ is a *knot* in the schedule sch_i. The schedule sch_i is divided into knots $K_{i1}, \ldots, K_{il_i}$ which are pairwise disjointing. Let p_u and p_v are a pair of processes in a knot $K_i(p_s)$ where the starting time st_{iu} is the minimum and the termination time et_{iv} is the maximum. That is, the process p_u is first performed and the process p_v is lastly finished in $K_i(p_s)$. The execution time TK_i of the knot $K_i(p_s)$ is et_{iv} - st_{iu}. Let $KP_i(t)$ be a *current knot* which is a set of current or previous processes which are connected with at least one current process in $P_i(t)$ at time t.

[Theorem] Let K_i be a knot in a schedule sch_i of a computer c_i. The execution time TK_i of the knot K_i is $\sum_{p_s \in K_i} minT_{is}$.

Let us consider a knot K_i of three processes p_1, p_2, and p_3 on a computer c_i as shown in Figure 2 (1). Here, K_i = $\{p_1, p_2, p_3\}$. First, suppose that the processes p_1, p_2, and p_3 are serially performed, i.e. et_{i1} = st_{i2} and et_{i2} = st_{i3}. Here, the execution time TK_i is et_{i3} - st_{i1} = $minT_{i1}$ + $minT_{i2}$ + $minT_{i3}$. Next, three processes p_1, p_2, and p_3 start at time st and terminate at time et as shown in Figure 2 (2). Here, the execution time TK_i = $minT_{i1}$ + $minT_{i2}$ + $minT_{i3}$. Lastly, let us consider a knot K_i where the processes are concurrently performed. The processes p_1, p_2, and p_3 start at the same time, st_{i1} = st_{i2} = st_{i3}, are concurrently performed, and the process p_3 lastly terminates at time et_{i3} after p_1 and p_2 as shown in Figure 2 (3). Here, the execution time TK_i of the knot K_i is

et_{i3} - st_{i1} = $minT_{i1}$ + $minT_{i2}$ + $minT_{i3}$. The current knot $KP_i(t_1)$ is $\{p_1, p_2, p_3\}$ and $KP_i(t_2)$ is $\{p_1, p_2\}$.

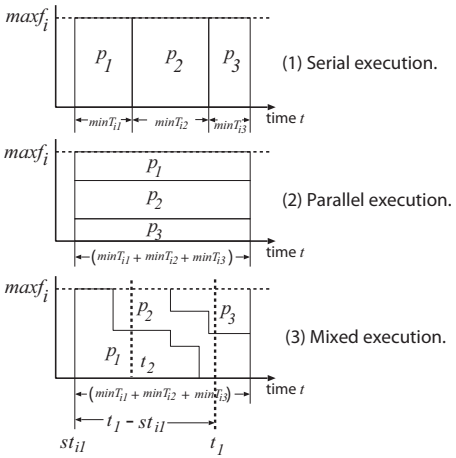

Fig. 2 Execution time of knot.

It depends on the scheduling algorithm how much each NCR $f_{is}(t)$ is in the equation (4), $f_{is}(t) = \alpha_{is} \cdot maxf_i$ where $\sum_{p_s \in P_i(t)} \alpha_{is} = 1$. In the fair scheduler, each $f_{is}(t)$ is the same as the others, i.e. $\alpha_{is} = 1/|P_i(t)|$:

$$f_{is}(t) = maxf_i \ / \ |P_i(t)|. \tag{5}$$

2.3 Estimated termination time

Suppose there are a set P of processes $\{p_1, \ldots, p_m\}$ and a set C of computers $\{c_1, \ldots, c_n\}$ in a system S. Here, we assume the system S to be heterogeneous, i.e. some pair of computers c_i and c_j have different specifications and performance. Suppose a process p_s is started on a computer c_i at time st_{is}. A set $P_i(t)$ of current processes are being performed on a computer c_i at time t.

[Computation model] Let $KP_i(t)$ be a current knot = $\{p_{i1}, ..., p_{il_i}\}$ of processes, where the starting time is st. The total execution time $T(st, t)$ of processes in the current knot $KP_i(t)$ is given as;

$$T(st, t) = minT_{i1} + minT_{i2} + \cdots + minT_{il_i} \tag{6}$$

In Figure 2 (3), t_1 shows the current time. A process p_1 is first initiated at time st_{i1} and is terminated before time t_1 on a computer c_i. A pair of processes p_2 and p_3 are currently performed at time t_1. Here, $KP_i(t)$ is a current knot $\{p_1, p_2, p_3\}$ at time t_1. $T(st_{i1}, t_1) = minT_{i1} + minT_{i2} + minT_{i3}$. The execution time from time st_{i1} to t_1 is t_1 - st_{i1}. At time t_1, we can estimate that the concurrent processes p_2 and p_3 are performed and terminate at the time $t_1 + T(st_{t1}, t_1) - (t_1 - st_{i1}) = st_{i1} + T(t_{i1}, t_1)$. st_{i1} is *starting time* of the current knot $KP_i(t)$.

The *estimated termination time* $ET_i(t)$ of current processes on a computer c_i means time when every current process of time t terminates if no other process would be performed after time t. $ET_i(t)$ is given as follows:

$$ET_i(t) = t + T(st_{is}, t) - (t - st_{is}) = st_{is} + T(st_{is}, t) \tag{7}$$

Suppose a new process p_s is started at current time t. By using the equation (7), we can obtain $ET_i(t)$ of the current processes on each computer c_i at time t. From the computation model, $ET_{is}(t)$ of a new process p_s starting on a computer c_i at time t is given as follows [Figure 3]:

$$ET_{is}(t) = ET_i(t) + minT_{is} \tag{8}$$

3. Power Consumption Models

3.1 Simple model

In this paper, we assume the simple computation model is taken for each computer, i.e. the maximum clock frequency to be stable for each computer c_i. Let $E_i(t)$ show the electric power consumption of a computer c_i at time t [W/tu] ($i = 1, \ldots, n$). $maxE_i$ and $minE_i$ indicate the maximum and minimum electric power consumption of a computer c_i, respectively. That is, $minE_i \leq E_i(t) \leq maxE_i$. $maxE$ and $minE$ show

max($maxE_1$, ..., $maxE_n$) and min($minE_1$, ..., $minE_n$), respectively. Here, $minE_i$ shows the power consumption of a computer c_i which is in idle state.

We define the *normalized power consumption rate* ($NPCR$) $e_i(t)$ [1/tu] of a computer c_i at time t as follows:

$$e_i(t) = E_i(t)/maxE \ (\leq 1). \tag{9}$$

Let $mine_i$ and $maxe_i$ show the maximum power consumption rate $minE_i$ / $maxE$ and the minimum one $maxE_i$ / $maxE$ of the computer c_i, respectively. If the fastest computer c_i maximumly spends the electric power with the maximum clock frequency, $e_i(t) = maxe_i = 1$. In the lower-speed computer c_j, i.e. $maxf_j < maxf_i$, $e_j(t) = maxe_j < 1$.

We propose two types of power consumption models for a computer c_i, *simple* and *multi-level* models. In the simple model, $e_i(t)$ is given depending on how many number of processes are performed as follows:

$$e_i(t) = \begin{cases} maxe_i & \text{if} \quad N_i(t) \geq 1. \\ mine_i & \text{if} \quad \text{otherwise.} \end{cases} \tag{10}$$

A personal computer with one CPU satisfies the simple model as discussed in the experiments of the succeeding section.

3.2 Multi-level model

In the multi-level model, the electric power consumption of a computer c_i depends on how many number of processes are concurrently performed on the computer c_i. The NPCR $e_i(t)$ of a computer c_i at time t is given as follows:

$$e_i(t) = \begin{cases} \beta_i(t) \cdot maxe_i & \text{if} \quad N_i(t) \geq 1. \\ mine_i & \text{if} \quad \text{otherwise.} \end{cases} \tag{11}$$

Here, the factor $\beta_i(t)$ shows how much the power consumption is degraded depending on the number of processes being concurrently performed on the computer c_i at time t, i.e. $mine_i$ / $maxe_i \leq \beta_i(t) \leq 1$ if $N_i(t) \geq 1$. $\beta_i(t_1) < \beta_i(t_2)$ if $N_i(t_1) < N_i(t_2)$.

In a computer c_i with multiple CPUs, the NPCR $e_i(t)$ depends on how many number of CPUs are active independently of the number of cores in each CPU. The NCPR $e_i(t)$ depends on the number of active CPUs. The number of active CPUs depends on the scheduling algorithm to allocate processes to CPUs.

3.3 Total power consumption

The total normalized power consumption $TPC_i(t_1, t_2)$ of a computer c_i from time t_1 to time t_2 is given as follows:

$$TPC_i(t_1, t_2) = \int_{t_1}^{t_2} e_i(t)dt \tag{12}$$

Next, $TPC_1 \cdot (t_1, t_2) \leq t_2$ - t_1. In the fastest computer c_i, $TPC_1 \cdot (t_1, t_2) = maxe_i \cdot (t_2 - t_1) = t_2 - t_1$ if at least one process is performed at any time from t_1 to t_2 in the simple model.

Let K_i be a knot of a computer c_i whose starting time is st_i and termination time is et_i. The normalized total power consumption of the computer c_i to perform every process in the knot K_i is $TPC_i(st_i, et_i)$. In the simple model, $TPC_i(st_i, et_i) = \int_{st_i}^{et_i} maxe_i dt = (et_i - st_i) \cdot maxe_i = \sum_{p_s \in K_i} minT_{is} \cdot maxe_i$.

4. Process Allocation Algorithms

4.1 Round-robin algorithms

We consider two types of the round-robin algorithms, weighted round robin (WRR) [20] and weighted least connection (WLC) [21] algorithms. For each of the WRR and WLC algorithms, we consider two cases, *Per* (performance) and *Pow* (power). In *Per* the weight is given in terms of the performance ratio of the servers. That is, the higher performance a server supports, the more number of processes are allocated to the server. In *Pow*, the weight is defined in terms of the power consumption ratio of the servers. The less power a server consumes, the more number of processes are allocated to the server.

4.2 Laxity-based algorithm

Some application has the deadline constraint TC_s on a process p_s issued by the application, i.e. a process p_s has to terminate until the deadline. Here, a process p_s has to be allocated to a computer c_i so that the process p_s can terminate by the deadline TC_s. $\mathbf{C}_s(t)$ denotes a set of computers which satisfy TC_s, i.e. $\mathbf{C}_s(t) = \{c_i \mid ET_{is}(t) \leq TC_s\}$. That is, in a computer c_i in $\mathbf{C}_s(t)$, the process p_s is expected to terminate by TC_s. Here, if the process p_s is allocated to one computer c_i in $\mathbf{C}_s(t)$, the process p_s can terminate

before TC_s.

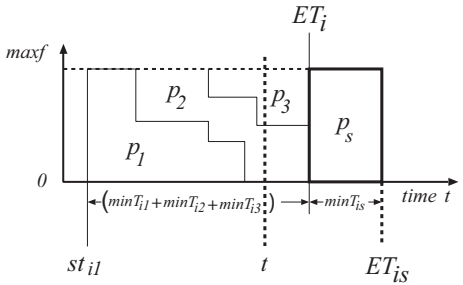

Fig. 3 Expected termination time ET_{is}.

Next, we assume that the NPCR $e_i(t)$ of each computer c_i is given as equation (10) according to the simple model. We can estimate the total power consumption laxity $le_{is}(t)$ of a process p_s between time t and $ET_{is}(t)$ at time t when the process p_s is allocated to the computer c_i [Figure 4]. $le_{is}(t)$ of the computer c_i is given as equation (13):

$$le_{is}(t) = maxe_i * (ET_{is}(t) - t) \tag{13}$$

Suppose a process p_s is issued at time t. A computer c_i in the computer set C is selected for the a process p_s with the constraint TC_s at time t as follows:

Alloc(t, C, p_s, TC_s) {
  $\mathbf{C_s}$ = ϕ; $\mathbf{NoC_s}$ = ϕ;
  for each computer c_i **in** C, {
    if $ET_{is}(t) \leq TC_s$, $\mathbf{C_s}$ = $\mathbf{C_s} \cup \{c_i\}$;
    else /* $ET_{is}(t) > TC_s$ */ $\mathbf{NoC_s}$ = $\mathbf{NoC_s} \cup \{c_i\}$; }
  if $\mathbf{C_s} \neq \phi$, { /* candidate computers are found */
    $computer$ = c_i such that $le_{is}(t)$ is the minimum in $\mathbf{C_s}$;
    return($computer$); }
  else { /* $\mathbf{C_s}$ = ϕ */
    $computer$ = c_i such that $ET_{is}(t)$ is minimum in $\mathbf{NoC_s}$;
    return($computer$);
  }
}

$\mathbf{C_s}$ and $\mathbf{NoC_s}$ are sets of computers which can and cannot satisfy the constraint TC_s, respectively. Here, $\mathbf{C_s} \cup \mathbf{NoC_s} = C$ and $\mathbf{C_s} \cap \mathbf{NoC_s} = \phi$.

In the procedure **Alloc**, if there is at least one computer which can satisfy the time constraint TC_s of process p_s, one of the computers which consumes the minimum power consumption is selected. If there is no computer which can satisfy the application time constraint TC_s, one of the computers which can most early terminate the process p_s is selected in the computer set C.

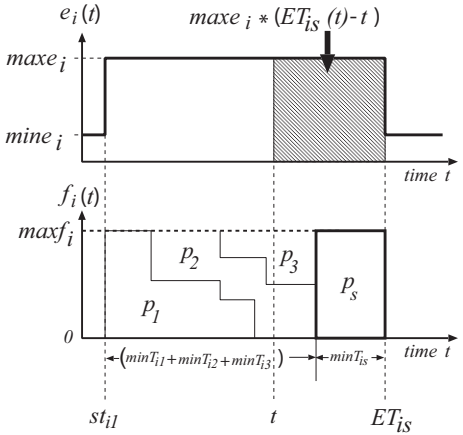

Fig. 4 Estimation of power consumption.

5. Evaluation

5.1 Environment

We measure how much electric power computers consume for Web applications. We

consider a cluster system composed of Linux Virtual Server (LVS) systems which are interconnected in gigabit networks as shown in Figure 5. The NAT based routing system VS-NAT[12] is used as the load balancer K. The cluster system includes three servers s_1, s_2, and s_3 in each of which Apache 2.0[11] is installed, as shown in Table 1. The load generator server L first issues requests to the load balancer K. Then, the load balancer K assigns each request to one of the servers according to some allocation algorithm. Each server s_i compresses the reply file by using the Deflate module[13] on receipt of a request from the load generator server L.

We measure the peak consumption of electric power and the average response time of each server s_i (i = 1, 2, 3). The power consumption ratio of the servers s_1, s_2, and s_3 is 0.9 : 0.6 : 1 as shown in Table 1. On receipt of a Web request, each server s_i finds a reply file of the request and compresses the reply file by using the Deflate module. The size of the original reply file is 1Mbyte and the compressed reply file is 7.8Kbyte in size. The Apache benchmark software[10] is used to generate Web requests, where the total number 10,000 of requests are issued where 100 requests are concurrently issued to each server. Here, the performance ratio of the servers s_1, s_2, and s_3 are 1 : 1.2 : 4 as shown in Table 1. Thus, s_3 is the fastest and mostly consumes the electric power. The server s_1 is slower than s_3 but more consumes the electric power than s_2.

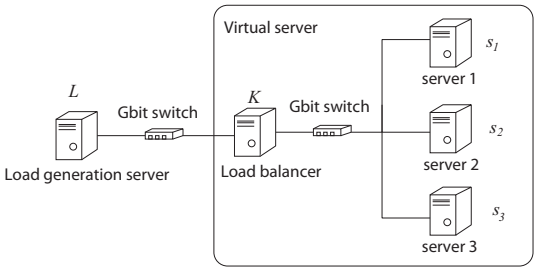

Fig. 5 Cluster system.

5.2 Experimental results

If the weight is based on the performance ratio (Per), the requests are allocated to the servers s_1, s_2, and s_3 with the ratio 1 : 1.2 : 4, respectively. On the other hand, if the weight is based on the power consumption ratio (Pow), the requests are allocated to the servers s_1, s_2, and s_3 with the ratio 0.9 : 0.6 : 1, respectively. Here, by using the Apache benchmark software, the load generation server L transmits totally 100,000 requests to the servers s_1, s_2, and s_3 where six requests are concurrently issued to the load balancer K. The total power consumption of the cluster system and the average response time of a request from a web server are measured. We consider a static web server where the size of a reply file for a request is not dynamically changed, i.e. the compressed version of the same HTML reply file is sent back to each user. In this experiment, the original HTML file and the compressed file are 1,025,027 [Byte] and 13,698 [Byte] in size, respectively. On the load balancer K, types of process allocation algorithms are adopted; the weighted round-robin (WRR)[20] algorithms, WRR-Per and WRR-Pow; the weighted least connection (WLC)[21] algorithms, WLC-Per and WLC-Pow.

Figure 6 shows the total power consumption [W/H] of the cluster system for time. In WRR-Per and WLC-Per, the total execution time and peak power consumption are almost the same. In addition, the total execution time and peak power consumption are almost the same in WRR-Pow and WLC-Pow. This experimental result shows that the total power consumption and total execution time are almost the same for the two allocation algorithms if the same weight ratio is used. Here, if the weight of the load balance algorithm is given in terms of the performance ratio (*Per*), the peak power consumption is higher than *Pow*. However, the total execution time of *Per* is longer than *Pow*.

Here, the total power consumption is calculated by the multiplication of the execution time and power consumption. The experiment shows the total power consumption is reduced by using the performance based weight (Per).

6. Concluding Remarks

In this paper, we discussed the simple and multi-level power consumption models of computers. The simple model shows a computer with one CPU while the multi-level model denotes a computer with multiple CPUs. We discussed the laxity-based algorithm to allocate a process to a computer so that the deadline constraint is satisfied and the total power consumption is reduced on the basis of the laxity concept. We obtained experimental results on electric power consumption of Web servers. We evaluated the simple model through the experiment of the PC cluster. Then, we showed the PC cluster

Table 1 Servers

	Server 1	Server 2	Server 3
Number of CPUs	1	1	2
Number of cores	1	1	2
CPU	Intel Pentium 4 (2.8GHz)	AMD Athlon 1648B (2.7GHz)	AMD Opteron 2216HE (2.4GHz)
Memory	1,024MB	4,096MB	4096MB
Maximum computation rate	1	1.2	4
Maximum power consumption rate $maxe_i$	0.9	0.6	1

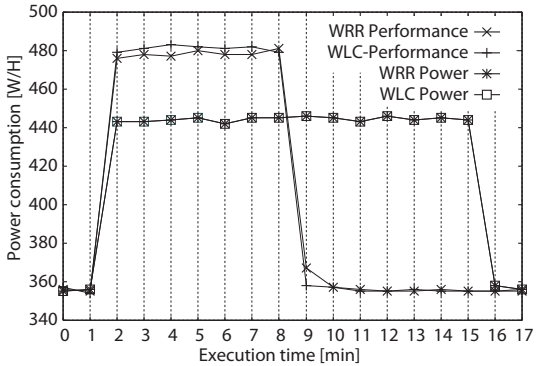

Fig. 6 Power consumption.

follows the simple model. We are now considering types of applications like database transactions and measuring the power consumption of multi-CPU servers.

References

1) I. F. Akyildiz and I. H. Kasimoglu.: Wireless Sensor and Actor Networks: Research Challenges. *Ad Hoc Networks journal (Elsevier)*, 2:351.367, 2004.
2) AMD, *http://www.amd.com/*.
3) Intel, *http://www.intel.com/*.
4) Green IT, *http://www.greenit.net*.
5) T. Heath, B. Diniz, E. V. Carrera, W. Meira, R. Bianchini.: Energy Conservation in Heterogeneous Server Clusters. *In PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*(2005), pp. 186-195.
6) Nancy A. Lynch, Distributed Algorithms. *Morgan Kaufmann Publisher*, 1st edition (April 1997), ISBN-10: 1558603484.
7) A. Montresor.: A robust Protocol for Building Superpeer overlay Topologies. *In: Proc. of the 4th International Conference on Peer-to-Peer Computing*, pp. 202-209, 2004.
8) R. Bianchini and R. Rajamony.: Power and Energy Management for Server Systems. *IEEE Computer, volume 37, number 11, November 2004. Special issue on Internet data centers*.
9) K. Rajamani and C. Lefurgy.: On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. *In: Proc of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 111-122, 2003.
10) ab - Apache HTTP server benchmarking tool, *http://httpd.apache.org/docs/2.0/programs/ab.html*.
11) Apache 2.0, *http://httpd.apache.org/*.
12) VS-NAT, *http://www.linuxvirtualserver.org/*.
13) Apache Module mod-deflate, *http://httpd.apache.org*.
14) M. Aron, P. Druschel, and W. Zwaenepoel. Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers. *In Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pp. 90-101, 2000.
15) A. Bevilacqua. A Dynamic Load Balancing Metthod on a Heterogeneous Cluster of Workstations. *Informatica, 23(1): 49-56, March 1999*.
16) R. Bianchini and E. V. Carrera. Analytical and Experimental Evaluation of Cluster-Based WWW Servers. *World Wide Web journa, 3(4), Decembar 2000*.
17) T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Self-Configuring Heterogeneous Server Clusters. *In Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, 2003.
18) K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 111-122, 2003.
19) M. Colajanni, V. Cardellini, and P. S. Yu. Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers. *In Proceeding of the 18th International Conference on Distributed Computing Systems*, pp. 295, 1998.
20) Weighted Round Robin (WRR), *http://www.linuxvirtualserver.org/docs/scheduling.html*.
21) Weighted Least Connection (WLC), *http://www.linuxvirtualserver.org/docs/scheduling.html*.