Journal of Mobile Multimedia, Vol. 9, No. 1&2 (2013) 053–065
© Rinton Press

ENERGY-AWARE PASSIVE REPLICATION OF PROCESSES

DILAWAER DUOLIKUN
Department of Advanced Sciences, Hosei University, 3-7-2, Kajino-machi, Koganei-shi
Tokyo 184-8584, Japan
dilewerdolkun@gmail.com

AILIXIER AIKEBAIER
National Institute of Information and Communications Technology, 4-2-1, Nukui-Kitamachi, Koganei
Tokyo 184-8795, Japan
alisher@nict.go.jp

TOMOYA ENOKIDO
Faculty of Bussiness Administration, , Rissho University , 4-2-16, Osaki, Shinagawa
Tokyo 141-8602, Japan
eno@ris.ac.jp

MAKOTO TAKIZAWA
Department of Advanced Sciences, Hosei University , 3-7-2, Kajino-machi, Koganei-shi
Tokyo 184-8584, Japan
makoto.takizawa@computer.org

In information systems, processes requested by clients have to be performed on servers so that not only QoS (quality of service) requirements like response time are satisfied but also the total electric power consumed by servers to perform processes has to be reduced. Furthermore, each process has to be reliably performed in the presence of server faults. In our approach to reliably performing processes, each process is redundantly performed on multiple servers. The more number of servers a process is performed on, the more reliably the process can be performed but the more amount of electric power is consumed by the servers. Hence, it is critical to discuss how to reliably and energy-efficiently perform processes on multiple servers. In this paper, we discuss how to reduce the total electric power consumed by servers in a cluster where each request process is passively replicated on multiple servers. Here, a process is performed on only one primary server while taking checkpoints and sending the checkpoints to secondary servers. If the primary server is faulty, one of the secondary servers takes over the faulty primary server and the process is performed from the check point on the new primary server. We evaluate the energy-aware passive replication scheme of a process in terms of total power consumption and average execution time and response time of each process in presence of server fault.

Keywords: Energy-aware server cluster; Fault-tolerant server cluster; Process replication; Energy-aware passive replication (EPR); Digital ecosystems;

1 Introduction

In information systems [4], a client issues a process request to a server in a cluster of servers. Here, the process has to be performed on the server so that not only QoS (quality of service) requirements like response time are satisfied but also the total electric power consumed by servers to perform the process has to be reduced [3]. Furthermore, each process has to be reliably performed in presence of server faults. In our approach [7, 8, 9, 10, 11, 12, 13] to reliably performing processes, replicas of each process are redundantly performed on multiple servers in a server cluster. The more number of replicas of a process are performed, the more reliably the process can be performed. However, the more amount of electric power is consumed by the servers. Hence, it is critical to discuss how to reliably and energy-efficiently perform processes on multiple servers.

The authors discuss energy-efficient clusters where each process is actively replicated [1] on multiple servers in papers [7, 14, 15]. Here, replicas of a process are redundantly performed on multiple servers in a cluster. Even if some servers stop by fault, each process can be successfully performed as long as at least one server is operational. Here, the more amount of electric power is consumed by multiple servers while the reliability and availability can be increased. The authors discuss the algorithms in order to reduce the electric power consumed by servers. Here, once a replica of a process successfully terminates on one server, the other replicas are forced to terminate in papers [14, 15]. Furthermore, the total power consumption of servers to redundantly perform processes is reduced in the approach that the starting time of each replica is made different from the others [15].

In this paper, we discuss an energy-aware passive replication (EPR) algorithm, i.e. how to reduce the power consumption of servers with the passive replication [1] of a process in a cluster of servers. A replica of a process is performed on one primary server and is not performed on the other secondary servers. The primary server takes a checkpoint on the replica, i.e. local state of the replica is stored in a log. The primary server sends local state of the replica taken at the checkpoint to the other secondary servers in the cluster. On receipt of the checkpoint from the primary server, each secondary server saves the checkpoint in the log. On the other hand, if the primary server is faulty while the replica is being performed, one of the secondary servers takes over the faulty primary server and restarts a replica of the process on the local state taken at the checkpoint. Compared with the active replication, a replica of a process is performed on only one primary server at a time. If a primary server is faulty, one of the secondary servers takes over the faulty primary server and the replica restarts by rolling back to the checkpoint. This means, it takes a longer time to perform each process if a primary server is faulty. Thus, the total electric power consumed by servers can be reduced in the passive replication compared with the active replication. However, it takes time to recover from a primary server since a replica of the process has to restart at a checkpoint. We evaluate the energy-aware passive replication (EPR) algorithm of a process in terms of the total power consumption and the execution time and response time of each process.

In section 2, we present replication ways of a process on multiple servers. In section 3, we briefly present the power consumption model of a server. In section 4, we discuss how to select a primary server and secondary servers in a server cluster. In section 5, we evaluate the algorithm for energy-efficiently, passively replicating processes.

2 Process Replication

2.1 Active replication

Suppose a cluster S is composed of multiple servers $s_1, \ldots, s_n$ ($n \geq 1$). The servers $s_1, \ldots,$ s_n and a client c_s are interconnected in an underlying network N. Each messages is delivered to every destination process with no message loss. Processes are assumed to be less reliable in the sending order through the network N. A process issued by a client is performed on a server in the cluster S. In this paper, a term *process* means an application process which is performed on a server. We assume each server suffers from stop-fault, no Byzantine fault [5] and a client c_s is not faulty in this paper. A process is replicated on multiple servers in order to be tolerant of server fault. There are two ways to replicate a process; *active* and *passive* types of replications [1], [2].

In the active replication [2], a client c_s issues a process request p_i to multiple servers $\ldots$, s_t, $\ldots$, s_u, $\ldots$ in the cluster S as shown in Figure 1. On receipt of the process request p_i from the client c_s, a replica p_{ti} of the process p_i is created on each server s_t. Then, the process replica p_{ti} is performed on the server s_t. On termination of the process replica p_{ti}, the server s_t sends a reply to the client c_s. Once the client c_s receives a reply from one server s_t, the process p_i commits since servers may only stop by fault. Here, the client c_s can ignore replies from the other servers. As long as at least one server is operational, the process p_i can be successfully performed in the cluster S. If servers might suffer from Byzantine fault [5], the client takes the majority of replies from servers. In this paper, servers are assumed to only stop by fault as presented.

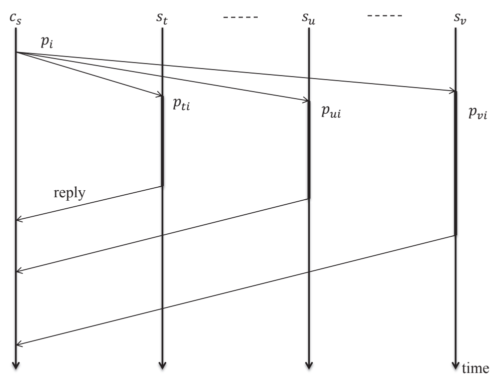

Fig. 1. Active replication.

Since a process is performed on more number of servers in the active replication, the more amount of electric power is consumed by the servers while the process can be reliably performed in presence of server faults [12]. In the papers [14, 15], the authors discuss how to reduce the total electric power consumption of the servers in the active replication of a process. Once a process replica successfully terminates on one server, process replicas being performed on the other servers are meaningless. Hence, every meaningless process replica is forced to terminate [14]. The electric power consumed by servers to perform meaningless process replicas after one process replica successfully terminates on one server can be reduced. Furthermore, every pair of process replicas do not start on servers at the same time [15]. That is, the starting time of each process replica is differentiated. By this way, the total amount of computation of process replicas, i.e. total electric power, can be reduced.

The semi-active replication [21] is discussed to perform non-deterministic processes on multiple servers. Here, replicas on secondary servers are performed while receiving checkpoints from a primary replica but do not send messages.

2.2 Passive replication

A process is performed on only one server in the passive replication [1] while performed on every server in the active replication. A client c_s first selects one server, say s_t in the server cluster S for a process request p_i as a *primary* server. The client c_s issues the process request p_i to the primary server s_t. On receipt of the process request p_i, a replica p_{ti} of the process p_i is created and performed on the primary server s_t. On termination of the process replica p_{ti}, the primary server s_t sends a reply to the client c_s. Here, the process p_i commits.

For each request process p_i, a *replica group* CS_i ($\subseteq S$) of servers including the primary server s_t are selected in a cluster S of servers. Here, the replica group CS_i includes a number rd_i ($1 \leq rd_i \leq n$) of servers. The client c_s selects one server as a primary server and the other servers are *secondary* servers in the replica group CS_i.

A primary server s_t may be faulty. A primary replica p_{ti} stops due to the stop fault of the primary server s_t. While the process replica p_{ti} is performed on the primary server s_t, checkpoints of the process replica p_{ti} are periodically taken on the primary server s_t as shown in Figure 2. Let cp^k_{ti} denote the kth checkpoint ($k \geq 1$) taken for a process replica p_{ti} on a server s_t. At the checkpoint cp^k_{ti}, the local state of the process replica p_{ti} is saved in the log L_{ti} and transmits the checkpoint cp^k_{ti} to every secondary server in the replica group CS_i. On receipt of the checkpoint cp^k_{ti} from the primary server s_t, a secondary server s_u saves the local state of the primary replica p_{ti} taken at the checkpoint cp^k_{ti} in the log L_{ui}. If the process replica p_{ti} successfully terminates on the primary server s_t, the termination notification message $Tnotif(p_{ti})$ is sent to the client c_s and then the process p_i commits.

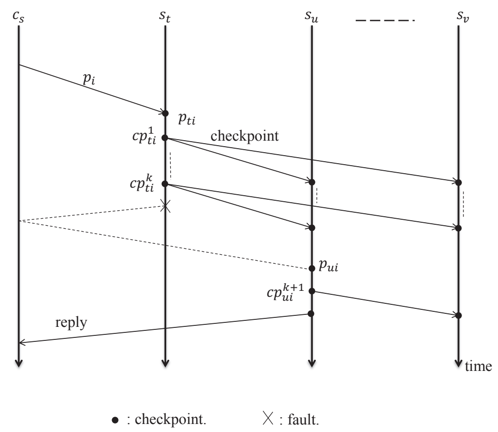

Fig. 2. Passive replication.

If the primary server s_t is faulty after taking the kth checkpoint cp^k_{ti} while the process replica p_{ti} is being performed, one of the secondary servers, say s_u takes over the faulty primary server s_t. In this paper, we assume the client c_s finds the primary server to be faulty according to the timeout mechanism. Then, the faulty server s_t is removed in the replica group CS_i. The client c_s selects a secondary server, say s_u in the replica group CS_i as a new primary server. A process replica p_{ui} of the process p_i is created on the primary server s_u by restoring the local state of the process replica p_{ti} in the checkpoint cp^k_{ti} stored in the log L_{ui}. On the primary server s_u, the process replica p_{ui} is restarted on the local state saved at the the kth checkpoint cp^k_{ti} most recently taken on the server s_t. The primary server s_u periodically takes checkpoints cp^{k+1}_{ui}, cp^{k+2}_{ui}, ..., $cp^{ncp_i}_{ui}$ of the process replica p_{ui} and sends the checkpoint cp^h_{ui} ($h \geq k+1$) to every secondary server in the replica group CS_i as discussed in the primary server s_t. A process replica is thus performed on only one server and only checkpoints are taken on the secondary servers. On the other hand, a process replica p_{ui} is restarted by rolling back to the kth checkpoint cp^k_{ti}. This means, it takes a longer time to recover from the fault of a primary server in the passive replication than the active replication.

3 Power Consumption Model

We consider the simple power consumption (SPC) model [7, 8, 9] of a server to perform processes. In the SPC model, the electric power consumption rate $E_t(\tau)$ [w] of a server s_t at time τ is either the minimum rate $minE_t$ or the maximum rate $maxE_t$ depending on a number of processes concurrently performed. Figure 3 shows the power consumption rate of a server with a one-core CPU which is obtained by measuring the power consumption of the server to perform processes for every one hundred [msec]. Here, only one process is performed on a server in the experimentation 1 (Exp.1). In the experimentations 2 (Exp.2) and 3 (Exp.3), multiple processes are concurrently performed on a server. Thus, if no process is performed on a server s_t at time τ, the power consumption rate $E_t(\tau)$ of the server s_t at time τ is $minE_t$. On the other hand, if at least one process is performed on a server s_t, the power consumption rate $E_t(\tau)$ is $maxE_t$ independently of the number n (≥ 1) of processes concurrently performed. For example, $minE_t$ is about 90 [W] and $maxE_t$ is 120 [W] in a computer with a single CPU in our experimentations as shown in Figure 3 [7, 8, 9, 16]. Types of power consumption models to perform types of processes are discussed in papers [10, 11, 16, 18].

The more number of processes are concurrently performed on a server, the longer it takes to perform each of the processes. Let $minT_{ti}$ show the minimum time to perform a process p_i on a server s_t, i.e. it takes $minT_{ti}$ [sec] to exclusively perform the process p_i without any other process. Let $minT_i$ show the minimum one of minimum computation time $minT_{1i}$, ..., $minT_{ni}$ of the process p_i on serves s_1, ..., s_n in a cluster S, $minT_i = min$ $\{minT_{1i}, \ldots, minT_{ni}\}$. That is, it takes $minT_i = minT_{ti}$ [sec] to exclusively perform the process p_i on the fastest server s_t in the cluster S. The maximum computation rate $maxF_{ti}$ of the process p_i is $minT_i$ / $minF_{ti}$ on the server s_t. The computation rate $F_{ti}(\tau)$ shows how much computation of a process replica p_{ti} is performed on a server s_t at time τ [9, 11, 13]. $F_{ti}(\tau) \leq maxF_{ti}$. Suppose a process replica p_{ti} starts at time st and ends at time et. Here, $\int_{st}^{et} F_{ti}(\tau)\, d\tau = minT_i$ [sec]. The computation rate $F_t(\tau)$ of a server s_τ at time τ is equally allocated to each current process replica p_{ti}. Let $CP_t(\tau)$ be a set of processes concurrently performed on a server s_t at time τ. $F_t(\tau) = \sum_{p_{ti} \in CP_t(\tau)} F_{ti}(\tau)$. It is noted $F_{ti}(\tau) = F_{tj}(\tau)$ for every pair of process replicas p_{ti} and p_{ui} in the fair process scheduling. $maxF_t$ indicates the maximum

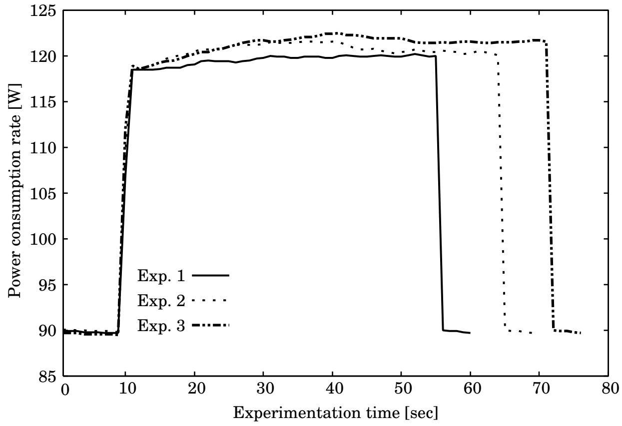

Fig. 3. Simple power consumption (SPC) model.

computation rate of a server s_t. If only a process p_{ti} is performed on a server s_t at time τ, $F_t(\tau) = F_{ti}(\tau)$ ($\leq maxF_t$). Here, the computation rate $F_t(\tau)$ is the maximum computation rate $maxF_t = maxF_{ti} = minT_i \ / \ minF_{ti}$. The more number of processes are concurrently performed at time τ, the smaller computation rate $F_t(\tau)$ is. $F_t(\tau) = \alpha_t(\tau) \cdot maxF_t$. Here, $\alpha_t(\tau)$ is the degradation factor which has the following properties:

(i) $0 \leq \alpha_t(\tau) \leq 1$.
(ii) $\alpha_t(\tau) = 1$ if $CP_t(\tau) = 1$.
(iii) $\alpha_t(\tau_1) \leq \alpha_t(\tau_2)$ if $CP_t(\tau_1) \geq CP_t(\tau_2)$.

For example, if only one process p_i is performed on a server s_t, $F_t(\tau) = maxF_t$, i.e. $\alpha_t(\tau) = 1$. If a pair of processes p_i and p_j are performed on a server s_t, $F_t(\tau) = 0.9 \cdot maxF_t$ where $\alpha_t(\tau) = 0.9$. Due to the process switch overhead, it takes a longer time to perform each of the processes, i.e. $1 \ / \ 0.9 = 1.11$ times longer than the exclusive execution of each of the processes.

The $energy-efficiency$ EF_t of a server s_t is defined to be a ratio $maxE_t \ / \ maxF_t$. The energy-efficiency EF_t shows how much amount of electric energy [W] a server s_t consumes to perform a computation unit. "$EF_t < EF_u$" means that a server s_t is more energy-efficient than another server s_u. For a process p_i, the *estimated* power consumption EP_{ti} of a server s_t to perform a process replica p_{ti} is $\alpha_t \cdot EF_t \cdot minT_{ti}$. Here, α_t is the degradation factor $\alpha_t(\tau)$ of the server s_t at current time τ. The degradation factor $\alpha_t(\tau)$ depends on the number of processes performed on a server s_t at time τ.

4 Server Selection Algorithms

A client c_s issues a request process p_i to a primary server in a server cluster $S = \{s_1, \ldots, s_n\}$ in the passive replication as discussed in the preceding section. We would like to discuss how a client c_s selects a primary server s_t and secondary servers for a request process p_i issued by a client c_s in the cluster S.

4.1 Replica groups of servers

A primary server s_t and $(rd_i - 1)$ $(rd_i \leq 1$) secondary servers to be in a replica group CS_i are selected in the cluster S for a request process p_i as follows:
[Server election algorithm]

(i) Select a primary server s_t where the estimated electric power consumption EP_{ti} to perform a process replica p_{ti} is the minimum in a cluster S of the servers.
(ii) Randomly select a number $(rd_i - 1)$ of secondary servers in the server cluster S.

Here, a replica group CS_i is composed of a primary server s_t and secondary servers selected in the selection algorithm. The client c_s issues a request process p_i to the primary server s_t. As discussed in the preceding section, one of the secondary servers takes over a primary server s_t if the primary server s_t stops by fault. In addition to a primary server s_t, secondary servers have to be selected in the replica group CS_i. Servers whose maximum power consumption rates are smaller can be selected as secondary servers. However, every pair of replica groups CS_i and CS_j for processes p_i and p_j, respectively, include a more number of common secondary servers. In addition, a process p_i is not performed on a secondary server in the replica group CS_i as long as a primary server s_t is operational. Even if an energy-efficient server s_u is selected to be a secondary server in the replica group CS_i, a process replica may not be performed on the server s_u. In addition, just checkpoints are taken on secondary servers to distribute overheads to every secondary server. Hence, secondary servers are randomly taken in the cluster S in this paper.

A process replica p_{ti} is performed on the primary server s_t as presented in the preceding section. Each time the primary server s_t takes the kth checkpoint cp_{ti}^k $(k \geq 1)$ of the process replica p_{ti}, the server s_t sends the checkpoint cp_{ti}^k to every secondary server in the replica group CS_i. In addition, the server s_t sends the information on the checkpoint cp_{ti}^k, i.e. the checkpoint number k to the client c_s. The client c_s recognizes which checkpoint cp_{ti}^k a primary server s_t has most recently taken when the client c_s detects the primary server s_t to be faulty. On receipt of the kth checkpoint cp_{ti}^k from the primary server s_t, each secondary server s_u saves the local state of the process replica p_{ti} in the log L_{ui} as presented before. Then, the secondary server s_u sends the checkpoint number k to the client c_s.

4.2 Selection of a primary server

In this paper, we assume the client c_s detects that the primary server s_t gets faulty. For example, the client c_s periodically sends an AYA (Are you alive) message to the primary server s_t. On receipt of the AYA message, the primary server s_t sends an IMA (I am alive) reply to the client c_s. If the client c_s had not received an IMA reply from the primary server s_t, the client c_s recognizes the primary server s_t to be faulty. If the client c_s thus detects the primary server s_t to be faulty after taking the kth checkpoint cp_{ti}^k, one secondary server s_u is selected to be a new primary server in the replica group CS_i as follows:
[Selection a primary server in secondary servers]

(i) Select a secondary server s_u where the checkpoint cp_{ti}^k is saved in the log L_{ui} and the estimated power consumption EP_{ti} to perform the process replica p_{ui} is the minimum in the replica group CS_i.

Here, the faulty primary server s_t is removed in the replica group CS_i. The new primary server s_u periodically takes checkpoints cp_{ui}^{k+1}, cp_{ui}^{k+2}, ..., $cp_{ui}^{ncp_i}$ while performing the process replica p_{ui}. As discussed here, secondary servers are randomly selected in the cluster S in this paper. Here, while a process replica p_{ti} can be energy-efficiently performed on a primary server s_t, a process replica p_{ui} may not be able to be energy-efficiently performed on a secondary server s_u if the primary server s_t gets faulty.

4.3 Revised selection algorithm

Another idea is that servers are classified into a number r_i ($\leq rd_i$) of classes C_{i1}, ..., C_{i,r_i} so that each server s_u in a class C_{il} is more energy-efficient than each server s_v in a class C_{im} for $l < m$, i.e. EF_u ($= maxE_u$ / $maxF_u$) $<$ EF_v ($= maxE_v$ / $maxF_v$). Then, servers are selected in each class C_{il} to be members of a replica group CS_i for $l = 1, \ldots, r_i$. For example, the number $r = \lceil rd_i/r_i \rceil$ of servers are included in each class C_{il}. Then, one of (rd_i $-$ r) servers are included in each class C_{i1}, ..., C_{i,rd_i-r}. Thus, for each process p_i, a replica group CS_i includes servers whose energy-efficiency is similarly distributed.

A primary server is selected for each request process p_i in classes C_{i1}, ..., C_{i,r_i} of a replica group CS_i as follows:
[Server selection algorithm 2]

(i) First, a primary server s_t is selected in the first class C_{i1} of the replica group CS_i.
(ii) Now, suppose a sever s_t of a class C_{il} is a primary sever in a replica group CS_i. Here, the other servers in the classes C_{il}, $C_{i,l+1}$, ..., C_{i,r_i} are secondary servers.
(iii) If the primary server s_t is detected to be faulty, the server s_t is removed in the class C_{il}. Then, a secondary server s_u is selected in secondary servers of the class C_{il} in the replica group CS_i.
(iv) If the class C_{il} is empty, a secondary server s_u is selected in the class $C_{i,l+1}$.

Here, a process p_i is first performed on a server which is most energy-efficient in a replica group CS_i. If a primary server is faulty, a server which is less energy-efficient takes over the primary server and a process replica is performed. Since the fault probability f_t of each server s_t is not large, a process can be energy-efficiently performed.

5 Evaluation

A client c_s issues a request process p_i to a cluster S of servers $s_1, \ldots, s_n$ ($n \geq 1$). We evaluate the energy-aware passive replication (EPR) algorithm in terms of total execution time and response time.

5.1 Environment

We assume a replica group CS_i includes ten servers $s_1, \ldots, s_{10}$ in the evaluation, i.e. $n = 10$.

It takes time to exchange messages among a client c_s and servers. Let d_{st} and d_{tu} be the delay time between a pair of a client c_s and a server s_t and a pair of servers s_t and s_u, respectively. Here, we assume the underlying network N is symmetric, i.e. $d_{st} = d_{ts}$ and d_{tu} $= d_{ut}$ for every client c_s and every pair of servers s_t and s_u. In this paper, we also assume the underlying network N is synchronous, i.e. the maximum delay time is bounded by some value is reliable, i.e. no message loss. Here, $maxd_{tu}$ and $mind_{tu}$ be the maximum delay time and the minimum delay time between a pair of servers s_t and s_u, i.e. $mind_{tu} \leq d_{tu} \leq maxd_{tu}$.

$maxd_{st}$ and $mind_{st}$ are the maximum delay time and the minimum delay time between a pair of a client c_s and a server s_t, i.e. $mind_{st} \leq d_{st} \leq maxd_{st}$. In the evaluation, we assume the maximum delay time among servers and a client is the same $maxd$ and the minimum delay time among serves and a client is also the same $mind$.

T_{ti} shows the execution time [sec] of a process replica p_{ti} on a server s_t. $minT_{ti}$ stands for the minimum execution time [sec] of a process replica p_{ti}. That is, it takes $minT_{ti}$ [sec] to exclusively perform a process replica p_{ti} without any other process on a server s_t. The more number of processes are concurrently performed on a server, the longer time it takes to perform each of the processes as presented in the preceding section. In this paper, we assume the degradation factor $\alpha_t(\tau)$ of each server s_t is 1 at every time τ for simplicity.

Suppose a process replica p_{ti} is performed on a primary server s_t. The primary server s_t periodically takes a checkpoint cp^k_{ti} of the process replica p_{ti}. We assume each process replica p_{ti} takes totally a number ncp_i of checkpoints, cp^1_{ti}, ..., $cp^{ncp_i}_{ti}$ ($ncp_i \geq 0$). This means, a checkpoint of each process replica p_{ti} is taken on a server s_t every T_{ti} / (ncp_i + 1) [sec]. In the evaluation, we assume each process p_i takes the same number ncp of checkpoints, i.e. ncp_i = ncp (≥ 0).

A server s_t might stop by fault. Each time a primary server s_t takes a checkpoint cp^k_{ti}, i.e. local state of the process replica p_{ti}, the primary server s_t sends the checkpoint cp^k_{ti} to every secondary server s_u in the replica group CS_i of a cluster S. On receipt of the kth checkpoint cp^k_{ti}, a secondary server s_u saves the local state taken at the checkpoint cp^k_{ti} in the log L_{ui}. Here, a process replica p_{ui} of a process p_i can be restarted on a local state in the checkpoint cp^k_{ti} on a secondary server s_u. That is, the local state of the process replica p_{ti} in the checkpoint cp^k_{ti} is restored in the process replica p_{ui} on the secondary server s_u. Here, k of (ncp_i + 1)the of the total computation of the process p_i completes in the process replica p_{ti} on the primary server s_t. The remaining part of the process p_i has to be performed on the secondary server s_u, i.e. $[(ncp_i - k + 1) / (ncp_i + 1)] \cdot minT_{ui}$ [sec]. Here, the secondary server s_u is a new primary server of the process p_i and the other (rd_i − 1) servers are secondary servers. The process replica p_{ui} is performed on the server s_u while periodically taking checkpoints in the same way as the previous primary server s_t. Here, the primary server s_u takes totally a number ($ncp_i - k$) of checkpoints. If the primary server s_u gets faulty, one of the secondary servers is selected to be a new primary server as presented here.

In this paper, we assume each server s_t suffers from stop-fault with probability f_t for each unit time [sec]. The client c_s detects the fault of the primary server s_c and informs of it to every secondary server in the replica group CS_i. We assume it takes $2 \cdot maxd_{tu}$ [sec] for a secondary server s_u to know the primary server s_t to be faulty since the primary server s_t gets faulty. In the evaluation, we assume every server s_t has the same fault probability f, i.e. $f_t = f$. We also assume a client c_s is not faulty.

In the evaluation, a process p_i is issued to a replica group CS_i of servers s_1, ..., s_n where $n = 10$. Here, we assume the minimum execution time $minT_i$ of the process p_i is 50 [msec]. The execution time T_{ti} of a process replica p_{ti} on a server s_t is randomly given in 50, 51, ..., 60 [msec]. The maximum delay time $maxd$ and minimum delay time $mind$ are 5 and 1 [msec], respectively. For every pair of servers s_t and s_u and a pair of the client c_s and each server s_t, the delay time d_{st} and d_{tu} are randomly given one of 1, 2, 3, 4, and 5 [msec].

5.2 Evaluation results

Figure 4 shows the total execution time of the process p_i for fault probability $f = 0.05$ and $f = 0.005$. The total execution time of a process p_i means the summation of execution time to perform a process replica p_{ti} on each server s_t. Here, there are ten servers in the replica group CS_i, $n = 10$ for the total number *ncp* (≤ 9) of checkpoints taken in the process p_i. For example, "$ncp = 2$" means a process p_i takes totally two checkpoints. For $f = 0.05$, the total execution time is about 100 [msec] and 65 [msec] for $ncp = 1$ and $ncp = 5$, respectively. For $f = 0.005$, the total execution time is about 55 [msec] and 50 [msec] for $ncp = 1$ and $ncp = 5$, respectively. Thus, the more frequently checkpoints are taken, the shorter the total execution time of a process is. The total power consumption consumed by the servers depends on the total execution time of a process as discussed in the preceding section.

Figure 5 shows the response time of the process p_i for the number *ncp* (≤ 9) of checkpoints and $n = 10$ with fault probability $f = 0.05$ and $f = 0.005$. The response time means how long it takes for a client c_s to receive a reply from a server since the client c_s sends a request process p_i to a primary server in a replica group CS_i. For example, the response time is about 140 [msec] and 100 [msec] for $ncp = 1$ and $ncp = 5$, respectively, for fault probability $f = 0.05$. For $f = 0.005$, the response time is about 140 [msec] and 100 [msec] for $ncp = 1$ and $ncp = 5$, respectively. Thus, the more frequently checkpoints are taken, the shorter response time is as the total execution time. Each secondary server saves the checkpoint in the log.

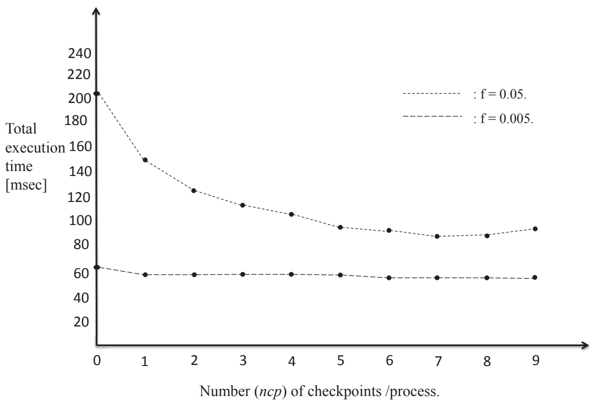

Fig. 5. Response time for number *ncp* of checkpoints.

On receipt of a checkpoint from a primary server. For example, suppose there are ten servers in a replica group CS_i. If a primary server takes a checkpoint, nine checkpoints are taken by nine secondary servers. Hence, totally ten checkpoints are taken. Figure 6 shows the total number of checkpoints taken by the servers with fault probability $f = 0.005$, 0.05, 0.1. The total number of checkpoints linearly increase as the number *ncp* of checkpoints per a server increase. The total number of checkpoints taken by the servers is measured for each number *ncp* of checkpoints taken by a process ($ncp = 1, \ldots, 9$). Each secondary server consumes the electric power to take a checkpoint. As discussed here, the more frequently checkpoints are taken, the shorter response time is. However, the more amount of electric power is consumed by servers to take checkpoints. There is trade-off between the electric power consumed by performing process replicas and taking checkpoints and response time.

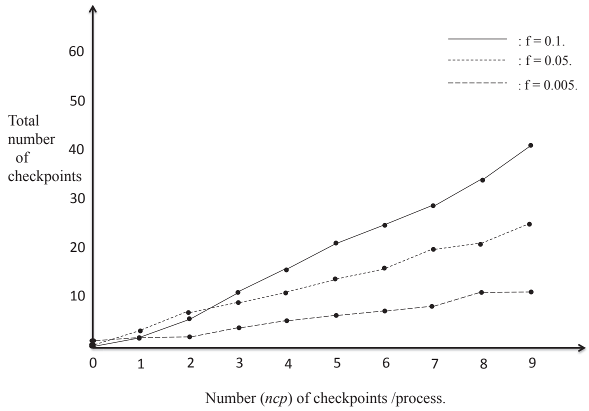

Fig. 6. Total number of checkpoints.

As shown in Figure 4, the total execution time to perform replicas decreases as the number *ncp* of checkpoints for each process to take increases. On the other hand, the total number of checkpoints taken by all the replicas of a process increases as *ncp* increases as shown in Figure 7. Suppose it takes one [msec] to take a checkpoint at each server. Figure 7 shows the total execution time to perform replicas of the process p_i and take checkpoints on servers in the cluster CS_i for fault probability $f = 0.05$ and 0.005. In this figure, if a process takes two checkpoints, the total execution time, i.e. total power consumption can be minimized for $f = 0.005$. For $f = 0.05$, the total execution time is minimum for $ncp = 6$. The total execution time for $ncp = 2$ is about 10% longer than $ncp = 6$. Thus, the number *ncp* of checkpoints for each process to take can be fixed for a given fault probability f by using Figure 7.

6 Concluding Remarks

In information systems, processes requested by clients have to be reliably and energy-efficiently performed on servers in a cluster. A process is replicated on multiple servers to increase the reliability. However, the more amount of electric power of servers is consumed to perform replicas of the process on multiple servers in a cluster. In this paper, we discussed how to reduce the total electric power consumed by servers in the passive replication of a process. Here, a process is performed only on one primary server and the primary server sends periodically sends checkpoints of the process to secondary servers. One of the secondary servers takes over the primary server if the primary server is faulty. In this paper, we discussed the energy-aware passive replication (EPR) algorithm to select a primary server and secondary servers in a cluster. We also discussed how a primary server is selected in secondary servers if a current primary server is faulty. We evaluated the energy-aware passive replication (EPR) algorithm in terms of total execution time and response time. The more number of check-

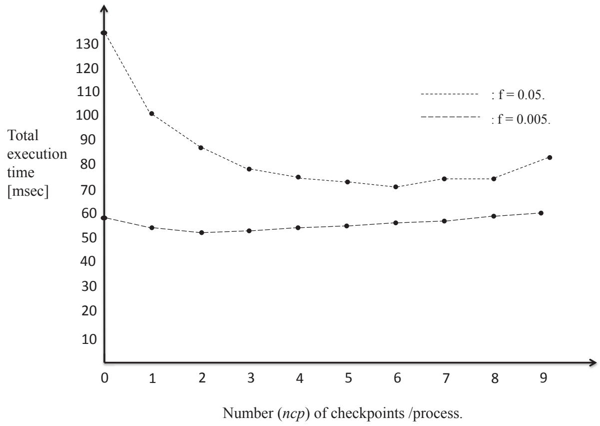

Fig. 7. Total execution time [msec] vs. number *ncp* of checkpoints.

point are taken for a process, the shorter total execution time and response time in presence of server fault. On the other hand, servers consume electric power to take checkpoints. We are now designing a power consumption model to passively replicate a process on multiple servers while taking checkpoints and recovering from primary server faults. We are also evaluating the total power consumption of servers to perform processes and take checkpoints by measuring the total electric power by the power meter [24].

References

1. P. A. Bernstein and N. Goodman (1983), The Failure and Recovery Problem for Replicated Databases, *Proc. of the 2nd ACM Symposium on Principles of Distributed Computing*, Human-centric Computing and Information Sciences, pp:114-122.
2. F. B. Scneider (1993), *Replication Management Using the State-machine Approach*, in Distributed Systems (2nd Ed.) (NY, USA), pp: 382-401.
3. R. Bianchini and R. Rajamony (2004), *Power and Energy Management for Server Systems*, Computer, vol. 37, no. 11, pp. 68–76.
4. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair (2012), *Distributed Systems Concepts and Design*, (4th ed), Addison-Wesley.
5. L. Lamport, R. Shostak, and M. Pease (2012), *The Byzantine Generals Problem*, Transactions on Programming Languages and Systems, vol.4, no.3, pp.382–401.
6. D. Duolikun, H. Hama, A. Aikebaier, T. Enokido, and M. Takizawa (2013), *Group Communication Protocols for Scalable Groups of Peers*, Proc. of the AINA-2013 workshop (WAINA-2013), pp: 1027-1032.
7. A. Aikebaier,T. Enokido, and M. Takizawa (2009), *Energy-Efficient Computation Models for Distributed Systems*, Proc. of the 12th International Conference on Network-Based Information Systems (NBiS-2009), pp:424-431.
8. T. Enokido, A. Aikebaier, S. M. Deen, and M. Takizawa (2010), *Power Consumption-based Server Selection Algorithms for Communication-based Systems.* Proc. of the 13th International Conference on Network-based Information Systems (NBiS-2010), pp:201-208,
9. T. Enokido, A. Aikebaier, and M. Takizawa (2010), *A Model for Reducing Power Consumption in Peer-to-Peer Systems*, IEEE Systems Journal, vol.4, issue.2, pp:221-22.

10. T. Enokido, K. Suzuki, A. Aikebaier, and M. Takizawa (2010), *Process Allocation Algorithm for Improving the Energy Efficiency in Distributed Systems*, Proc. of IEEE the 24th International Conference on Advanced Information Networking and Applications (AINA-2010), pp:142-149.
11. T. Enokido, A. Aikebaier, and M. Takizawa (2011), *Process Allocation Algorithms for Saving Power Consumption in Peer-to-Peer Systems*, IEEE Transactions on Industrial Electronics, vol.58, no. 6, pp:2097 - 2105.
12. T. Enokido and M. Takizawa (2012), *An Extended Power Consumption Model for Distributed Applications*, Proc. of IEEE the 26th International Conference on Advanced Information Networking and Applications (AINA-2012), pp:912 - 919.
13. T. Enokido and M. Takizawa (2013), *An Integrated Power Consumption Model for Distributed Systems*, IEEE Transactions on Industrial Electronics (TIE), vol.60, no.2, pp:824-836.
14. T. Enokido, A. Aikebaier, and M. Takizawa (2013), *An Energy-Efficient Redundant Execution Algorithm by Terminating Meaningless Redundant Processes*, Proc. of IEEE the 27th International Conference on Advanced Information Networking and Applications (AINA-2013), pp:1-8.
15. T. Enokido, A. Aikebaier, and M. Takizawa (2013), *The Evaluation of the Improved Redundant Power Consumption Laxity-Based (IRPCLB) Algorithm in Homogeneous and Heterogeneous Clusters*, Proc. of the 7th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2013), CD-ROM.
16. T. Inoue, M. Ikeda, T. Enokido, A. Aikebaier, and M. Takizawa (2011), *A Power Consumption Model for Storage-based Applications*, Proc. of the Fifth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2011), pp:612 - 617.
17. T. Inoue, T. Enokido, A. Aikebaier, and M. Takizawa (2011), *A Power Consumption Model of a Storage Server, Proc. of the 14-th International Conference on Network-Based Information Systems* (NBiS-2011), page:382 - 387, 2011.
18. T. Inoue, A. Aikebaier, T. Enokido, and M. Takizawa (2012), *Algorithms for Selecting Energy-efficient Storage Servers in Storage and Computation Oriented Applications*, Proc. of IEEE the 26th International Conference on Advanced Information Networking and Applications (AINA-2012), pp:217 - 224.
19. T. Inoue, A. Aikebaier, T. Enokido, and M. Takizawa (2012), *Energy-aware Distributed Systems for Computation and Storage-based Applications*, Proc. of the 6th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2012), pp: 269-275.
20. T. Inoue, A. Aikebaier, T. Enokido, and M. Takizawa (2013), *A Dynamic Energy-aware Server Selection Algorithm*, Proc. of IEEE the 27th International Conference on Advanced Information Networking and Applications (AINA-2013), pp: 17-24.
21. D. Powell (1991), Delta 4, *a Generic Architecture for Dependable Distributed computing*, Springer-verlag, 484 pages.
22. A. Waluyo, W. Rahayu, D. Taniar, and B. Srinivasan (2011), *A Novel Structure and Access Mechanism for Mobile Broadcast Data in Digital Ecosystems*, IEEE Transactions on Industrial Electronics, vol. 58, no. 6, pp:2173 - 2182.
23. G. Zhao, K. Xuan, W. Rahayu, D. Taniar, M. Safar, M. Gavrilova, and B. Srinivasan (2011), Voronoi-based Continuous k Nearest Neighbor Search in Mobile Navigation, IEEE Transactions on Industrial Electronics, vol. 58, no. 6, pp:2247 - 2257.
24. "UWmeter", http://www.metaprotocol.com/UWmeter /Features.html (2011), (5-7-2011).