Computation and Transmission Rate Based Algorithm for Reducing the Total Power Consumption

Tomoya Enokido*
Rissho University
Tokyo, Japan
eno@ris.ac.jp

Ailixier Aikebaier†
Seikei University
Tokyo, Japan
alisher.akber@computer.org

Makoto Takizawa‡
Seikei University
Tokyo, Japan
makoto.takizawa@computer.org

Abstract

In information systems, it is critical to reduce the total electrical power consumption of computers and networks in order to realize the digital ecosystems and the green IT technologies. The extended power consumption laxity-based (EPCLB) algorithm is proposed to select a server in a set of servers so as to not only satisfy deadline constraint but also reduce the total power consumption of servers in general applications. However, each time a load balancer receives a new request, the load balancer has to collect status of each server and calculate the estimated power consumption for the request in the EPCLB algorithm. Hence, the computation and communication overhead to estimate the power consumption is large on the load balancer if the number of clients is increased. In addition, the status of each server might be changed during the estimation process. Then, it is difficult to correctly estimate the power consumption. In this paper, we newly propose an algorithm to select a server in a set of servers so that the total power consumption of servers and the overhead of a load balancer can be reduced. We evaluate the algorithm in terms of the power consumption of servers and the overhead of a load balancer compared with the EPCLB and traditional round-robin (RR) algorithms.

1 Introduction

Information systems are getting scalable so that various types of computational devices like server computers and sensor nodes are interconnected in types of networks like wireless and wired networks. In order to realize digital ecosystems [1, 2] and *green IT* technologies [3, 4], the total electric power consumption of computers and networks have to be reduced. In wireless sensor networks [5] and ad hoc networks [6, 7], routing algorithms to reduce the power consumption of battery in each node are discussed. There are various kinds of load balancing [8, 9] and resource management [10] algorithms are proposed in cluster systems. In addition, various types algorithms to find required number of servers in homogeneous and heterogeneous servers are discussed [11, 12, 13]. Biancini *et al*. [12, 13] discussed how to reduce the power consumption of a cluster of homogeneous servers by turning off servers which are not required for executing a collection of web requests. However, it is difficult to dynamically turn on and off servers in some types of systems like mission critical systems. In addition, the computer cannot be turned off by other persons different from the owners in some types of computation models like peer-to-peer (P2P) and grid models. In this paper, we does not consider to turn off servers to reduce the total power consumption of servers.

There are various kinds of applications on distributed systems like Web applications [14] and Google file systems [15]. In the papers, [16, 17, 18, 19, 20], computation-based and communication-based applications are considered to discuss the power consumption. In the computation-based applications, a server mainly consumes CPU resources to process the request. On the other hand, a server transmits a

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 2, number: 2, pp. 1-18

*Did all the difficult work

†Did numerous tests and provided a lot of suggestions

‡Masterminded

large volume of data to a client like file transfer protocol (FTP) applications in the communication-based applications. The *simple* and *multi-level* models [16, 17] for computation and power consumption of a server in computation-based applications and the power consumption model [18, 19] in communication-based applications are discussed by abstracting properties which dominate the power consumption of the server based on experimental results of real computers. The PCLB (*power consumption laxity based*) algorithm is proposed to select a server in a set of servers so as to not only satisfy deadline constraint but also reduce the total power consumption of servers in computation-based applications. On the other hand, the PCB (*power consumption-based*) algorithm [18, 19] and EPCB (*extended* PCB) algorithm [20] are proposed to select a server in a set of possible servers so that the total power consumption can be reduced in communication-based applications. In reality, most applications are composed of both the computation and communication modules. In the paper [21], an integrated model for showing the computation and power consumption of general applications is proposed. The EPCLB (*extended power consumption laxity-based*) algorithm is proposed to select one of servers so that not only the total power consumption of servers can be reduced but also the deadline constraint holds.

However, in the EPCLB algorithm, a load balancer has to collect status of each server and calculate the estimated power consumption each time the load balancer receives a new request. If the number of clients concurrently performed is increased, the computation and communication overhead to estimate the power consumption of servers is increased on the load balancer. Then, the performance of the load balancer is decreased and the load balancer might be bottleneck of the system. In addition, the status of each server might be changed during the estimation process. In reality, it is difficult to correctly estimate the power consumption of servers in the EPCLB algorithm.

In this paper, we newly propose the CTRB (*computation and transmission rate based*) algorithm to select a server in a set of servers so that the total power consumption of the servers and the overhead of a load balancer can be reduced. We evaluate the CTRB algorithm in terms of the power consumption of servers and the overhead of a load balancer compared with the traditional round-robin (RR) and the EPCLB algorithms. We show the CTRB algorithm can reduce the power consumption of servers than the RR algorithm. In addition, we show the CTRB algorithm can reduce the overhead of a load balancer than the EPCLB algorithm.

In section 2, we show the experimental results of total power consumption of computers. In section 3, we discuss models for computation and transmission. In section 4, we define the power consumption model. In section 5, we discuss the CTRB algorithm to reduce the total power consumption of servers and the overhead of a load balancer based on the power consumption. In section 6, we evaluate the CTRB algorithm compared with the EPCLB and RR algorithms.

2 Experimentations

2.1 Experimental environment

We consider a general type of application which is composed of both the computation and communication modules like Web applications. As shown in Figure 1, a server s_1 with data d of 544 [MB] is interconnected with three clients c_1, c_2, and c_3 in one-Gbps networks. Table 1 shows the specification of the server s_1. On receipt of the request from a client c_s, the server s_1 compresses the data d to a file f of 530 [MB] by using the Deflate function [22] and then transmits the file f to a client c_s.

2.2 Concurrent execution of requests

First, we measure how much electric power the server s_1 consumes to perform the following five types of experimentations by using the power meter Watts up? .Net [23] as shown in Figure 1.

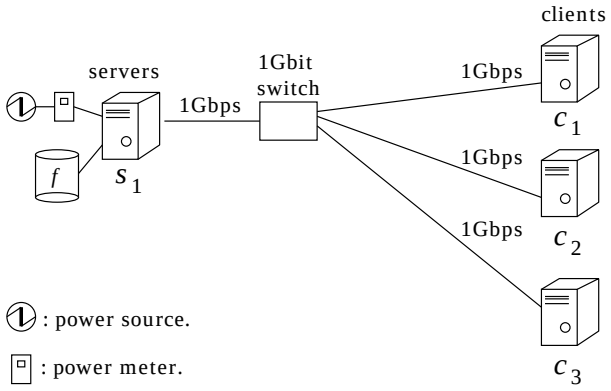


Figure 1: Experimental environment.

Table 1: Server s_1

Server	s_1
Number of CPUs and cores / CPU	1 CPU and 1 core / CPU
CPU	AMD Athlon 1648B (2.7GHz)
Memory	4,096MB
NIC	Broadcom Gbit Ether (1Gbps)

E1. A client c_1 issues a request to the server s_1.

E2. A client c_2 issues a request to the server s_1.

E3. A client c_3 issues a request to the server s_1.

E4. A pair of clients c_1 and c_2 concurrently issue requests to the server s_1.

E5. Three clients c_1, c_2, and c_3 concurrently issue requests to the server s_1.

Table 2 summarizes the computation time for compressing data, transmission time, and power consumption rate for each experimentation. It takes 46, 44, and 45 [sec] to compress the data d in the experimentations E1, E2, and E3, respectively. A pair of the compression time in the experimentations E4 and E5 are 93 and 146 [sec], respectively. It takes 2.1 and 3.2 times longer time to compress the data d in the experimentations E4 and E5, respectively, than E1, E2, and E3. Figure 2 shows the power consumption rate [W/sec] for elapse time [sec]. The server s_1 consumes the electric power to compress the data d at rates 120, 119, 119, 120, and 121 [W/sec] in the experimentations E1, E2, E3, E4, and E5, respectively. The experimentations imply the simple computation model [16, 17] holds for compressing the data. That is, any process is performed on a server with the maximum clock frequency.

Next, we would like to discuss the transmission process. In the experimentations E1, E2, and E3, the average transmission rates with the clients c_1, c_2, and c_3 are 241, 140, and 124 [Mbps], respectively. The average power consumption rates during transmission in the experimentations E1, E2, and E3 are 105, 98, and 97 [W/sec], respectively. In the experimentation E4, the server s_1 transmits the file f to a pair of clients c_1 and c_2 after compressing the data d. Then, the server s_1 ends the transmission of the file f to the client c_1 before transmitting to the client c_2. Here, the average transmission rates for the clients c_1 and c_2 are 234 and 131 [Mbps], respectively. The total transmission rate at which the server s_1 concurrently transmits the file f to the clients c_1 and c_2 is 365 (= 234 + 131) [Mbps]. Here, the average

Table 2: Concurrent execution of requests.

Exp.	Compression time [sec]	Power consumption rate of compression [W/sec]	Transmission rate [Mbps]	Power consumption rate of transmission [W/sec]	Transmission time [sec]
E1	46	120	241	105	18
E2	44	119	140	98	30
E3	45	119	124	97	34
E4	93	120	234 (c_1) 131 (c_2)	112 (c_1 and c_2) 99 (c_2)	18 (c_1) 33 (c_2)
E5	146	121	219 (c_1) 137 (c_2) 61 (c_3)	116 (c_1, c_2, and c_3) 106 (c_2 and c_3) 95 (c_3)	19 (c_1) 31 (c_2) 69 (c_3)

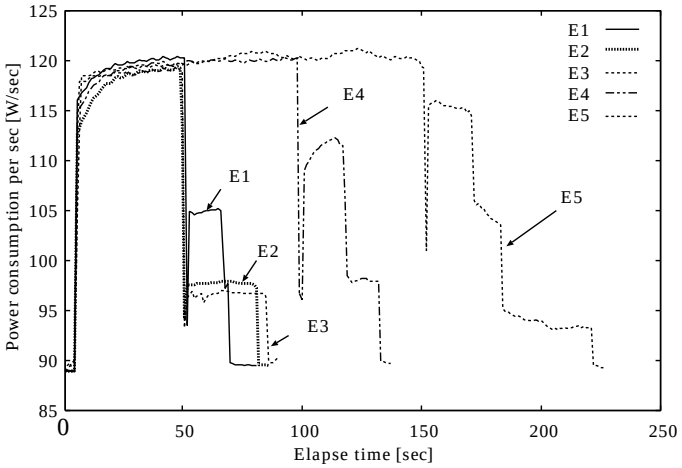


Figure 2: Concurrent execution of requests.

power consumption rate is 112 [W/sec]. The average power consumption rate is 99 [W/sec] where the server s_1 transmits to the client c_2. In the experimentation E5, the server s_1 transmits the file f to three clients c_1, c_2, and c_3 after compression. Then, the server s_1 ends the transmission to the client c_1 before the other transmissions. Next, the transmission for the client c_2 is ended before the client c_3. Here, the average transmission rates for the clients c_1, c_2, and c_3 are 219, 137, and 61 [Mbps], respectively. Here, the total transmission rate is 417 (= 219 + 137 + 61) [Mbps]. This means that the maximum transmission rate of the server s_1 is 417[Mbps]. The transmission rate for the client c_3 (61 [Mbps]) is 49% degraded than the transmission rate (124 [Mbps]) in the experimentation E3. The total rate of the maximum receipt rates of three clients is larger than the maximum transmission rate (417 [Mbps]) of the server s_1. Hence, the transmission rate for the client c_3 is degraded.

The average power consumption rates of the server s_1 to transmit to three clients c_1, c_2, and c_3 at 417 [Mbps] and to a pair of clients c_2 and c_3 at 198 [Mbps] are 116 and 106 [W/sec], respectively. The average power consumption rate to the client c_3 at 61 [Mbps] is 95 [W/sec]. The power consumption rate depends on the transmission rate [18, 19, 20]. In addition, the maximum power consumption rate for transmitting data is smaller than compressing the data.

2.3 Concurrent execution of compression and transmission

We consider how much amount of power a server s_1 consumes to concurrently perform computation and transmission processes. The following three types of experimentations are performed concurrently with a computation process to compress data:

CT1. The server s_1 transmits a file f to only a client c_1.

CT2. The server s_1 concurrently transmits a file f to clients c_1 and c_2.

CT3. The server s_1 concurrently transmits a file f to clients c_1, c_2, and c_3.

Figure 3 shows the power consumption rate of a server s_1 for elapse time. Table 3 summarizes time for computation and transmission and power consumption. In the experimentations CT1, CT2, and CT3, the transmission time is shorter than the compression time. The power consumption rates of the server s_1 in the experimentations CT1, CT2, and CT3 are 120, 121, and 122 [W/sec], respectively. This means that even if the server s_1 transmits the file f concurrently to multiple clients, the server s_1 consumes the maximum power consumption as long as at least one compression process is performed. On the other hand, the more number of concurrent processes, the longer compression time. For example, it takes 46, 55, and 62 [sec] to compress the data d in the experimentations CT1, CT2, and CT3, respectively. This means it takes 1.2 times longer time to compress the data d if one more transmission process is additionally executed. It takes similar time to transmit the file f to each of clients c_1, c_2, and c_3 in the experimentations CT1, CT2, and CT3 as presented in the preceding subsection. Hence, even if compression processes are performed concurrently with a transmission process, it takes almost the same time to transmit data.

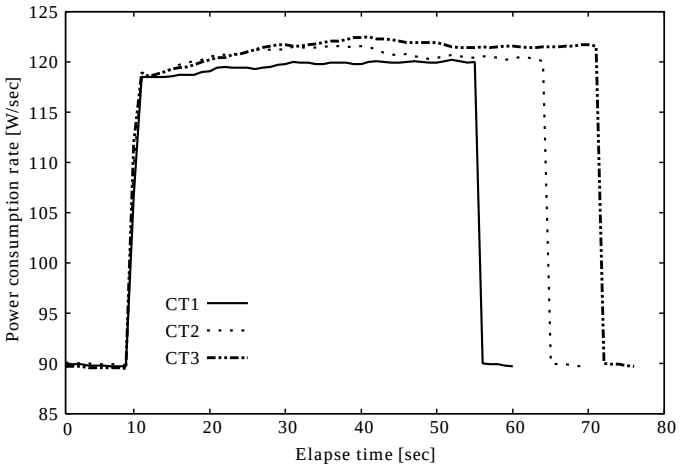


Figure 3: Power consumption rates for concurrent executions of compressions and transmissions.

3 System Model

3.1 Communication Model

Let S be a set of multiple servers s_1, ..., s_n ($n \geq 1$) each of which holds a full replica of data d. Let C be a set of clients c_1, ..., c_m ($m \geq 1$) which issue requests to servers in the server set S to obtain the data. A client c_s first issues a request to a load balancer K. The load balancer K selects one server s_i in the set S

Table 3: Concurrent execution of compressions and transmissions.

Exp.	Compression time [sec]	Transmission rate [Mbps]	Transmission time [sec]	Power consumption rate [W/sec]
CT1	46	240	18	120
CT2	55	217 (c_1) 134 (c_2)	20 32	121
CT3	63	212 (c_1) 134 (c_2) 71 (c_3)	20 32 60	122

and forwards the the request to the server s_t. On receipt of a request, the server s_t performs the request process p_i and manipulates the data d and then transmits a reply file f to the client c_s. A process p_i is thus composed of a pair of a *computation* subprocess cp_i where CPU resources are mainly used and a *transmission* subprocess tp_i where a reply file is transmitted, i.e. $p_i = \langle cp_i, tp_i \rangle$. A term *process* means an *application process* for a request in this paper. A process being performed is *current* at time τ. A process which already terminates before time τ is *previous* at time τ.

3.2 Computation Model

$P_t(\tau)$ shows a set of current processes on a server s_t at time τ. $P(\tau)$ is $\cup_{t=1,...,n} P_t(\tau)$. $NP_t(\tau) = |P_t(\tau)|$. Let $CP_t(\tau)$ and $CT_t(\tau)$ be sets of current computation and transmission processes on a server s_t at time τ, respectively. $NC_t(\tau) = |CP_t(\tau)|$ and $NT_t(\tau) = |CT_t(\tau)|$. $|P_t(\tau)| = |CP_t(\tau)| + |CT_t(\tau)|$. For $CP_t(\tau) = \{cp_i\}$ and $P_t(\tau) = \{p_i\}$, $NC_t(\tau) = 1$ and $NT_t(\tau) = 0$. Here, a computation process cp_i is *exclusively* performed on a server s_t at time τ. If $cp_i \in CP_t(\tau)$ and ($NC_t(\tau) > 1$ or $NT_t(\tau) > 0$), cp_i is *interleaved* with another process $p_j \in P_t(\tau)$ ($i \neq j$) on the server s_t at time τ.

Suppose a computation process cp_i is performed on a server s_t. Here, we consider the following parameters:

- T_{ti} = total computation time of a computation process cp_i on a server s_t.
- $minT_{ti}$ = minimum computation time of a computation process cp_i where cp_i is exclusively performed on a server s_t ($1 \leq minT_{ti} \leq T_{ti}$).
- $maxT_i = max(minT_{1i}, ..., minT_{ni})$.
- $minT_i = min(minT_{1i}, ..., minT_{ni})$.
- $F_{ti} = 1 / T_{ti}$ = *average computation rate* (*ACR*) of a computation process cp_i on a server s_t ($0 < F_{ti} \leq 1 / minT_{ti} \leq 1$) [1/tu].
- $maxF_{ti} = 1 / minT_{ti}$ = maximum *ACR* of a computation process cp_i on a server s_t.
- $maxF_i = max(maxF_{1i}, ..., maxF_{ni})$.
- $minF_i = min(maxF_{1i}, ..., maxF_{ni})$.

If a computation process cp_i is exclusively performed on the fastest server s_t and the slowest server s_u, $minT_i = minT_{ti}$ and $maxT_i = minT_{ui}$, respectively. We assume it takes at least one time unit [*tu*] to perform a computation process on any server. F_{ti} shows how many percentages of the total amount of computation of a computation process cp_i are performed for one time unit.

The more number of processes are performed, the longer it takes to perform each of the processes on a server s_t. Let $\alpha_t(\tau)$ be the *computation degradation rate* of a server s_t at time τ ($0 \leq \alpha_t(\tau) \leq 1$) [1/tu]. $\alpha_t(\tau_1) \leq \alpha_t(\tau_2) \leq 1$ if $NC_t(\tau_1) \leq NC_t(\tau_2)$. $\alpha_t(\tau) = 1$ if $NC_t(\tau) \leq 1$. In this paper, $\alpha_t(\tau)$ is assumed to be $\varepsilon_t^{NC_t(\tau)-1}$ where $0 \leq \varepsilon_t \leq 1$. Let $\theta_t(\tau)$ indicate the *transmission-computation degradation rate* of s_t when multiple transmission processes are concurrently performed ($0 \leq \theta_t(\tau) \leq 1$) [1/tu]. $\theta_t(\tau_1) \leq \theta_t(\tau_2) \leq 1$ if $NT_t(\tau_1) \leq NT_t(\tau_2)$. $\theta_t(\tau) = 1$ if $NT_t(\tau) \leq 1$. In this paper, $\theta_t(\tau)$ is assumed to be $\vartheta_t^{NT_t(\tau)-1}$ where $0 \leq \vartheta_t \leq 1$.

In the experimentation E1 of Table 2, it takes 46 [sec] to exclusively perform a computation process cp_1 on a server s_1. Here, $T_{11} = minT_{11} = 46$ [sec]. $F_{11} = maxF_{11} = 1/46 = 0.02$ [1/sec]. In the experimentation E4, another computation process cp_2 where $minT_{12} = 44$ starts at the same time as the computation process cp_1. Hence, $\alpha_t(\tau) = \varepsilon_t = 0.97$ and $NC_t(\tau) = 2$. Hence, $T_{11} = 1.03 \cdot minT_{11} = 47.3$ and $T_{12} = 1.03 \cdot minT_{12} = 45.3$ [sec]. Then, the total compression time in the experimentation E4 is 93 [sec]. On the other hand, it takes 46 [sec] to perform the computation process cp_1 on the server s_1 with which one transmission process is concurrently performed in the experimentation CT1 of Table 3. In the experimentation CT2, it takes 55 [sec] to perform the computation process cp_1 where a pair of transmission processes are concurrently performed. Here, $\theta_t(\tau) = \vartheta_t = 0.84$ and $NT_t(\tau) = 2$. Hence, $T_{11} = 1.19 \cdot minT_{11} = 55$ [sec].

We define the *normalized computation rate* (*NCR*) $f_{ti}(\tau)$ of a computation process cp_i on a server s_t at time τ as follows:

[Normalized computation rate (NCR)]

$$f_{ti}(t) = \begin{cases} \alpha_t(\tau) \cdot \theta_t(\tau) \cdot maxF_{ti}/maxF_i & [1/\text{tu}]. \\ \alpha_t(\tau) \cdot \theta_t(\tau) \cdot minT_i/minT_{ti} & [1/\text{tu}]. \end{cases} \tag{1}$$

The maximum NRC $maxf_{ti}$ is $maxF_{ti}$ / $maxF_i$. $0 \leq f_{ti}(\tau) \leq maxf_{ti} \leq 1$. $f_{ti}(\tau)$ shows how many number of computation steps of a computation process cp_i are performed on a server s_t at time τ. In this paper, we assume CPU resource is uniformly allocated to every current computation process as follows:

$$f_{ti}(\tau) = \alpha_t(\tau) \cdot \theta_t(\tau) \cdot maxf_i \; / \; |NC_t(\tau)|. \tag{2}$$

Suppose only a computation process cp_i is exclusively performed on every server at time τ. $f_{ti}(\tau) = 1$ for the fastest server s_t. If a server s_t is faster than another server s_u and the computation process cp_i is exclusively performed on the servers s_t and s_u, $f_{ui}(\tau) < f_{ti}(\tau) \leq 1$. The normalized computation rate $f_t(\tau)$ of the server s_t is $\sum_{cp_i \in CP_t(\tau)} f_{t_i}(\tau)$. $f_t(\tau) = maxf_t$ if cp_i is exclusively performed on the server s_t.

Next, suppose that a computation process cp_i starts and terminates on a server s_t at time $s\tau_{ti}$ and $e\tau_{ti}$, respectively. Here, $T_{ti} = e\tau_{ti} - s\tau_{ti}$. Here, $\int_{s\tau_{ti}}^{e\tau_{ti}} f_{ti}(\tau)\, dt = minT_i$. The computation laxity $lc_{ti}(\tau) = minT_i - \int_{s\tau_{ti}}^{\tau} f_{ti}(\tau)\, d\tau$ shows how much computation the server s_t has to spend to perform the computation process cp_i at time τ.

We define a *simple computation model* which each server s_t satisfies the following properties:

[Simple computation model]

1. $maxf_{ti} = maxf_{tj} = maxf_t$ for every pair of different computation processes cp_i and cp_j on a server s_t.
2. $f_t(\tau) = \alpha_t(\tau) \cdot \theta_t(\tau) \cdot maxf_t$.

3.3 Transmission Model

Suppose a server s_t concurrently transmits files f_1, ..., f_m to a set C_t of clients c_1, ..., c_m at rates $tr_{t1}(\tau)$, ..., $tr_{tm}(\tau)$ ($m \geq 1$), respectively, at time τ. Let b_{ts} be the maximum network bandwidth [bps] between

the server s_t and a client c_s. Let $Maxtr_t$ be the maximum transmission rate [bps] of the server s_t ($\leq b_{ts}$). Here, the total transmission rate $tr_t(\tau)$ of the server s_t is $tr_{t1}(\tau) + \cdots + tr_{tm}(\tau)$ and $0 \leq tr_t(\tau) \leq Maxtr_t$.

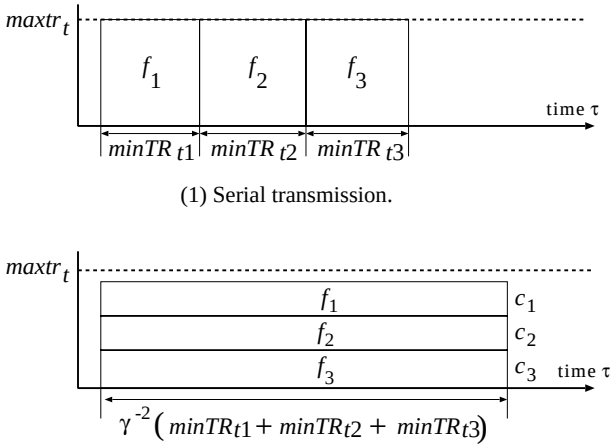
Each client c_s receives a file f_s at receipt rate $rr_s(\tau)$ at time τ. Let $Maxrr_s$ be the maximum receipt rate of a client c_s. We assume each client c_s receives a file from at most one server at rate $Maxrr_s$ (= $rr_s(\tau)$). The server s_t allocates the client c_s with transmission rate $tr_{ts}(\tau)$ so that $tr_{ts}(\tau) \leq Maxrr_s$.

Let TR_{ts} be the total transmission time of a file f_s from a server s_t to a client c_s. Let $minTR_{ts}$ show the minimum transmission time $|f_s| / min(Maxrr_s, Maxtr_t)$ [sec] of a file f_s from a server s_t to a client c_s where $|f_s|$ is the size [bit] of the file f_s. $TR_{ts} \geq minTR_{ts}$. Suppose a server s_t starts and ends transmitting a file f_s to a client c_s at time $s\tau$ and $e\tau$, respectively. Here, $\int_{s\tau}^{e\tau} tr_{ts}(\tau)\, d\tau = |f_s|$ and the transmission time TR_{ts} is ($e\tau$ - $s\tau$). If the server s_t sends only the file f_s to the client c_s, $tr_{ts}(\tau) = min(Maxtr_t, Maxrr_s)$ [bps]. The transmission laxity $lt_{ts}(\tau)$ is $|f_s|$ - $\int_{\tau}^{e\tau} tr_{ts}(\tau)\, d\tau$ [bit] at time τ.

First, we consider a model where a server s_t satisfies the following properties:

[Server-bound model] If $Maxrr_1 + \cdots + Maxrr_m \geq \sigma(\tau) \cdot Maxtr_t$, $\sum_{c_{ts} \in CT_t(\tau)} tr_{ts}(\tau) = \sigma(\tau) \cdot Maxtr_t$ for every time τ.

$\sigma(\tau)$ (≤ 1) is the *transmission degradation* factor. Here, we assume $\sigma(\tau) = \gamma^{(1-|NT_t(\tau)|)}$ ($0 < \gamma \leq 1$) at time τ. The more number of clients s_t transmits, the longer it takes. The *effective transmission rate* of the server s_t is $\sigma(\tau) \cdot Maxtr_t$.



(1) Serial transmission.

(2) Parallel transmission.

Figure 4: Transmission time.

Let us consider three files f_1, f_2, and f_3 which a server s_t transmits to clients c_1, c_2, and c_3, respectively. First, suppose that a server s_t serially transmits the files f_1, f_2, and f_3, i.e. $e\tau_1 = s\tau_2$ and $e\tau_2 = s\tau_3$ as shown in Figure 4. Here, the transmission time TR_t is $e\tau_3$ - $s\tau_1 = minTR_{t1} + minTR_{t2} + minTR_{t3}$. Next, suppose a server s_t starts concurrently transmitting files f_1, f_2, and f_3 at time $s\tau$ and terminates at time $e\tau$ as shown in Figure 4 (2). Here, $NT_t(\tau) = 3$ and $\gamma^{-2} \cdot TR_t = minTR_{t1} + minTR_{t2} + minTR_{t3}$. For $\gamma = 0.98$, it takes about 1.4% longer time than the serial transmission.

On the other hand, we consider another environment where some client c_s cannot receive a file from a server s_t at the maximum transmission rate $Maxtr_t$, i.e. $Maxrr_s < Maxtr_t$. Hence, the transmission rate $tr_{ts}(\tau)$ is the maximum receipt rate $Maxrr_s$ of c_s.

[Client-bound model] If $Maxrr_1 + \cdots + Maxrr_m \leq \sigma_t(\tau) \cdot Maxtr_t$, $\sum_{c_{ts} \in CT_t(\tau)} tr_{ts}(\tau) = Maxrr_1 + \cdots + Maxrr_m$ for every time τ.

4 Power consumption model

Let $maxE_t$ be the maximum electric power consumption of a server s_t. $minE_t$ is the minimum electric power consumption. $E_t(\tau)$ shows the electric power consumption rate of a server s_t at time τ [W/tu] (t = $1,\ldots,n$), $minE_t \leq E_t(\tau) \leq maxE_t$. The electric power consumption rate $PT_t(tr)$ [18, 19] of a server s_t at time τ when only transmission processes are being performed is given as follows:

$$PT_t(tr_t(\tau)) = \beta_t(|NT_t(\tau)|) \cdot \delta_t \cdot tr_t(\tau) + minE_t. \tag{3}$$

Here, δ_t is the power consumption rate for a server s_t to transmit one Mbits [W/Mb]. For $m = |NT_t(\tau)|$, $\beta_t(m)$ shows how much power consumption is increased for the number m of transmission processes, $\beta_t(m) \geq 1$ and $\beta_t(m) > \beta_t(m$ - 1). There is a fixed point $maxm_t$ such that $\beta_t(maxm_t$ - 1) $\leq \beta_t(maxm_t) = \beta_t(maxm_t + h)$ for $h > 0$. $maxPT_t = \beta_t(maxm_t) \cdot \delta_t \cdot Maxtr_t + minE_t$.

[Simple power consumption model]

1.
$$maxE_t \geq maxPT_t. \tag{4}$$

2.
$$E_t(\tau) = \begin{cases} maxE_t & \text{if } NC_t(\tau) \geq 1. \\ PT_t(tr_t(\tau)) & \text{if } NC_t(\tau) = 0 \text{ and } NT_t(\tau) \geq 1. \\ minE_t & \text{otherwise.} \end{cases} \tag{5}$$

If at least one computation process cp_t is performed, the electric power is maximally consumed on a server s_t. If only transmission processes are performed, the electric power consumption depends on the total transmission rate of a server s_t.

5 Selection Algorithm of Servers

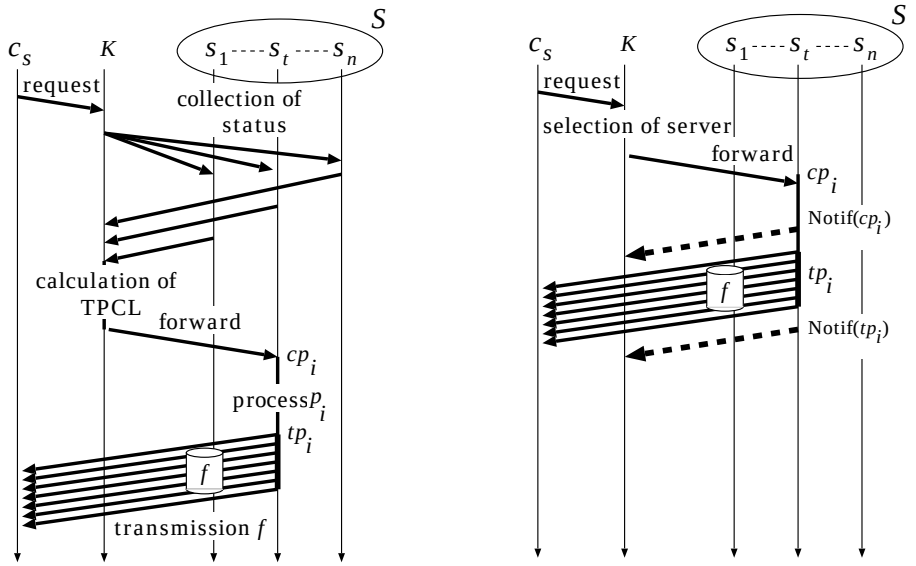
5.1 Transmission rates

Suppose a server s_t receives a request from a client c_s at time τ. The request is performed as a process p_{ts} (= $\langle cp_{ts}, tp_{ts}\rangle$) on the server s_t. The maximum transmission rate $maxtr_t(\tau)$ of the server s_t depends on the transmission degradation factor $\sigma_t(\tau) = \gamma^{(1-|NT_t(\tau)|)}$. If a new transmission process tp_{ts} is started and a current transmission process tp_{ts} is terminated at τ, $CT_t(\tau) = CT_t(\tau) \cup \{tp_{ts}\}$ and $CT_t(\tau) = CT_t(\tau)$ - $\{tp_{ts}\}$, respectively. Here, the maximum transmission rate $maxtr_t(\tau)$ is calculated as $\gamma^{1-NT_t(\tau)} \cdot Maxtr_t$. Here, $0 < \gamma \leq 1$. In order to more efficiently use the total transmission rate $maxtr_t(\tau)$, the following algorithm [18, 19] to allocate each client c_s with the transmission rate $tr_{ts}(\tau)$ is discussed:

1. V = 0; R = 0; $TS = maxtr_t / NT_t(\tau)$;
2. For each client c_s, $tr_t(\tau) = TS$ and $R = R + (TS$ - $Maxrr_s)$ if $Maxrr_s \leq TS$. Otherwise, $tr_t(\tau) = Maxrr_s$ and $V = V + (Maxrr_s$ - $TS)$.
3. For each client c_s, $tr_{ts}(\tau) = tr_{ts}(\tau) + V \cdot (Maxrr_s$ - $tr_{ts}(\tau)) / R$ if $tr_{ts}(\tau) < Maxrr_s$.

5.2 Communication between a load balancer and servers

In the EPCLB algorithm [21], a server s_t which consumes the minimum power consumption is selected for a client c_s, i.e. a server s_t whose total power consumption laxity (TPCL) is minimum is selected in the server set S. The TPCL shows how much power a server s_t has to consume to perform up all the computation processes and transmission processes at time τ. In order to calculate the TPCL, a load balancer K has to collect the computation and communication laxities of every server in the server set S for each time the load balancer K receives a request from a client c_s as shown in Figure 5 (1). After collecting the computation and communication laxities from every server, the load balancer K calculates the TPCL and forwards the request to a server s_t whose TPCL is minimum. Due to the communication delay between a load balancer K and servers, the status of each server might be changed during the estimation process in the EPCLB algorithm. In addition, if the number of clients concurrently performed is increased, the computation and communication overhead to estimate the TPCL of servers is increased on the load balancer K. Hence, it is difficult to correctly estimate the TPCL of servers in the real environments.



(1) EPCLB algorithm. (2) CTRB algorithm.

Figure 5: Communication between a load balancer and servers.

In this paper, we propose an algorithm for reducing the total power consumption of servers and the overhead of a load balancer, which does not consider the power consumption laxity. Hence, it does not need to collect the computation and communication laxities of every server in the server set S for each time the load balancer K receives a request from a client c_s as shown in Figure 5 (2). We assume that a request message sent by a client c_s to a load balancer K includes the maximum receipt rate $Maxrr_s$ of the client c_s. The load balancer K selects one server s_t in the server set S and forwards the request to the server s_t. On receipt of the request, the server s_t first performs a computation process cp_t of the request process p_t and manipulates the data d. If the computation process cp_t is terminated, the server s_t sends the termination notification $Notif(cp_t)$ of the computation process cp_t to the load balancer K and then starts the transmission process tp_t of the request process p_t. Finally, if the transmission process tp_t is terminated, the server s_t sends the termination notification $Notif(tp_t)$ of the transmission process tp_t to the load balancer K.

5.3 Computation and transmission rate based (CTRB) algorithm

We discuss how to select a server so that the total power consumption of servers and the overhead of a load balancer K can be reduced. Suppose a load balancer K holds the following parameters of every server s_t:

- $maxf_t$ = the maximum normalized computation rate of the server s_t.
- ε_t = the computation degradation rate of the server s_t.
- ϑ_t = the transmission-computation degradation rate of the server s_t.
- $maxE_t$ = the maximum power consumption rate of the server s_t.
- $Maxtr_t$ = the maximum transmission rate of the server s_t.

Suppose every current computation process on the server s_t is terminated and shifts to transmission process at time τ. $MaxRR_t(\tau)$ shows how much receipt rate is maximumly requested on each server s_t at time τ. $MaxRR_t(\tau) = \Sigma_{c_{ts} \in CP_t(\tau)} Maxrr_s + \Sigma_{c_{ts} \in CT_t(\tau)} Maxrr_s$. A load balancer K holds the following variables:

- $\mathbf{CP}(\tau) = \{CP_1(\tau), ..., CP_n(\tau)\}$.
- $\mathbf{CT}(\tau) = \{CT_1(\tau), ..., CT_n(\tau)\}$.
- $\mathbf{MR}(\tau) = \{MaxRR_1(\tau), ..., MaxRR_n(\tau)\}$.

A load balancer K can confirm what computation and transmission processes are being performed on each server s_t at time τ by checking the variables $\mathbf{CP}(\tau)$ and $\mathbf{CT}(\tau)$. In addition, the load balancer K can confirm how much receipt rate are maximumly requested on each server s_t at time τ by checking the variable $\mathbf{MR}(\tau)$.

Suppose a load balancer K forwards a request received from a client c_s to a server s_t at time τ. The variables are manipulated on the load balancer K as follows:

FORWARD() {
  $CP_t(\tau) = CP_t(\tau) \cup \{cp_{ts}\}$;
  $MaxRR_t(\tau) = \Sigma_{c_{ts} \in CP_t(\tau)} Maxrr_s + \Sigma_{c_{ts} \in CT_t(\tau)} Maxrr_s$;
}

Suppose a load balancer K receives a termination notification of a computation process cp_{ts} or a transmission process tp_{ts} from a server s_t. The variables are manipulated on the load balancer K as follows:

RECEIVE() {
  if $Message = Notif(cp_{ts})$ { /* the computation process cp_{ts} is terminated on a server s_t. */
    $CP_t(\tau) = CP_t(\tau)$ - $\{cp_{ts}\}$;
    $CT_t(\tau) = CT_t(\tau) \cup \{tp_{ts}\}$;
  } **else if** $Message = Notif(tp_{ts})$ { /* the transmission process tp_{ts} is terminated on a server s_t. */
    $CT_t(\tau) = CT_t(\tau)$ - $\{tp_{ts}\}$;
    $MaxRR_t(\tau) = \Sigma_{c_{ts} \in CP_t(\tau)} Maxrr_s + \Sigma_{c_{ts} inCT_t(\tau)} Maxrr_s$;
  }

}

The *computation and transmission rate based* (CTRB) algorithm is implemented on a load balancer K. In the CTRB algorithm, the servers s_1, ..., s_n in the server set S are totally ordered based on the ascending order of the maximum power consumption $maxE_t$. $maxE_1 \leq \cdots maxE_t \cdots \leq maxE_n$.

The effective normalized computation rate $f_t(\tau)$ of each server s_t at time τ is $\alpha_t(\tau) \cdot \theta_t(\tau) \cdot maxf_t = \varepsilon_t^{|CP_t(\tau)|-1} \cdot \vartheta_t^{|CT_t(\tau)|-1} \cdot maxf_t$. If $\alpha_t(\tau) \cdot \theta_t(\tau) = 0.5$, the computation time to perform up all current computation processes on the server s_t is two times longer. In addition, the power consumption to perform up all current computation processes on the server s_t is two times larger. Here, $Over(s_t)$ is a threshold value to judge the computation overload of a server s_t. If $Over(s_t) > \alpha_t(\tau) \cdot \theta_t(\tau)$, a server s_t is overloaded with respect to computation at time τ. Otherwise, the server s_t is not overloaded with respect to computation. For example, if $Over(s_t) = 0.7$ and $\alpha_t(\tau) \cdot \theta_t(\tau) = 0.5$ on a server s_t, the server s_t is judged to be overloaded.

Next, if $MaxRR_t(\tau) > Maxtr_t$, a server s_t is judged to be overloaded with respect to transmission, i.e. some clients assigned to a server s_t cannot receive files with the maximum receipt rates of the clients. Otherwise, the server s_t is not overloaded with respect to transmission.

In the CTRB algorithm, if a server s_1 is overloaded with respect to computation or transmission, a request from a client c_s is forwarded to the second server s_2. Thus, if severs s_1, ..., s_i are overloaded with respect to computation or transmission, a request is forwarded to a server s_{i+1} ($i < n$). If every server is overloaded with respect to computation or transmission, a request is forwarded to a server s_i based on the basic round robin (RR) algorithm. That is, if the previous request is assigned to a server s_i, the next request is assigned to a server s_{i+1}.

On receipt of a request from a client c_s, a load balancer K selects a server s_t in the server set S by the following procedure:

CTRB(τ, c_s, $Maxrr_s$, s_i) { /* s_i is a server assigned a previous request. */
  $server = \phi$;
  for each server s_t **in** S {
    if not($Over(s_t) > \varepsilon_t^{|CP_t(\tau)|-1} \cdot \vartheta_t^{|CT_t(\tau)|-1}$ **or** $MaxRR_t(\tau) > Maxtr_t$) {
      $server = s_t$;
      break;
    }
  }
  if $server = \phi$, $server = s_{i+1}$;
  $s_i = server$;
  return($server$);
}

6 Evaluation

We evaluate the CTRB algorithm in terms of total power consumption [Ws], elapse time, and total number of messages between a load balancer K and servers compared with the EPCLB and RR algorithms [9] through the simulation. There are ten servers s_1, ... s_{10} each of which holds full replicas of three pieces of data d_1, d_2, and d_3 of 556, 278, and 112 [Mega-byte], respectively, as shown in Table 4, $S = \{s_1, ..., s_{10}\}$. The servers in the server set S are ordered based on the ascending order of the maximum power consumption $maxE_t$. Each time a server s_t receives a request to obtain data d_h from a client, the

server s_t compresses the data d_h to a file f_h and then transmits the file f_h to a client c_s. The sizes of compressed files $|f_1| = 500$, $|f_2| = 250$, and $|f_3| = 100$ [MB]. Here, the minimum computation time $minT_{1d_1}$, $minT_{1d_2}$, and $minT_{1d_3}$ to compress data d_1, d_2, and d_3 on the fastest server s_5 are 42, 20, and 4 [sec], respectively. The minimum power consumption rate $minE_t$ is randomly selected between 87[W/tu] and 150 [W/tu]. The maximum power consumption rate $maxE_t$ is randomly selected between 1.2 times and 1.4 times higher than $minE_t$. The maximum normalized computation rate (NCR) $maxf_t$ of each server s_t is randomly selected between 1 and 0.73 [1/tu] (t = 1, ..., 10) so that $maxf_5 = 1$ and $maxf_{10} = 0.73$. The computation degradation rate ε_t is randomly selected between 0.99 and 0.95. A threshold value $Over(s_t)$ to judge the computation overload of each server s_t is 0.8. The transmission-computation degradation rate ϑ_t is randomly selected between 0.83 and 0.89. The transmission degradation factor γ is randomly selected between 0.98 and 0.97. The power consumption coefficient δ_t to transmit one Mbits is randomly selected between 0.02 and 0.11 [W/Mb]. The increasing rate of the power consumption $\beta_t(m)$ for the number m of clients to which the server s_t concurrently transmits files is randomly selected between 1.09 and 1.41. The maximum transmission rate $Maxtr_t$ is randomly selected between 175 and 450 [Mbps]. The maximum power consumption rate $maxPT_t(tr)$ for transmission process is 95% of $maxE_t$. In this paper, we take one [sec] as one time unit [tu]. In this simulation, we assume that the communication delay between a load balancer and servers is small and bounded, i.e. the communication delay between a load balancer and servers can be neglected. Hence, in the EPCLB algorithm, the TPCL of each server can be correctly estimated at every time τ and a server which consumes the minimum TPCL for each request is correctly selected.

Table 4: Servers

Server	$maxE_t$	$minE_t$	$maxf_t$	ε_t	ϑ_t	δ_t	$\beta_t(m)$	$Maxtr_t$	γ_t	$maxPT_t(tr)$
s_1	104	87	0.99	0.98	0.89	0.02	1.1	450	0.98	99
s_2	107	89	0.79	0.99	0.88	0.02	1.09	440	0.98	101
s_3	113	94	0.91	0.98	0.88	0.02	1.21	418	0.97	107
s_4	131	101	0.94	0.97	0.87	0.04	1.19	390	0.98	125
s_5	210	150	1	0.95	0.83	0.11	1.48	390	0.98	200
s_6	150	115	0.85	0.98	0.87	0.05	1.41	356	0.97	142
s_7	159	122	0.76	0.96	0.83	0.03	1.1	304	0.97	151
s_8	176	135	0.88	0.96	0.85	0.08	1.32	280	0.98	167
s_9	190	136	0.82	0.96	0.84	0.07	1.25	356	0.98	181
s_{10}	200	143	0.73	0.95	0.83	0.10	1.39	175	0.97	190

Totally 750 number of clients issue requests of data d_h to a load balancer K at time $s\tau_s$. Each client c_s issues one request at time $s\tau_s$. The maximum receipt rate $Maxrr_s$ of each client c_s is randomly selected between 1 and 100 [Mbps]. Each client c_s randomly selects data d_h. The starting time $s\tau_s$ is randomly selected between 0 and 9,000 [sec]. Figure 6 shows a traffic pattern, i.e. how many number of clients start at time. In the simulation, the CTRB, EPCLB, and RR algorithms are performed for the same traffic pattern.

Figure 7 shows the power consumption rate [W/sec] of the servers in the server set S at time τ. Table 5 shows the total power consumption [Ws] which is obtained by subtracting the minimum power consumption $minE_t$ of each server s_t in the idle state from the total power consumption. The total power consumption of the CTRB algorithm is 642,645 [Ws]. The total power consumption of the EPCLB algorithm is 475,447 [Ws]. The total power consumption of the RR algorithm is 745,447 [Ws]. In the CTRB and EPCLB algorithms, 14% and 36% of power consumption can be reduced, respectively, compared with the RR algorithm. The total power consumption can be more reduced in the EPCLB algorithm

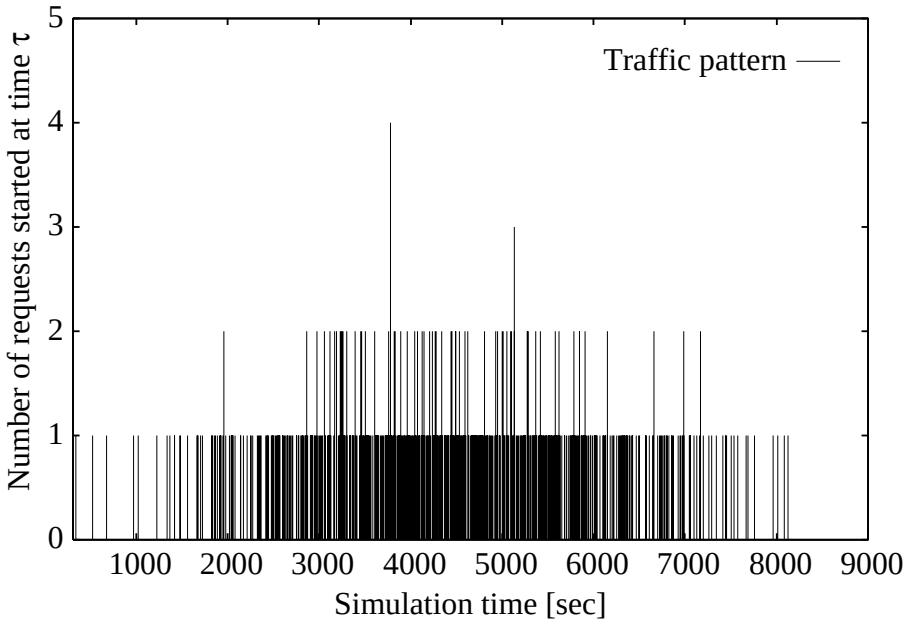


Figure 6: Total number of processes starting at time $s\tau_s$.

than the CTRB algorithm. In this simulation, we assume that the communication delay between a load balancer and servers can be neglected. Hence, the TPCL of each server can be correctly estimated in the EPCLB algorithm. However, the communication delay is increased in real environments due to the overhead of a load balancer to estimate the TPCL of each server in the EPCLB algorithm. Hence, in real environment, it is difficult to correctly estimate the TPCL of servers in the EPCLB algorithm and the total power consumption of the EPCLB algorithm is more increased. On the other hand, there is no need to exchange messages between a load balancer and servers to select a server for each request in the CTRB algorithm. The assumption of the communication delay in this simulation does not influence the simulation result of the CTRB algorithm.

Table 5: Total power consumption [Ws]

CTRB	EPCLB	RR
642,645	475,447	745,447

Table 6 shows the elapse time to terminate the total number 750 of requests in the CTRB, EPCLB, and RR algorithms. It takes 8,155, 8,154, and 8,155 [sec] in the CTRB, EPCLB and RR algorithms, respectively. The difference of the elapse time among the CTRB, EPCLB and RR algorithms is neglectable.

Table 6: The elapse time [sec]

CTRB	EPCLB	RR
8,155	8,154	8,155

Table 7 shows the number of messages between a load balancer K and servers to forward the total number 750 of requests to a server in the CTRB, EPCLB and RR algorithms. 2,250, 15,750, and 750 messages are exchanged between a load balancer K and servers in the CTRB, EPCLB, and RR algorithms, respectively. In the EPCLB algorithm, a load balancer K has to send messages to every server

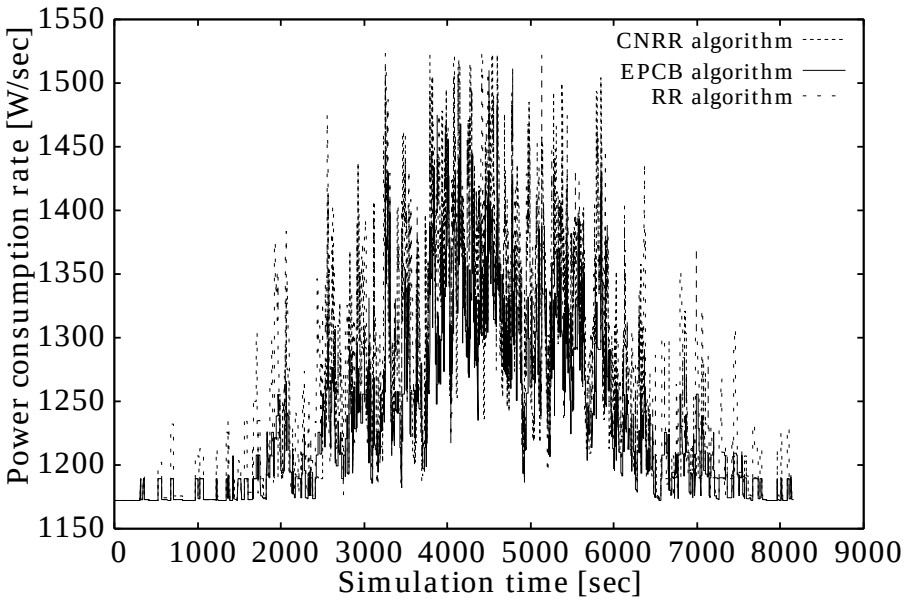


Figure 7: Power consumption rate of servers [W/sec].

to collect the current status of each server in the server set S each time the load balancer K receives a request from a client c_s as shown in Figure 5 (1). After collecting the current status from every server, the load balancer K starts to calculate the TPCL and forwards the request to a server s_i whose TPCL is minimum. On the other hand, there is no need to exchange messages between a load balancer K and servers to select a server for each request in the CTRB algorithm as shown in Figure 5 (2). Hence, the number of messages exchanged between the load balancer K and servers can be more reduced in the CTRB algorithm than the EPCLB algorithm. In addition, the server selection time for each request in the CTRB algorithm can be shorter than the EPCLB algorithm.

Table 7: The number of messages

CTRB	EPCLB	RR
2,250	15,750	750

From the evaluation results, the difference of elapse time among the CTRB, EPCLB, and RR algorithms is neglectable. The total power consumption can be more reduced in the CTRB and EPCLB algorithms than the RR algorithm. The total power consumption can be more reduced in the EPCLB algorithm than the CTRB algorithm. However, more number of messages have to be exchanged between a load balancer K and servers to forward a request to a server in the EPCLB algorithm than the CTRB and RR algorithms. In real environments, the status of each server might be changed during the estimation process due to the communication and computation overheads in the EPCLB algorithm. In reality, it is difficult to correctly estimate the power consumption of servers in the EPCLB algorithm. On the other hand, the total power consumption of servers can be reduced in the CTRB algorithm than the RR algorithm. In addition, the number of messages to be exchanged between a load balancer K and servers can be more reduced in the CTRB algorithm than the EPCLB algorithm. Therefore, the CTRB algorithm is simpler and more useful than the EPCLB and RR algorithms for real environments.

7 Concluding Remarks

In information systems, it is critical to reduce the total power consumption of computers and networks in order to realize the digital ecosystems and the green IT technologies. In the previous studies, the EPCLB algorithm is proposed to select one of servers so that the total power consumption of the servers can be reduced for general types of applications. In the EPCLB algorithm, a server whose TPCL is minimum is selected for a new request in a set of servers. The TPCL shows how much electric power a server has to consume to perform up all the computation and transmission processes at time τ. Therefore, in the EPCLB algorithm, a load balancer has to collect current status and calculate the TPCL of every server to select a server for a request each time the load balancer receives a new request. Here, if the number of clients concurrently performed is increased, the computation and communication overheads to estimate the TPCL of servers are increased on a load balancer. In addition, the status of each server might be changed during the estimation process due to the communication delay between a load balancer and servers. Hence, in the EPCLB algorithm, it is difficult to correctly estimate the TPCL of servers and the load balancer might be bottleneck of the system in real environments.

In this paper, we newly proposed the CTRB algorithm to select a server in a set of servers so that the total power consumption of servers and the overhead of a load balancer can be reduced. In the CTRB algorithm, a server in a set of servers is selected for a new request without considering the TPCL of servers. Hence, a load balancer does not need to collect current status of every server in a server set S each time the load balancer receives a new request from a client. As the result, the overhead of a load balancer to select a server for each request can be reduced. We evaluate the CTRB algorithm in terms of the total power consumption of servers, the elapse time to terminate the every request, and the total number of messages between a load balancer and servers compared with the RR and EPCLB algorithms. The difference of the elapse time among the CTRB, EPCLB, and RR algorithms is neglectable. In the CTRB and EPCLB algorithms, 14% and 36% of the total power consumption can be reduced, respectively, compared with the RR algorithm. The total power consumption can be more reduced in the EPCLB algorithm than the CTRB algorithm. However, more number of messages have to be exchanged between a load balancer and servers to forward a request to a server in the EPCLB algorithm than the CTRB algorithm. Hence, in real environments, the estimation of the TPCL in the EPCLB algorithm might not be correct and the load balancer might be bottleneck of the system. On the other hand, the number of messages to be exchanged between a load balancer and servers can be more reduced in the CTRB algorithm than EPCLB algorithm. Therefore, the CTRB algorithm is simpler and more useful than the EPCLB and RR algorithms for real environments.

References

[1] A. Waluyo, W. Rahayu, D. Taniar, and B. Srinivasan, "A novel structure and access mechanism for mobile broadcast data in digital ecosystems," *accepted for publication in IEEE Trans. on Industrial Electronics*, 2009.

[2] G. Zhao, K. Xuan, W. Rahayu, D. Taniar, M. Safar, M. Gavrilova, and B. Srinivasan, "Voronoi-based continuous k nearest neighbor search in mobile navigation," *accepted for publication in IEEE Trans. on Industrial Electronics*, 2009.

[3] S. Hemmert, "Green hpc: From nice to necessity," *Computing in Science and Engineering*, vol. 12, no. 6, pp. 8–10, November-December 2010.

[4] R. Murphy, T. Sterling, and C. Dekate, "Advanced architectures and execution models to support green computing," *Computing in Science and Engineering*, vol. 12, no. 6, pp. 38–47, November-December 2010.

[5] A. Barolli, F. Xhafa, C. Sanchez, and M. Takizawa, "A study on the performance of serch methods for mesh router nodes placement," in *Proc. of the 25th IEEE International Conference on Advanced Information*

Networking and Applications (AINA'11), Biopolis, Singapore. IEEE, March 2011, pp. 756–763.

[6] M. Ikeda, E. Kulla, M. Hiyama, L. Barolli, and M. Takizawa, "Experimental results of a manet testbet in indoor stairs environment," in *Proc. of the 25th IEEE International Conference on Advanced Information Networking and Applications (AINA'11), Biopolis, Singapore*. IEEE, March 2011, pp. 779–786.

[7] A. Durresi, M. Durresi, V. Paruchuri, and L. Barolli, "Ad hoc communications for emergency conditions," in *Proc. of the 25th IEEE International Conference on Advanced Information Networking and Applications (AINA'11), Biopolis, Singapore*. IEEE, March 2011, pp. 787–794.

[8] A. Bevilacqua, "A dynamic load balancing metthod on a heterogeneous cluster of workstations," *Informatica*, vol. 23, no. 1, pp. 49–56, 1999.

[9] "Job scheduling algorithms in linux virtual server," http://www.linuxvirtualserver.org/docs/scheduling.html, 2010.

[10] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster reserves: A mechanism for resource management in cluster-based network servers," in *Proc. of the 2000 ACM International Conference on Measurement and Modeling of Computer systems (SIGMETRICS'00), Santa Clara, California, USA*. ACM, June 2000, pp. 90–101.

[11] 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 (ISPASS'03), Austin, Texas, USA*. IEEE, March 2003, pp. 111–122.

[12] T. Heath, B. Diniz, E. V. Carrera, W. Meira, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proc. of the 10th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP'05), Chicago, Illinois, USA*. ACM, June 2005, pp. 186–195.

[13] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *IEEE Computer*, vol. 37, no. 11, pp. 68–74, November 2004.

[14] R. Bianchini and E. Carrera, "Analytical and experimental evaluation of cluster-based network servers," *World Wide Web journal*, vol. 3, no. 4, pp. 215–229, December 2000.

[15] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proc. of the 19th ACM symposium on Operating systems principles (SOSP'03), New York, USA*. ACM, October 2003, pp. 29–43.

[16] T. Enokido, K. Suzuki, A. Aikebaier, and M. Takizawa, "Laxity based algorithm for reducing power consumption in distributed systems," in *Proc. of the 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'10), Krakow, Poland*. IEEE, February 2010, pp. 321–328.

[17] T. Enokido, A. Aikebaier, and M. Takizawa, "A model for reducing power consumption in peer-to-peer systems," *IEEE Systems Journal*, vol. 4, no. 2, pp. 221–229, June 2010.

[18] T. Enokido, K. Suzuki, A. Aikebaier, and M. Takizawa, "Algorithms for reducing the total power consumption in data communication-based applications," in *Proc. of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA'10), Perth, Australia*. IEEE, April 2010, pp. 142–149.

[19] T. Enokido, A. Aikebaier, and M. Takizawa, "Power consumption-based server selection algorithms for communication-based systems," in *Proc. of the 13th International Conference on Network-Based Information Systems (NBiS'10), Takayama, Gifu, Japan*. IEEE, September 2010, pp. 201–208.

[20] ——, "Energy-efficient server selection algorithms for network applications," in *Proc. of the 5th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA'10), Fukuoka, Japan*. IEEE, November 2010, pp. 159–166.

[21] ——, "An integrated power consumption model for communication and transaction based applications," in *Proc. of the 25th IEEE International Conference on Advanced Information Networking and Applications (AINA'11), Biopolis, Singapore*. IEEE, March 2011, pp. 695–702.

[22] T. A. S. Foundation, "Apache module mod-deflate," http://httpd.apache.org/docs/2.0/en/mod/mod deflate.html, 2010.

[23] E. E. Devices, "Watts up? .Net," https://www.wattsupmeters.com/secure/products.php?pn=0, 2009.

Tomoya Enokido received BE and ME Degrees in Computers and Systems Engineering from Tokyo Denki University, Japan in 1997 and 1999, respectively. After that he worked for NTT Data Corporation, he joined Tokyo Denki University in 2002. He received his DE Degree in Computer Science from Tokyo Denki University in 2003. After that he worked for Computers and Systems Engineering as a research associate, he joined Faculty of Business Administration of Rissho University in 2005. He is an associate professor in the Faculty of Business Administration, Rissho University. His research interests include distributed systems. He is a member of IEEE..

Ailixier Aikebaier received his BE degree in Computers and Systems Engineering from XinJiang University, China, in 2000 and ME Degree in Computers and Systems Engineering from Tokyo Denki University, Japan in 2009. He received his DE Degree in Computer Science from Seikei University, Japan in 2011. He won the best paper award at CISIS2008 and CISIS2010. His research interests include distributed systems, P2P networks, consensus problems and fault-tolerant systems.

Makoto Takizawa is a Professor in the Department of Computer and Information Science, Seikei University. He was a Professor and the Dean of the Graduate School of Science and Engineering, Tokyo Denki University. He was a Visiting Professor at GMD-IPSI, Keele University, and Xidian University. He was on the Board of Governors and is a Golden Core member of IEEE CS and is a fellow of IPSJ. He received his DE in Computer Science from Tohoku University. He chaired many international conferences like IEEE ICDCS, ICPADS, and DEXA. He founded IEEE AINA. His research interests include distributed systems and computer networks.