Improved Energy-Efficient Quorum Selection Algorithm by Omitting Meaningless Methods

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 our previous studies, the energy efficient quorum selection (EEQS) algorithm is proposed to construct a quorum for each method in the quorum based locking protocol so that the total electric energy of servers to perform methods can be reduced. In this paper, the improved energy efficient quorum selection (IEEQS) algorithm is proposed to furthermore reduce the total electric energy of servers by omitting meaningless methods. Evaluation results show the total electric energy of servers, the average execution time of each transaction, and the number of aborted transactions can be reduced in the IEEQS algorithm than the EEQS algorithm.

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

1 Introduction

In object-based systems [1,6], 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. In order to provide reliable application services [2,3], each object is replicated on multiple servers. A transaction is an atomic sequence of methods [4] to manipulate objects. Conflicting methods issued by multiple transactions have to be serialized [5] to keep the replicas of each object mutually consistent. In the two-phase locking (2PL) protocol [4], 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. Since all the replicas have to be locked for every write method, the 2PL protocol is not efficient

L. Barolli et al. (Eds.): EIDWT 2018, LNDECT 17, pp. 171–182, 2018.
https://doi.org/10.1007/978-3-319-75928-9_15

in write-dominated application. In the quorum-based protocol [6,7], 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 for read and write methods have to be "$nQ^r + nQ^w > N$" where N is the total number of replicas. In the quorum-based protocol, the more number of write methods are issued, the fewer number of write quorums can be taken. As a result, the overhead to perform write methods can be reduced. However, the total amount of electric energy consumed in a system is larger than non-replication systems since each method issued to an object is performed on multiple replicas. In our previous studies, the *energy efficient quorum selection* ($EEQS$) algorithm [9] is proposed to construct a quorum for each method issued by a transaction so that the total electric energy of servers to perform the method is the minimum. Here, the total electric energy of a server cluster can be reduced in the EEQS algorithm than the traditional quorum-based locking protocol.

In this paper, we first define meaningless methods which are not required to be performed on each replica of an object based on the precedent relation and semantics of methods. Next, the *improved energy efficient quorum selection* ($IEEQS$) algorithm is proposed to furthermore reduce the total electric energy of a server cluster to perform methods by omitting meaningless methods on each replica. We evaluate the IEEQS algorithm compared with the EEQS algorithm. The evaluation results show the total electric energy of a server cluster, the average execution time of each transaction, and the number of aborted transactions in the IEEQS algorithm can be more reduced than the EEQS algorithm.

In Sect. 2, we discuss the data access model and power consumption model of a server. In Sect. 3, we discuss the IEEQS algorithm. In Sect. 4, we evaluate the IEEQS algorithm compared with the EEQS 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. 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, methods are classified into *read* (r) and *write* (w) methods. Write methods are furthermore classified into *full* write (w^f) and *partial* write (w^p) methods, i.e. $w \in \{w^f, w^p\}$. In a full write method, a whole data in an object is fully written. In a partial write method, a part of data in an object is written. Suppose a file object F supports *modify*, *insert*, *delete*, and *read* methods. Here, a modify method is a full write method. Insert and delete methods are partial write methods. 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. In this paper, conflicting relations among methods are as shown in Table 1.

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 $(1 \leq l \leq n)$ [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 ($S_h \subseteq S$).

Table 1. Conflicting relation among methods.

		read (r)	write (w)	
			full (w^f)	partial (w^p)
read (r)		Compatible	Conflict	Conflict
write (w)	full (w^f)	Conflict	Conflict	Conflict
	partial (w^p)	Conflict	Conflict	Conflict

2.2 Quorum-Based Locking Protocol

A *transaction* is an atomic sequence of methods [4]. A transaction T_i issues r and w methods to manipulate replicas of objects. Multiple conflicting transactions are required to be *serializable* [4,5] to keep replicas of each 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 $\mathbf{T}$. A transaction T_i *precedes* another transaction T_j ($T_i \rightarrow_H T_j$) in a schedule H iff (if and only if) 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 [4].

In this paper, multiple conflicting transactions are serialized based on the *quorum-based locking* protocol [6,7]. Let $\mu(op)$ be a *lock mode* of a method op ($\in \{r, w\}$). In this paper, a lock mode $\mu(w)$ is adapted to a full and partial write methods. A lock mode $\mu(r)$ is adapted to a read method. 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 $\mu(op_2)$. 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 ($Q_h^{op} \subseteq R(o_h)$). 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) $nQ_h^w > nR(o_h)/2$.

In the quorum-based locking protocol, a transaction T_i locks replicas of an object o_h by the following procedure [6]:

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.

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 $fr_t(\tau) \cdot maxRR_t$ and $fw_t(\tau) \cdot maxWR_t$, respectively. 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 $1/(|RP_t(\tau)| + rw_t \cdot |WP_t(\tau)|)$ and $1/(wr_t \cdot |RP_t(\tau)| + |WP_t(\tau)|)$, respectively. $0 \leq rw_t \leq 1$ 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 rb_h^q - $\Sigma_{\tau=st_{ti}}^{\tau} RR_{ti}(\tau)$ and 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) [8] to perform storage and computation processes 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:

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

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 at least one r method is performed on the server s_t. The server s_t consumes the electric power WE_t [W] if 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 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 electric energy $TE_t(\tau_1, \tau_2)$ [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 electric energy $TPE_t(\tau_1, \tau_2)$ 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 electric energy 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:

$\boldsymbol{TPECL_t}(\boldsymbol{\tau})$ {
  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 end */
  for each w-method $w_{ti}(o_h^q)$ in $WP_t(\tau)$, {
    $lw_{ti}(\tau + 1) = lw_{ti}(\tau) - WR_{ti}$;
    if $lw_{ti}(\tau + 1) = 0$, $WP_t(\tau + 1) = WP_t(\tau) - \{w_{it}(o_h^q)\}$;
  } /* for end */
  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 Improved EEQS (IEEQS) Algorithm

3.1 Quorum Selection

In the *improved energy-efficient quorum selection* (*IEEQS*) algorithm, replicas to be members of a quorum of each method are selected so that the total electric energy of a server cluster S to perform the method is the minimum. 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 whose total processing electric energy laxity is the minimum for each method op by following quorum selection ($\boldsymbol{QS}$) procedure [9]:

QS(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 $WP_t(\tau) = WP_t(\tau) \cup \{op\}$; /* $op = w$ */
      $TPE_t(\tau) = \boldsymbol{TPECL_t}(\tau)$;
    } /* for end */
    $server$ = a server s_t where $TPE_t(\tau)$ is the minimum;
    $S_h^{op} = S_h^{op} \cup \{server\}$; $S_h = S_h - \{server\}$; $nQ_h^{op} = nQ_h^{op} - 1$;
  } /* while end */
  return(S_h^{op});
}

3.2 Meaningless Methods

A method op_1 *precedes* op_2 in a schedule H ($op_1 \rightarrow_H op_2$) iff (1) the methods op_1 and op_2 are issued by the same transaction T_i and op_1 is issued before op_2, (2) the method op_1 issued by a transaction T_i conflicts with the method op_2 issued by a transaction T_j and $T_i \rightarrow_H T_j$, or (3) $op_1 \rightarrow_H op_3 \rightarrow_H op_2$ for some method op_3. Let H_h be a *local schedule* of methods which are performed on an object o_h in a schedule H.

[Definition]. A method op_1 *locally precedes* another method op_2 in a local schedule H_h ($op_1 \rightarrow_{H_h} op_2$) iff $op_1 \rightarrow_H op_2$.

A partial write method op_1 locally precedes another full write method op_2 in a local schedule H_h ($op_1 \rightarrow_{H_h} op_2$) on the object o_h. Here, the partial write

method op_1 is not required to be performed on the object o_h if the full write method op_2 is surely performed on the object o_h just after the method op_1, i.e. the method op_2 can *absorb* the method op_1.

[Definition]. A full write method op_1 *absorbs* another partial or full write method op_2 in a local subschedule H_h on an object o_h if $op_2 \rightarrow_{H_h} op_1$, and there is no read method op' such that $op_2 \rightarrow_{H_h} op' \rightarrow_{H_h} op_1$, or op_1 absorbs op'' and op'' absorbs op_2 for some method op''.

[Definition]. A method op is *meaningless* iff the method op is absorbed by another method op' in the local subschedule H_h of an object o_h.

3.3 Omitting Meaningless Methods

Suppose three replicas o_1^1, o_1^2, and o_1^3 of an object o_1 are stored in three servers s_1, s_2, and s_3, respectively, i.e. $S_1 = \{s_1, s_2, s_3\}$. The version numbers v_1^1, v_1^2, and v_1^3 of replicas o_1^1, o_1^2, and o_1^3 are 2, 1, and 2, respectively, as shown in Fig. 1. The quorum numbers nQ_1^w and nQ_1^r for the object o_1 are two, respectively. 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 $T_i.Q_h^{op}$.

Suppose a pair of replicas o_1^1 and o_1^2 are locked by a transaction T_1 with lock mode $\mu(w)$ and a partial write methods $w_1^p(o_1)$ is issued to a w-quorum $T_1.Q_1^{w_1^p(o_1)} = \{o_1^1, o_1^2\}$ as shown in Fig. 1. The partial write method $w_{11}^p(o_1^1)$ is performed on the replica o_1^1 since the version number v_1^1 of the replica o_1^1 is the maximum in the w-quorum $T_1.Q_1^{w_1^p(o_1)}$, i.e. v_1^1 $(= 2) > v_1^2$ $(= 1)$. Then, the version number v_1^1 is incremented by one, i.e. $v_1^1 = 3$. The updated data and version number v_1^1 $(= 3)$ are sent to the replica o_1^2. In the traditional quorum-based locking protocol, data and version number of the replica o_1^2 are replaced with the newest values as soon as the replica o_1^2 receives the updated data and version number, i.e. the partial write method $w_{21}^p(o_1^2)$ is performed on the replica o_1^2. In the IEEQS algorithm, the version number of the replica o_1^2 is replaced with the newest value but data of the replica o_1^2 is not replaced with the newest values until the next method is performed on the replica o_1^2. This means that the partial write method $w_{21}^p(o_1^2)$ to replace data of a replica o_1^2 is delayed until the next method is performed on the replica o_1^2. Suppose a pair of replicas o_1^2 and o_1^3 are locked by a transaction T_2 with lock mode $\mu(w)$ and a full write methods $w_2^f(o_1)$ is issued to a w-quorum $T_2.Q_1^{w_2^p(o_1)} = \{o_1^2, o_1^3\}$ after the transaction T_1 commits. The full write method $w_{22}^f(o_1^2)$ issued by the transaction T_2 is performed on the replica o_1^2 since the version number v_1^2 is the maximum in the w-quorum $T_2.Q_1^{w_2^p(o_1)}$, i.e. v_1^2 $(= 3) > v_1^3$ $(= 2)$. Here, the partial write method $w_{21}^p(o_1^2)$ issued by the transaction T_1 is meaningless since the full write method $w_{22}^f(o_1^2)$ issued by the transaction T_2 absorbs the partial write method $w_{21}^p(o_1^2)$ on the replica o_1^2. Hence, the full write method $w_{22}^f(o_1^2)$ is performed on the replica o_1^2 without performing the partial write method $w_{21}^p(o_1^2)$ and the version number of the replica o_1^2 is incremented by one, i.e. $v_1^2 = 4$. That is, the meaningless method $w_{21}^p(o_1^2)$ is omitted on the replica o_1^2.

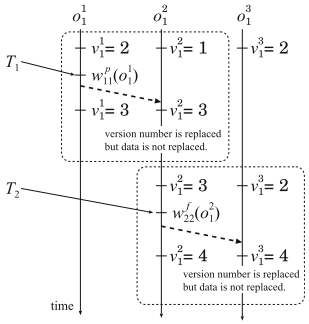

Fig. 1. Example of meaningless methods

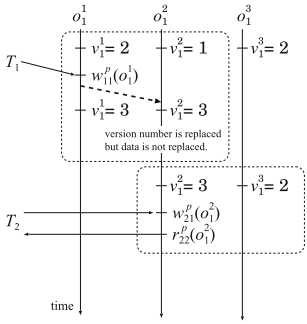

Fig. 2. Execution of read methods.

Suppose a pair of replicas o_1^2 and o_1^3 are locked by a transaction T_2 with lock mode $\mu(r)$ and a read method $r_1(o_1)$ is issued to a r-quorum $T_2.Q_1^{r_1(o_1)} = \{o_1^2, o_1^3\}$ after the transaction T_1 commits as shown in Fig. 2. The transaction T_2 reads data in the replica o_1^2 since the version number v_1^2 is the maximum in the r-quorum $T_2.Q_1^{r_1(o_1)}$, i.e. v_1^2 (= 3) > v_1^3 (= 2). Here, the partial write method $w_{21}^p(o_1^2)$ issued by a transaction T_1 has to be performed before the read method $r_{22}(o_1^2)$ is performed since the read method $r_{22}(o_1^2)$ has to read data written by the partial write method $w_{21}^p(o_1^2)$.

Let $o_h^q.DW$ be a write method $w_{ti}(o_h^q)$ issued by a transaction T_i to replace data of a replica o_h^q in a server s_t with the updated data d_h, which is waiting for the next method op to be performed on the replica o_h^q. Suppose a transaction T_i issues a method op to a quorum Q_h^{op} for manipulating an object o_h. The method $op(o_h)$ is performed on a replica o_h^q whose version number is the maximum in the quorum Q_h^{op}. In the IEEQS algorithm, the method op is performed on the replica o_h^q whose version number is the maximum in the quorum Q_h^{op} by the following **IEEQS_Perform** procedure:

IEEQS_Perform$(op(o_h^q))$ {
  if $op(o_h^q) = r$, {
    if $o_h^q.DW = \phi$, perform$(op(o_h^q))$;
    else { /* $o_h^q.DW \neq \phi$ */
      perform$(o_h^q.DW)$; perform$(op(o_h^q))$; $o_h^q.DW = \phi$;
    }
  }
  else { /* $op = w$ */
    $v_h^q = v_h^q + 1$;
    if $o_h^q.DW \neq \phi$ and $o_h^q.DW$ is not meaningless, {
      perform$(o_h^q.DW)$; perform$(op(o_h^q))$; $o_h^q.DW = \phi$;
    }
    else perform$(op(o_h^q))$; /* $o_h^q.DW = \phi$ or $o_h^q.DW$ method is omitted */
    v_h^q and updated data d_h are sent to every replica $o_h^{q'}$ in a quorum Q_h^{op};
  }
}

Each time a replica $o_h^{q'}$ in a write quorum Q_h^w receives the newest version number v_h^q and updated data d_h from another replica o_h^q as a result obtained by performing a write method $w_{ti}(o_h^q)$, the replica $o_h^{q'}$ manipulate the version number v_h^q and updated data d_h by the following **IEEQS_Replace** procedure:

IEEQS_Replace(v_h^q, d_h, $w_{ti}(o_h^q)$) {
  $v_h^{q'} = v_h^q$;
  if $o_h^{q'}.DW = \phi$, $o_h^{q'}.DW = w_{ti}(o_h^q)$;
  else { /* $o_h^{q'}.DW \neq \phi$ */
    if $w_{ti}(o_h^q)$ absorbs $o_h^{q'}.DW$, $o_h^{q'}.DW = w_{ti}(o_h^q)$;
    else {
      perform($o_h^{q'}.DW$); $o_h^{q'}.DW = w_{ti}(o_h^q)$;
    }
  }
}

4 Evaluation

4.1 Environment

We evaluate the IEEQS algorithm in terms of the average execution time of each transaction, the average number of aborted transactions, and the total electric energy of a server cluster S compared with the EEQS algorithm. A homogeneous server cluster S which is composed of ten homogeneous servers $s_1, \ldots, s_{10}$ ($n = 10$) is considered. In the server cluster S, every server s_t ($t = 1, \ldots, 10$) follows the same data access model and power consumption model as shown in Table 2. Parameters of each server s_t are given based on the experimentations [8]. There are fifty objects $o_1, \ldots, o_{50}$ in a system. The size of data in each object o_h is randomly selected between 50 and 100 [MB]. Each object o_h supports *read* (r), *full write* (w^f), and *partial write* (w^p) methods. The total number of replicas for every object is five, i.e. $nR(o_h) = 5$. Replicas of each object are randomly distributed on five servers in the server cluster S. The quorum numbers nQ_h^w and nQ_h^r on every object o_h are three, respectively, i.e. $nQ_h^w = nQ_h^r = 3$.

Table 2. 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]

The number m of transactions are issues to manipulate objects. Each transaction issues three methods randomly selected from one-hundred fifty methods on the fifty objects. The total amount of data of an object o_h is fully written by

each full write (w^f) method. On the other hand, a half size of data of an object o_h is written and read by each partial write (w^p) and read (r) methods. 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

Let ET_i be the execution time [sec] of a transaction T_i where the transaction T_i commits. Suppose a transaction T_i starts at time st_i and commits at time et_i. 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 ($0 \leq m \leq 1{,}000$). 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 3 shows the average execution time AET [sec] of the m transactions in the IEEQS and EEQS algorithms. In the IEEQS and EEQS algorithms, the average execution time AET increases as the total number m of transactions increases since more number of transactions are concurrently performed. For $0 < m \leq 1{,}000$, the average execution time AET can be more reduced in the IEEQS algorithm than the EEQS algorithm. In the IEEQS algorithm, each transaction can commit without waiting for performing meaningless methods. Hence, the average execution time of each transaction can be more reduced in the IEEQS algorithm than the EEQS algorithm.

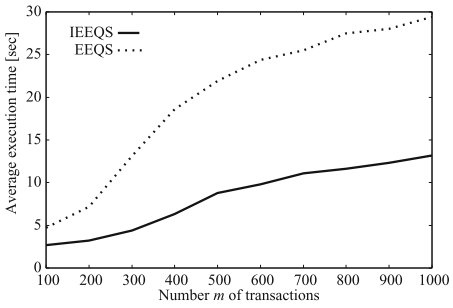

Fig. 3. Average execution time AET [sec] of each transaction.

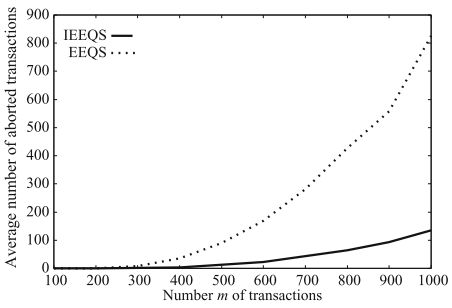

Fig. 4. Average number of aborts for each transaction

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 or 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 AT_i of aborted instances for each transaction T_i is measured ten times for each total number m of transactions ($0 \leq m \leq 1{,}000$). Let AT_i^{tm} be the number AT_i of aborted transaction instances obtained in tmth simulation. The average number AAT of aborted instances 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 4 shows the average number AAT of aborted transaction instances to perform the total number m of transactions in the IEEQS and EEQS algorithms. 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 IEEQS and EEQS algorithms as the total number m of transactions increases. For $0 < m \leq 1{,}000$, the average number AAT of aborted instances of each transaction can be more reduced in the IEEQS algorithm than the random algorithm. The average execution time of each transaction can be reduced in the IEEQS algorithm than the EEQS algorithm. As a result, the number of aborted transactions can be more reduced in the IEEQS algorithm than the EEQS algorithm since the number of transaction to be concurrently performed can be reduced.

4.4 Average Total Energy Consumption of a Server Cluster

Let TEC_{tm} be the total electric energy [J] to perform the number m of transactions ($0 \leq m \leq 1{,}000$) in the server cluster S obtained in the tm-th simulation. The total electric energy TEC_{tm} is measured ten times for each number m of transactions. Then, the average total electric energy $ATEC$ [J] of the server cluster S is calculated as $\sum_{tm=1}^{10} TEC_{tm}/10$ for each number m of transactions.

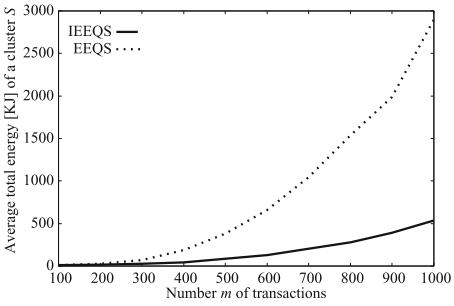

Fig. 5. Average total energy consumption (ATEC) [KJ].

Figure 5 shows the average total electric energy $ATEC$ of the server cluster S to perform the number m of transactions in the IEEQS and EEQS algorithms. For $0 \leq m \leq 1{,}000$, the average total electric energy $ATEC$ of the server cluster S can be more reduced in the IEEQS algorithm than the EEQS algorithm.

In the IEEQS algorithm, meaningless methods are omitted on each replica. In addition, the average execution time and the number of aborted instances of each transaction can be more reduced in the IEEQS algorithm than the EEQS algorithm. As a result, the average total electric energy $ATEC$ of the server cluster S can be more reduced in the IEEQS algorithm than the EEQS algorithm.

Following the evaluation, the total electric energy of a server cluster, the average execution time of each transaction, and the number of aborted transactions in the IEEQS algorithm can be reduced than the EEQS algorithm, respectively. Hence, the IEEQS algorithm is more useful than the EEQS algorithm.

5 Concluding Remarks

In this paper, we newly proposed the Improved EEQS (IEEQS) algorithm to reduce the total electric energy of a server cluster in the quorum-based locking protocol by omitting meaningless methods. We evaluated the IEEQS algorithm compared with the EEQS algorithm. The evaluation results show the total electric energy of a server cluster, the average execution time of each transaction, and the number of aborted transactions can be more reduced in the IEEQS algorithm than the EEQS algorithm. Following the evaluation, the IEEQS algorithm is more useful than the EEQS 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. In: Distributed Systems, 2nd edn. ACM Press (1993)
3. Sawamura, S., Barolli, A., Aikebaier, A., Enokido, T., Takizawa, M.: Design and evaluation of algorithms for obtaining objective trustworthiness on acquaintances in P2P overlay networks. Int. J. Grid Utility Comput. (IJGUC) **2**(3), 196–203 (2011)
4. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading (1987)
5. Gray, J.N.: Notes on database operating systems. In: Operating Systems, vol. 60, pp. 393–481 (1978)
6. Tanaka, K., Hasegawa, K., Takizawa, M.: Quorum-based replication in object-based systems. J. Inf. Sci. Eng. **16**(3), 317–331 (2000)
7. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. J. ACM **32**(4), 814–860 (1985)
8. 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)
9. Enokido, T., Duolikun, D., Takizawa, M.: Energy-efficient Quorum selection algorithm for distributed object-based systems. In: Proceedings of the 11th International Conference on Conference on Complex, Intelligent, and Software Intensive Systems, (CISIS-2017), pp. 32–42 (2017)