Energy-Efficient Quorum Selection Algorithm for Distributed Object-Based Systems

Tomoya Enokido[1(✉)], Dilawaer Duolikun[2], and Makoto Takizawa[2]

[1] Faculty of Business Administration, Rissho University, Tokyo, Japan
eno@ris.ac.jp
[2] Department of Advanced Sciences, Faculty of Science and Engineering, Hosei University, Tokyo, Japan
dilewerdolkun@gmail.com, makoto.takizawa@computer.org

Abstract. Distributed applications are composed of multiple objects and each object is replicated in order to increase reliability, availability, and performance. On the other hand, the larger amount of electric energy is consumed in a system since multiple replicas of each object are manipulated on multiple servers. In this paper, the energy efficient quorum selection (EEQS) algorithm is proposed to construct a quorum for each method issued by a transaction in the quorum based locking protocol so that the total electric energy consumption of servers to perform methods can be reduced. We show the total energy consumption of servers, the average execution time of each transaction, and the number of aborted transactions can be reduced in the EEQS algorithm compared with the random algorithm in the evaluation.

Keywords: Energy-aware information systems · Quorum-based locking protocol · Object-based systems · EEQS algorithm · Data management

1 Introduction

In object-based systems [1,5], applications manipulate objects distributed on multiple servers. Each object is a unit of computation resource like a file and is an encapsulation of data and methods to manipulate the data in the object. A transaction is an atomic sequence of methods [3] to manipulate objects. A collection of conflicting transactions are required to be serializable [4] to keep objects consistent. In order to provide reliable application services [2], each object is replicated on multiple servers. Replicas of each object have to be mutually consistent. In the two-phase locking (2PL) protocol [3], one of the replicas of an object for a *read* method and all the replicas for a *write* method are locked before manipulating the object to keep the replicas mutually consistent, i.e. *read-one-write-all* scheme. However, the 2PL protocol is not efficient in write-dominated application, since all the replicas have to be locked for every write method. On the other hand, numbers nQ^r and nQ^w of replicas of an object are locked in the quorum-based protocol [5,6] for *read* and *write* methods, respectively.

L. Barolli and O. Terzo (eds.), *Complex, Intelligent, and Software Intensive Systems*, Advances in Intelligent Systems and Computing 611, DOI 10.1007/978-3-319-61566-0_4

Subsets of replicas locked for read and write methods are referred to as *read* and *write quorums*, respectively. The quorum numbers nQ^r and nQ^w have to be "$nQ^r + nQ^w > N$" where N is the total number of replicas. Here, the more number of write methods are issued, the smaller number of write quorum can be taken. As a result, the overhead to perform write methods can be reduced. On the other hand, since methods issued to each object are performed on multiple replicas, the total amount of electric energy consumed in a system is larger than non-replication systems. It is critical to not only realize the fault-tolerant application service but also reduce the total energy consumption of an object-based system as discuss in the Green computing [7,8].

In this paper, an *energy efficient quorum selection* (*EEQS*) algorithm is proposed to construct a quorum for each method issued by a transaction in the quorum based locking protocol so that the total electric energy consumption of servers to perform methods can be reduced. We evaluate the EEQS algorithm in terms of the total energy consumption of servers, the average execution time of each transaction, and the number of aborted transactions compared with the random algorithm. The evaluation results show the total energy consumption of servers, the average execution time of each transaction, and the number of aborted transactions in the EEQS algorithm can be maximumly reduced to 31%, 40%, and 65% of the random algorithm, respectively.

In Sect. 2, we discuss the data access model and power consumption model of a server. In Sect. 3, we discuss the EEQS algorithm. In Sect. 4, we evaluate the EEQS algorithm compared with random algorithm.

2 System Model

2.1 Objects and Transactions

A system is composed of multiple servers s_1, ..., s_n ($n \geq 1$) interconnected in reliable networks. That is, messages can be delivered to their destinations in the sending order and without message loss. Let S be a cluster of servers s_1, ..., s_n ($n \geq 1$). Let O be a set of objects o_1, ..., o_m ($m \geq 1$) [1]. Each object o_h is a unit of computation resource like a file and is an encapsulation of data and methods to manipulate the data in the object o_h. In this paper, we assume each object o_h supports *read* (r) and *write* (w) methods for manipulating data in the object o_h. Let $op(o_h)$ be a state obtained by performing a method op ($\in \{r, w\}$) on an object o_h. A pair of methods op_1 and op_2 on an object o_h are *compatible* if and only if (iff) $op_1 \circ op_2(o_h) = op_2 \circ op_1(o_h)$. Otherwise, a method op_1 $conflicts$ with another method op_2. For example, a pair of read methods r_1 and r_2 are compatible on an object o_h. On the other hand, a write method conflicts with read and write methods on an object o_h.

Each object o_h is replicated on multiple servers to make the system more reliable and available. Let $R(o_h)$ be a set of replicas o_h^1, ..., o_h^l ($l \geq 1$) [2] of an object o_h. Let $nR(o_h)$ be the total number of replicas of an object o_h, i.e. $nR(o_h) = |R(o_h)|$. Replicas of each object o_h are distributed on multiple servers in a server cluster S. Let S_h be a subset of servers which hold a replica of an object

o_h in a server cluster S ($S_h \subseteq S$). For example, a server cluster S is composed of five servers s_1, ..., s_5 as shown in Fig. 1. There are three objects o_1, o_2, and o_3. There are three replicas of each object o_h, i.e. $nR(o_h) = 3$ ($h = 1, ..., 3$). Here, $S_1 = \{s_1, s_2, s_5\}$ since replicas o_1^1, o_1^2, and o_1^3 of the object o_1 are stored in the servers s_1, s_2, and s_5.

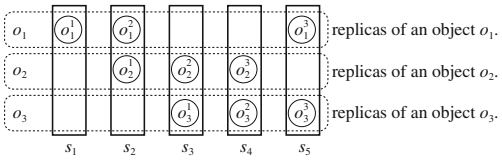

Fig. 1. A server cluster S and objects.

A *transaction* is an atomic sequence of methods [3]. A transaction T_i is initiated in a client cl_i and issues r and w methods to manipulate replicas of objects. Multiple conflicting transactions are required to be *serializable* [3,4] to keep object mutually consistent. Let $\mathbf{T}$ be a set of $\{T_1, ..., T_k\}$ ($k \geq 1$) of transactions. Let H be a schedule of the transactions in $\mathbf{T}$, i.e. a sequence of methods performed in $\mathbf{T}$. A transaction T_i *precedes* another transaction T_j ($T_i \rightarrow_H T_j$) in a schedule H iff a method op_i from the transaction T_i is performed before a method op_j from the transaction T_j and op_i conflicts with op_j. A schedule H is serializable iff the precedent relation $\rightarrow_H$ is acyclic.

2.2 Quorum-Based Locking Protocol

In this paper, multiple conflicting transactions are serialized by using the *quorum-based locking* protocol [5,6]. Let Q_h^{op} ($op \in \{r, w\}$) be a subset of replicas of an object o_h to be locked by a method op, named a *quorum* of the method op on the object o_h ($Q_h^{op} \subseteq R(o_h)$). Let nQ_h^{op} be the *quorum number* of a method op on a object o_h, i.e. $nQ_h^{op} = |Q_h^{op}|$. The quorums have to satisfy the following constraints: (1) $Q_h^r \subseteq R(o_h)$, $Q_h^w \subseteq R(o_h)$, and $Q_h^r \cup Q_h^w = R(o_h)$. (2) $nQ_h^r + nQ_h^w > nR(o_h)$, i.e. $Q_h^r \cap Q_h^w \neq \phi$. (3) $nQ_h^w > nR(o_h)/2$. Let $\mu(op)$ be a *lock mode* of a method op ($\in \{r, w\}$). If op_1 is compatible with op_2 on an object o_h, the lock mode $\mu(op_1)$ is compatible with $\mu(op_2)$. Otherwise, a lock mode $\mu(op_1)$ conflicts with another lock mode $\mu(op_2)$.

A transaction T_i locks replicas of an object o_h by using the following quorum-based locking protocol [5] before manipulating the replicas with a method op.

[Quorum-based locking protocol]

1. A quorum Q_h^{op} for a method op is constructed by selecting nQ_h^{op} replicas in a set $R(o_h)$ of replicas.
2. If every replica in a quorum Q_h^{op} can be locked by a lock mode $\mu(op)$, the replicas in the quorum Q_h^{op} are manipulated by the method op.

3. When the transaction T_i commits or aborts, the locks on the replicas in the quorum Q_h^{op} are released.

Each replica o_h^q has a *version number* v_h^q. Suppose a transaction T_i reads an object o_h. The transaction T_i selects nQ_h^r replicas in the set $R(o_h)$, i.e. *read* (r) quorum Q_h^r. If every replica in the r-quorum Q_h^r can be locked by a lock mode $\mu(r)$, the transaction T_i reads data in a replica o_h^q whose version number v_h^q is the maximum in the r-quorum Q_h^r. Every r-quorum surely includes at least one newest replica since $nQ_h^r + nQ_h^w > nR(o_h)$. Next, suppose a transaction T_i writes data in an object o_h. The transaction T_i selects nQ_h^w replicas in the set $R(o_h)$, i.e. *write* (w) quorum Q_h^w. If every replica in the w-quorum Q_h^w can be locked by a lock mode $\mu(w)$, the transaction T_i writes data in a replica o_h^q whose version number v_h^q is maximum in the w-quorum Q_h^w and the version number v_h^q of the replica o_h^q is incremented by one. The updated data and version number v_h^q of the replica o_h^q are sent to every other replica in the w-quorum Q_h^w. Then, data and version number of each replica in the w-quorum Q_h^w are replaced with the newest values. When a transaction T_i commits or aborts, the locks on every replica in a quorum Q_h^{op} ($op \in \{r, w\}$) are released.

2.3 Data Access Model

Methods which are being performed and already terminate are *current* and *previous* at time τ, respectively. Let $RP_t(\tau)$ and $WP_t(\tau)$ be sets of current *read* (r) and *write* (w) methods on a server s_t at time τ, respectively. Let $P_t(\tau)$ be a set of current r and w methods on a server s_t at time τ, i.e. $P_t(\tau) = RP_t(\tau) \cup WP_t(\tau)$. Let $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$ be methods issued by a transaction T_i to read and write data in a replica o_h^q on a server s_t, respectively. By each method $r_{ti}(o_h^q)$ in a set $RP_t(\tau)$, data is read in a replica o_h^q at rate $RR_{ti}(\tau)$ [B/sec] at time τ. By each method $w_{ti}(o_h^q)$ in a set $WP_t(\tau)$, data is written in a replica o_h^q at rate $WR_{ti}(\tau)$ [B/sec] at time τ. Let $maxRR_t$ and $maxWR_t$ be the maximum read and write rates [B/sec] of r and w methods on a server s_t, respectively. The read rate $RR_{ti}(\tau)$ ($\leq maxRR_t$) and write rate $WR_{ti}(\tau)$ ($\leq maxWR_t$) are given as follows:

$$RR_{ti}(\tau) = fr_t(\tau) \cdot maxRR_t. \qquad WR_{ti}(\tau) = fw_t(\tau) \cdot maxWR_t. \tag{1}$$

Here, $fr_t(\tau)$ and $fw_t(\tau)$ are degradation ratios. $0 \leq fr_t(\tau) \leq 1$ and $0 \leq fw_t(\tau) \leq 1$. The degradation ratios $fr_t(\tau)$ and $fw_t(\tau)$ are given as follows:

$$fr_t(\tau) = \frac{1}{|RP_t(\tau)| + rw_t \cdot |WP_t(\tau)|}. \quad fw_t(\tau) = \frac{1}{wr_t \cdot |RP_t(\tau)| + |WP_t(\tau)|}. \tag{2}$$

Here, $0 \leq rw_t \leq 1$ and $0 \leq wr_t \leq 1$.

The *read laxity* $lr_{ti}(\tau)$ [B] and *write laxity* $lw_{ti}(\tau)$ [B] of methods $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$ show how much amount of data are read and written in a replica o_h^q by the methods $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$ at time τ, respectively. Suppose that methods $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$ start on a server s_t at time st_{ti}, respectively. At time st_{ti},

the read laxity $lr_{ti}(\tau) = rb_h^q$ [B] where rb_h^q is the size of data in a replica o_h^q. The write laxity $lw_{ti}(\tau) = wb_h^q$ [B] where wb_h^q is the size of data to be written in a replica o_h^q. The read laxity $lr_{ti}(\tau)$ and write laxity $lw_{ti}(\tau)$ at time τ are given as $lr_{ti}(\tau) = rb_h^q - \Sigma_{\tau=st_{ti}}^{\tau} RR_{ti}(\tau)$ and $lw_{ti}(\tau) = wb_h^q - \Sigma_{\tau=st_{ti}}^{\tau} WR_{ti}(\tau)$, respectively.

2.4 Power Consumption Model of a Server

Let $E_t(\tau)$ be the electric power [W] of a server s_t at time τ. $maxE_t$ and $minE_t$ show the maximum and minimum electric power [W] of the server s_t, respectively. The *power consumption model for a storage server* (PCS model) [7] to perform storage and computation process are proposed. In this paper, we assume only r and w methods are performed on a server s_t. According to the PCS model, the electric power $E_t(\tau)$ [W] of a server s_t to perform multiple r and w methods at time τ is given as follows:

$$E_t(\tau) = \begin{cases} WE_t & \text{if } |WP_t(\tau)| \geq 1 \text{ and } |RP_t(\tau)| = 0. \\ WRE_t(\alpha) & \text{if } |WP_t(\tau)| \geq 1 \text{ and } |RP_t(\tau)| \geq 1. \\ RE_t & \text{if } |WP_t(\tau)| = 0 \text{ and } |RP_t(\tau)| \geq 1. \\ minE_t & \text{if } |WP_t(\tau)| = |RP_t(\tau)| = 0. \end{cases} \tag{3}$$

A server s_t consumes the minimum electric power $minE_t$ [W] if no method is performed on the server s_t, i.e. the electric power in the idle state of the server s_t. The server s_t consumes the electric power RE_t [W] if $|WP_t(\tau)| = 0$ and $|RP_t(\tau)| \geq 1$, i.e. only and at least one r method is performed on the server s_t. The server s_t consumes the electric power WE_t [W] if $|WP_t(\tau)| \geq 1$ and $|RP_t(\tau)| = 0$, i.e. only and at least one w method is performed on the server s_t. The server s_t consumes the electric power $WRE_t(\alpha)$ [W] $= \alpha \cdot RE_t + (1 - \alpha) \cdot WE_t$ [W] where $\alpha = |RP_t(\tau)| \ / \ (|RP_t(\tau)| + |WP_t(\tau)|)$ if $|WP_t(\tau)| \geq 1$ and $|RP_t(\tau)| \geq 1$, i.e. both at least one r method and at least one w method are concurrently performed. Here, $minE_t \leq RE_t \leq WRE_t(\alpha) \leq WE_t \leq maxE_t$.

The total energy consumption $TE_t(\tau_1, \tau_2)$ [J] of a server s_t from time τ_1 to τ_2 is $\Sigma_{\tau=\tau 1}^{\tau_2} E_t(\tau)$. The processing power $PE_t(\tau)$ [W] of a server s_t at time τ is $E_t(\tau) - minE_t$. The total processing energy consumption $TPE_t(\tau_1, \tau_2)$ of a server s_t from time τ_1 to τ_2 is given as $TPE_t(\tau_1, \tau_2) = \Sigma_{\tau=\tau 1}^{\tau_2} PE_t(\tau)$. The total processing energy consumption laxity $tpecl_t(\tau)$ shows how much electric energy a server s_t has to consume to perform every current r and w methods on the server s_t at time τ. The total processing energy consumption laxity $tpecl_t(\tau)$ of a server s_t at time τ is obtained by the following $\boldsymbol{TPECL_t}$ procedure:

$\boldsymbol{TPECL}_t(\tau)$ {
if $RP_t(\tau) = \phi$ and $WP_t(\tau) = \phi$, **return**(0);
$laxity = E_t(\tau)$ - $minE_t$; /* $PE_t(\tau)$ of a server s_t at time τ */
for each r-method $r_{ti}(o_h^q)$ in $RP_t(\tau)$, {
  $lr_{ti}(\tau + 1) = lr_{ti}(\tau)$ - RR_{ti};
  if $lr_{ti}(\tau + 1) = 0$, $RP_t(\tau + 1) = RP_t(\tau)$ - $\{r_{ti}(o_h^q)\}$;

}
for each w-method $w_{ti}(o_h^q)$ in $WP_t(\tau)$, {
$lw_{ti}(\tau + 1) = lw_{ti}(\tau)$ - WR_{ti};
if $lw_{ti}(\tau + 1) = 0$, $WP_t(\tau + 1) = WP_t(\tau)$ - $\{w_{it}(o_h^q)\}$;
}
return($laxity$ + $\boldsymbol{TPECL_t}(\tau + 1)$);
}

In the $\boldsymbol{TPECL_t}$ procedure, each time τ data is read in a replica o_h^q by a method $r_{ti}(o_h^q)$, the read laxity $lr_{ti}(\tau)$ of the method $r_{ti}(o_h^q)$ is decremented by read rate RR_{ti}. Similarly, the write laxity $lw_{ti}(\tau)$ of a method $w_{ti}(o_h^q)$ is decremented by write rate WR_{ti} each time τ data is written in a replica o_h^q by the method $w_{ti}(o_h^q)$. If the read laxity $lr_{ti}(\tau + 1)$ and write laxity $lw_{ti}(\tau + 1)$ get 0, every data is read and written in the replica o_h^q by the methods $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$, respectively, and the methods terminate at time τ.

3 Quorum Selection Algorithm

We propose an *energy-efficient quorum selection* (*EEQS*) algorithm to select replicas to be members of a quorum of each method in the quorum-based locking protocol so that the total energy consumption of a server cluster S to perform read and write methods can be reduced. Suppose a transaction T_i issues a method op ($op = \{r, w\}$) to manipulate an object o_h at time τ. Each transaction T_i selects a subset S_h^{op} ($\subseteq S_h$) of nQ_h^{op} servers in a subset S_h by following ***EEQS*** procedure:

EEQS(op, o_h, τ) { /* $op \in \{r, w\}$ */
$S_h^{op} = \phi$;
while ($nQ_h^{op} > 0$) {
for each server s_t in S_h, {
if $op = r$, $RP_t(\tau) = RP_t(\tau) \cup \{op\}$;
else $WP_t(\tau) = WP_t(\tau) \cup \{op\}$; /* $op = w$ */
$TPE_t(\tau) = \boldsymbol{TPECL_t}(\tau)$;
}
$server$ = a server s_t where $TPE_t(\tau)$ is the minimum;
$S_h^{op} = S_h^{op} \cup \{server\}$; $S_h = S_h$ - $\{server\}$; $nQ_h^{op} = nQ_h^{op}$ - 1;
}
return(S_h^{op});
}

Suppose a server cluster S is composed of five servers s_1, ..., s_5 and replicas of three objects o_1, o_2, and o_3 are distributed on multiple servers in the server cluster S as shown in Fig. 1, i.e. $S_1 = \{s_1, s_2, s_5\}$, $S_2 = \{s_2, s_3, s_4\}$, and $S_3 = \{s_3, s_4, s_5\}$. Every server s_t ($t = 1, ..., 5$) follows the same data access model and power consumption model as shown in Table 1. The size of data in every object o_h ($h = 1, ..., 3$) is 80 [MB]. There are three replicas for each object o_h, i.e. $nR(o_h) = 3$. The quorum numbers nQ_h^w and nQ_h^r for every object o_h are two, i.e. $nQ_h^w = nQ_h^r = 2$.

Table 1. Homogeneous cluster S

Server s_t	$maxRR_t$	$maxWR_t$	rw_t	wr_t	$minE_t$	WE_t	RE_t
s_t	80 [MB/sec]	45 [MB/sec]	0.5	0.5	39 [W]	53 [W]	43 [W]

At time τ_0, a pair of replicas o_1^1 and o_1^3 stored in the servers s_1 and s_5 are being locked by a transaction T_1 with a lock mode $\mu(w)$ and a pair of write methods $w_{11}(o_1^1)$ and $w_{51}(o_1^3)$ are being performed on the servers s_1 and s_5, respectively, as shown in Fig. 2. Let $T_i.Q_h^{op}$ be a quorum to perform a method op issued by a transaction T_i. Let $T_i.S_h^{op}$ be a subset of servers which hold replicas in a quorum $T_i.Q_h^{op}$ constructed by a transaction T_i. The w-quorum $T_1.Q_1^w$ is $\{o_1^1, o_1^3\}$ since the quorum number $nQ_1^w = 2$. The subset $T_1.S_1^w$ is $\{s_1, s_5\}$ since a pair of replicas o_1^1 and o_1^3 are stored in the servers s_1 and s_5, respectively. A pair of write laxities $lw_{11}(\tau_0)$ and $lw_{51}(\tau_0)$ are 45 [MB], respectively, at time τ_0.

Suppose a transaction T_2 issues a write method to the object o_3 at time τ_0. The size of data to be written in the object o_3 by the write method issued by the transaction T_2 is 45 [MB], i.e. the write laxity $lw_{t2}(\tau_0) = 45$ [MB]. Here, $R(o_3) = \{o_3^1, o_3^2, o_3^3\}$ and $S_3 = \{s_3, s_4, s_5\}$ as shown in Fig. 1. First, the transaction T_2 constructs a w-quorum $T_2.Q_3^w$ by the procedure **EEQS**(w, o_3, τ_0). Suppose a write method $w_{32}(o_3^1)$ is issued to a replica o_3^1 stored in the server s_3 at time τ_0. No method is performed on the server s_3 at time τ_0. Hence, $WP_3(\tau_0) = WP_3(\tau_0) \cup \{w_{32}(o_3^1)\} = \{w_{32}(o_3^1)\}$. Since only one write method $w_{32}(o_3^1)$ is performed on the server s_3 at time τ_0, the degradation ratio $fw_3(\tau_0)$ is $1/(wr_3 \cdot |RP_3(\tau_0)| + |WP_3(\tau_0)|) = 1/(0.5 \cdot 0 + 1) = 1$ and the write method $w_{32}(o_3^1)$ is performed on the server s_3 at write rate $WR_{32}(\tau_0) = fw_3(\tau_0) \cdot maxWR_3 = 1 \cdot 45 = 45$ [MB/sec]. Hence, the write laxity $lw_{32}(\tau_1)$ gets 0 since $lw_{32}(\tau_0) - WR_{32}(\tau_0) = 45$ [MB] $- 45$ [MB] $= 0$ at time τ_1. Here, the write method $w_{32}(o_3^1)$ terminates at time τ_1 and no method is performed after time τ_1. Similarly, if a write method $w_{42}(o_3^2)$ is issued to a replica o_3^2 stored in the server s_4 at time τ_0 as shown in Fig. 2, the write method $w_{42}(o_3^2)$ terminates at time τ_1 since no method is performed on the server s_4 at time τ_0. Suppose a write method $w_{52}(o_3^3)$ is issued to a replica o_3^3 stored in the server s_5 at time τ_0. Here, a pair of write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ are concurrently performed on the server s_5 at time τ_0, i.e. $WP_5(\tau_0) = \{w_{51}(o_1^3), w_{52}(o_3^3)\}$ and $|WP_5(\tau_0)| = 2$. Here, the degradation ratio $fw_5(\tau_0)$ is $1/(wr_5 \cdot |RP_5(\tau_0)| + |WP_5(\tau_0)|) = 1/(0.5 \cdot 0 + 2) = 0.5$. A pair of the write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ are concurrently performed on the server s_5 at write rate $WR_{51}(\tau_0) = WR_{52}(\tau_0) = fw_5(\tau_0) \cdot maxWR_5 = 0.5 \cdot 45 = 22.5$ [MB/sec], respectively. Hence, the write laxity $lw_{51}(\tau_1)$ is 22.5 [MB/sec] at time τ_1 since $lw_{51}(\tau_0) - WR_{51}(\tau_0) = 45$ [MB] $- 22.5$ [MB] $= 22.5$ [MB]. Similarly, the write laxity $lw_{52}(\tau_1)$ is 22.5 [MB] at time τ_1. At time τ_1, a pair of the write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ are still concurrently performed on the server s_5 at write rate 22.5 [MB/sec]. The write laxity $lw_{51}(\tau_2)$ gets 0 at time τ_2 since $lw_{51}(\tau_1) - WR_{51}(\tau_1) = 22.5$ [MB] $- 22.5$ [MB] $= 0$. Similarly, the write laxity $lw_{52}(\tau_2)$ gets 0 at time τ_1. Here, a pair of write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ terminate at time τ_2.

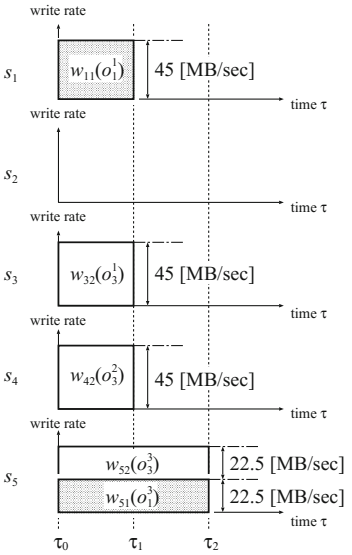

Fig. 2. Example of method execution.

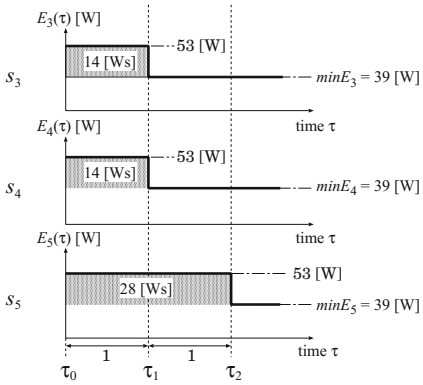

Fig. 3. Total processing energy consumption laxity [J].

Figure 3 shows the electric power [W] of the servers s_3, s_4, and s_5 to perform the write methods as shown in Fig. 2. The electric power $E_t(\tau)$ [W] of a server s_t at time τ is given in formula (3). At time τ_0 to τ_1, only the write method $w_{32}(o_3^1)$ is performed on the server s_3, i.e. $|WR_3(\tau_0)| = 1$ and $|RP_3(\tau_0)| = 0$. Hence, the electric power consumption $E_3(\tau_0) = WE_3 = 53$ [W]. Similarly, $E_4(\tau_0) = WE_4 = 53$ [W] in the server s_4. In the server s_5, only a pair of write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ are performed, i.e. $|WR_5(\tau_0)| = 2$ and $|RP_5(\tau_0)| = 0$. Hence, the electric power consumption $E_5(\tau_0) = WE_5 = 53$ [W]. The total processing power consumption $TPE_3(\tau_0, \tau_1)$ is $E_3(\tau_0) - minE_3 = 53 - 39 = 14$ [W]. Similarly, $TPE_4(\tau_0, \tau_1)$ and $TPE_5(\tau_0, \tau_1)$ are 14 [W], respectively. At time τ_1 to τ_2, a pair of write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ are performed on the server s_5, i.e. $|WR_5(\tau_1)| = 2$ and $|RP_5(\tau_1)| = 0$. Hence, $E_5(\tau_1) = WE_5 = 53$ [W] and $TPE_5(\tau_1, \tau_2) = 53 - 39 = 14$ [W].

The hatched area shows the total processing energy consumption laxity $tpecl_t(\tau_0)$ [J] of each server s_t ($t = \{3, 4, 5\}$) where the write method $w_{t2}(o_3)$ issued by the transaction T_2 is performed on the server s_t at time τ_0. Here, $tpecl_3(\tau_0) = TPE_3(\tau_0, \tau_1) = 14$ [J]. $tpecl_4(\tau_0) = TPE_4(\tau_0, \tau_1) = 14$ [J]. $tpecl_5(\tau_0) = TPE_5(\tau_0, \tau_1) + TPE_5(\tau_1, \tau_2) = 14 + 14 = 28$ [J]. Here, a w-quorum T_2,Q_3^w is constructed by a pair of replicas o_3^1 and o_3^2 stored in the servers s_3 and s_4 since $nQ_3^w = 2$ and $tpecl_3(\tau_0) = tpecl_4(\tau_0) < tpecl_5(\tau_0)$, i.e. $T_2.Q_3^w = \{o_3^1, o_3^2\}$ and $T_2.S_3^w = \{s_3, s_4\}$.

4 Evaluation

4.1 Environment

We evaluate the EEQS algorithm in terms of the total energy consumption of a server cluster S, the average execution time of each transaction, and the average number of aborted transactions compared with the random algorithm. In the random algorithm, a quorum for each method is randomly selected. In this evaluation, a homogeneous server cluster S which is composed of ten homogeneous servers s_1, ..., s_{10} ($n = 10$) is considered. In the server cluster S, every server s_t ($t = 1, ..., 10$) follows the same data access model and power consumption model as shown in Table 1. Parameters of each server s_t are given based on the experimentations [7]. There are fifty objects o_1, ..., o_{50} in a system, i.e. $O = \{o_1, ..., o_{50}\}$. The size of data in each object o_h is randomly selected between 50 and 100 [MB]. Each object o_h supports *read* (r) and *write* (w) methods. The total number of replicas for every object is five, i.e. $nR(o_h) = 5$ and $R(o_h) = \{o_h^1, ..., o_h^5\}$ ($h = 1, ..., 50$). Replicas of each object are randomly distributed on five servers in the server cluster S. The quorum number nQ_h^w of a w method on every object o_h is three, i.e. $nQ_h^w = 3$. The quorum number nQ_h^r of a r method on every object o_h is three, $nQ_h^r = 3$.

The number m of transactions are issues to manipulate objects in a system. Each transaction issues three methods randomly selected from one-hundred methods on the fifty objects. By each r and w method issued by a transaction T_i to a replica o_h^q of an object o_h, the total amount of data of the replica o_h^q are fully read and written, respectively. The starting time of each transaction T_i is randomly selected in a unit of one second between 1 and 360 [sec].

4.2 Average Execution Time of Each Transaction

We evaluate the EEQS algorithm in terms of the average execution time [sec] of each transaction. Let ET_i be the execution time [sec] of a transaction T_i where the transaction T_i commits. For example, suppose a transaction T_i starts at time st_i and commits at time et_i. Here, the execution time ET_i of the transaction T_i is $et_i - st_i$ [sec]. The execution time ET_i for each transaction T_i is measured ten times for each total number m of transactions ($0 \leq m \leq 500$). Let ET_i^{tm} be the execution time ET_i obtained in tm-th simulation. The average execution time AET [sec] of each transaction for each total number m of transactions is $\sum_{tm=1}^{10} \sum_{i=1}^{m} ET_i^{tm}/(m \cdot 10)$.

Figure 4 shows the average execution time AET [sec] in the server cluster S to perform the total number m of transaction in the EEQS and random algorithms. In the EEQS and random algorithms, the average execution time AET increases as the total number m of transactions increases since more number of transactions are concurrently performed. For $0 < m \leq 500$, the average execution time AET can be more shorter in the EEQS algorithm than the random algorithm. This means that the data access resources in the server cluster S can be more efficiently utilized in the EEQS algorithm than the random algorithm.

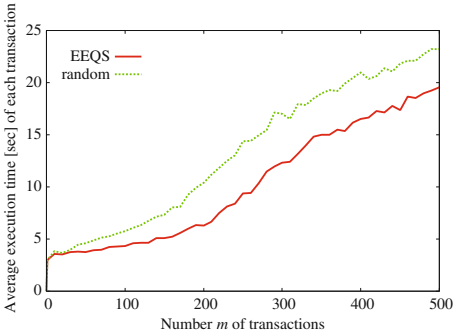

Fig. 4. Average execution time AET [sec] of each transaction.

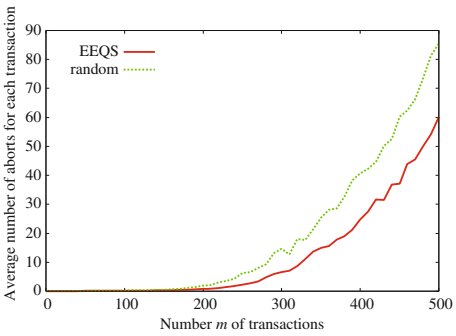

Fig. 5. Average number of aborts for each transaction

4.3 Average Number of Aborted Transaction Instances

If a transaction T_i could not lock every replica in an r-quorum Q_h^r and w-quorum Q_h^w, the transaction T_i aborts. Then, the transaction T_i is restarted after δ time units. The time units δ [sec] is randomly selected between twenty and thirty seconds in this evaluation. Every transaction T_i is restarted until the transaction T_i commits. Each execution of a transaction is referred to as transaction *instance*. We measure how many number of transaction instances are aborted until each transaction commits. Let AT_i be the number of aborted instances of a transaction T_i. The number of aborted instances AT_i for each transaction T_i is measured ten times for each total number m of transactions ($0 \leq m \leq 500$). Let AT_i^{tm} be the number of aborted transaction instances AT_i obtained in tmth simulation. The average number of aborted instances AAT of each transaction for each total number m of transactions is $\sum_{tm=1}^{10} \sum_{i=1}^{m} AT_i^{tm}/(m \cdot 10)$.

Figure 5 shows the average number of aborted transaction instances AAT in the server cluster S to perform the total number m of transactions in the EEQS and random algorithms. In the EEQS and random algorithms, the average number of aborted transaction instances AAT increases as the total number m of transactions increases. The more number of transactions are concurrently performed, the more number of transactions cannot lock replicas. Hence, the number of aborted transactions instance increases in the EEQS and random algorithms. For $0 < m \leq 500$, the average number of aborted instances AAT of each transaction can be more reduced in the EEQS algorithm than the random algorithm. The data access resources in the server cluster S can be more efficiently utilized in the EEQS algorithm than the random algorithm. Hence, the average execution time of each transaction can be shorter in the EEQS algorithm than the random algorithm. As a result, the number of aborted transactions can be more reduced in the EEQS algorithm than the random algorithm since the number of transaction to be concurrently performed can be reduced.

4.4 Average Total Energy Consumption of a Server Cluster

We evaluate the EEQS algorithm in terms of the average total energy consumption [J] of the homogeneous server cluster S to perform the number m of transactions. Let TEC_{tm} be the total energy consumption [J] to perform the number m of transactions ($0 \leq m \leq 500$) in the server cluster S obtained in the tm-th simulation. The total energy consumption TEC_{tm} is measured ten times for each number m of transactions. Then, the average total energy consumption $ATEC$ [J] of the server cluster S is calculated as $\sum_{tm=1}^{10} TEC_{tm}/10$ for each number m of transactions.

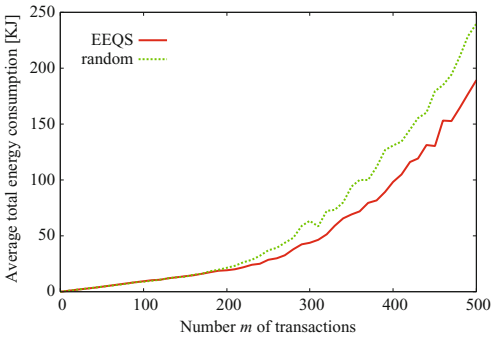

Fig. 6. Average total energy consumption (ATEC) [KJ].

Figure 6 shows the average total energy consumption $ATEC$ of the server cluster S to perform the number m of transactions in the EEQS and random algorithms. In the EEQS and random algorithms, the average total energy consumption $ATEC$ of the server cluster S increases as the number m of transactions increases. For $0 \leq m \leq 500$, the average total energy consumption $ATEC$ of the server cluster S can be more reduced in the EEQS algorithm than the random algorithm. In the EEQS algorithm, each time a transaction T_i issues a method op ($\in \{r, w\}$) to an object o_h, the transaction T_i selects a subset nS_h^{op} ($\subseteq S_h$) of nQ_h^{op} servers which hold a replica o_h^q of the object o_h so that the total processing energy consumption laxity of a server cluster S is the minimum. In addition, the average execution time and the number of aborted instances of each transaction can be more reduced in the EEQS algorithm than the random algorithm. As a result, the average total energy consumption $ATEC$ of the server cluster S to perform the number m of transactions can be more reduced in the EEQS algorithm than the random algorithm.

Following the evaluation, the total energy consumption of a server cluster, the average execution time of each transaction, and the number of aborted transactions in the EEQS algorithm can be maximumly reduced to 31%, 40%, and 65% of the random algorithm, respectively. Hence, the EEQS algorithm is more useful than the random algorithm.

5 Concluding Remarks

In this paper, we newly proposed the EEQS algorithm to select a quorum for each method issued by a transaction in the quorum based locking protocol so that the total energy consumption of a server cluster to perform methods issued by transactions can be reduced. We evaluated the EEQS algorithm in terms of the total energy consumption of a server cluster, the average execution time of each transaction, and the number of aborted transactions compared with the random algorithm. The evaluation results show the average total energy consumption of a server cluster, the average execution time of each transaction, and the average number of aborted transaction instances can be more reduced in the EEQS algorithm than the random algorithm. Hence, the EEQS algorithm is more useful than the random algorithm.

References

1. Object Management Group Inc.: Common object request broker architecture (CORBA) specification, version 3.3, part 1 - interfaces (2012). http://www.omg.org/spec/CORBA/3.3/Interfaces/PDF
2. Schneider, F.B.: Replication Management Using the State-Machine Approach. Distributed Systems. ACM Press, New York (1993)
3. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, Boston (1987)
4. Gray, J.N.: Notes on data base operating systems. In: Bayer, R., Graham, R.M., Seegmüller, G. (eds.) Operating Systems. LNCS, vol. 60, pp. 393–481. Springer, Heidelberg (1978)
5. Tanaka, K., Hasegawa, K., Takizawa, M.: Quorum-based replication in object-based systems. J. Inf. Sci. Eng. **16**(3), 317–331 (2000)
6. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. J. ACM **32**(4), 814–860 (1985)
7. Sawada, A., Kataoka, H., Duolikun, D., Enokido, T., Takizawa, M.: Energy-aware clusters of servers for storage and computation applications. In: Proceedings of the 30th IEEE International Conference on Advanced Information Networking and Applications (AINA-2016), pp. 400–407 (2016)
8. Natural Resources Defense Council (NRDS): Data center efficiency assessment - scaling up energy efficiency across the data center lndustry: Evaluating key drivers and barriers (2014). http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf