# Multi-Version Concurrency Control to Reduce the Electric Energy Consumption of Servers

Tomoya Enokido[1(✉)], Dilawaer Duolikun[2], and Makoto Takizawa[3]

[1] Faculty of Business Administration, Rissho University, 4-2-16, Osaki,
Shinagawa-ku, Tokyo 141-8602, Japan
`eno@ris.ac.jp`
[2] Department of Advanced Sciences, Faculty of Science and Engineering,
Hosei University, 3-7-2, Kajino-cho, Koganei-shi, Tokyo 184-8584, Japan
[3] Research Center for Computing and Multimedia Studies, Hosei University,
3-7-2, Kajino-cho, Koganei-shi, Tokyo 184-8584, Japan
`makoto.takizawa@computer.org`

**Abstract.** The MVCC (Multi-Version Concurrency Control) is so far proposed to increase the concurrency of multiple conflicting transactions and the scalability of a distributed system. However, the larger number of transactions are concurrently performed, the larger amount of electric energy is consumed by servers in a system. In our previous studies, the EEMVTO (Energy-Efficient Multi-Version Timestamp Ordering) algorithm is proposed to not only reduce the total electric energy consumption of servers but also increase the throughput of a system by not performing meaningless write methods on each object. In this paper, the IEEMVTO (Improved EEMVTO) algorithm is newly proposed to furthermore reduce the total electric energy consumption of servers by not performing meaningless read methods in addition to meaningless write methods. The evaluation results show the total electric energy consumption of servers can be more reduced in the IEEMVTO algorithm than the EEMVTO algorithm.

**Keywords:** Multi-version concurrency control · Energy-Efficient Multi-Version Timestamp Ordering (EEMVTO) · Improved EEMVTO (IEEMVTO) algorithm · Object-based system · Transaction

## 1 Introduction

In current information systems, a huge number of IoT (Internet of Things) devices [1,2] are deployed in a system and each IoT device collects various types of data like temperature and humidity which are required by an application. A huge volume of data is gathered from these IoT devices in order to realize applications and the data gathered from IoT devices is encapsulated along with methods to manipulate data as an object [3] like database systems. An application is composed of multiple objects distributed to multiple physical servers in an

L. Barolli (Ed.): BWCCA 2022, LNNS 570, pp. 180–191, 2023.
https://doi.org/10.1007/978-3-031-20029-8_18

object-based system [3,4,6]. A transaction [7,8] is an atomic sequence of methods to manipulate objects. In order to utilize an application service, a transaction is created on a client and issues methods supported by each target object. Multiple conflicting transactions have to be serialized [4,6–11] to keep every object mutually consistent. The *MVCC* (*Multi-Version Concurrency Control*) [9,10] is proposed to not only serialize conflicting transactions but also increase the concurrency of transactions and scalability of a system. In the MVCC, each read method is ensured to read the latest committed version of each object. In addition, each read method is not blocked by the other methods. As a result, the MVCC can increase the concurrency of transactions and the throughput of a system. In order to realize the MVCC, the *MVTO* (*Multi-Version Timestamp Ordering*) algorithm [9,10] is proposed. However, the more number of transactions are issued in a system, the larger amount of electric energy is consumed by servers since every method issued to each target object is surely performed on each object. Hence, it is critical to discuss how to not only increase the concurrency of transactions and the throughput of a system but also reduce the total electric energy consumption of servers as discussed in Green computing systems [5,6,12–15].

In our previous studies, *meaningless write methods* [16] which are not required to be performed on each object are defined based on the precedent relation among transactions and the semantics of methods. Then, the *EEMVTO* (*Energy-Efficient Multi-Version Timestamp Ordering*) algorithm [16] is proposed to not only reduce the total electric energy consumption of servers but also increase the throughput of a system by not performing meaningless write methods on each object. In this paper, we newly introduce *meaningless read methods* which are not required to be performed on each object. Then, the *Improved EEMVTO* (*IEEMVTO*) algorithm is newly proposed to furthermore reduce the total electric energy consumption of servers and the execution time of each transaction by not performing both meaningless read and write methods. The IEEMVTO algorithm is evaluated in terms of the total electric energy consumption of servers and the average execution time of each transaction compared with the EEMVTO algorithm. Evaluation results show the total electric energy consumption of servers and the average execution time of each transaction in the IEEMVTO algorithm can be more reduced than the EEMVTO algorithm.

In Sect. 2, we present the system model and the MVTO algorithm. In Sect. 3, we propose the IEEMVTO algorithm. In Sect. 4, we evaluate the IEEMVTO algorithm compared with the EEMVTO algorithm.

## 2 System Model

### 2.1 Object-Based Systems

A system is composed of a cluster $S$ of multiple servers $s_1$, ..., $s_n$ ($n \geq 1$) and clients interconnected in reliable networks. Let $O$ be a set of objects $o_1$, ..., $o_m$ ($m \geq 1$) in the system. An object [3] is an unit of computation resource like a database. Each object $o_h$ is an encapsulation of data $d_h$ and methods to manipulate data $d_h$ in the object $o_h$. Each object $o_h$ is allocated to a server $s_t$ in the cluster $S$. Methods are classified into *read* ($r$) and *write* ($w$) methods in this paper. Write methods are furthermore classified into *full* write ($fw$) and *partial* write ($pw$) methods, i.e. $w \in \{fw, pw\}$. A full write method fully writes a whole data $d_h$ in an object $o_h$. A partial write method writes only a part of data $d_h$ in an object $o_h$.

## 2.2 Multi-Version Timestamp Ordering (MVTO) Algorithm

A *transaction* is an atomic sequence of methods [8]. A transaction $T^i$ issues read ($r$) and write ($w$) methods to manipulate objects in the set $O$. Let $\mathbf{T}$ be a set $\{T^1, ..., T^k\}$ ($k \geq 1$) of transactions issued in a system. Multiple conflicting transactions are required to be *serializable* [7,8] to keep all the objects mutually consistent. The *MVCC* (Multi-Version Concurrency Control) [9] is proposed to increase the concurrency of transactions and the throughput of a system. Let $H$ be a schedule [9] of the transaction set $\mathbf{T}$. Each object $o_h$ has a totally ordered set $D_h$ of multiple versions $d_h^1$, ..., $d_h^l$ ($l \geq 1$) of data $d_h$. A totally ordered relation $\ll_h$ ($\subseteq D_h^2$) shows an order of versions of data $d_h$ of an object $o_h$ written in a schedule $H$. $d_h^i \ll_h d_h^j$ means $d_h^i$ is written before $d_h^j$ in an object $o_h$. Let $\ll$ be an union of version orders $\ll_h$ for every data $d_h$ in a schedule $H$, i.e. $\ll_h = \bigcup_{o_h \in O} \ll_h$. A transaction $T^j$ *reads data from* another transaction $T^i$ ($T^i \rightarrow_H T^j$) in a schedule $H$ iff the transaction $T^j$ reads a version $d_h^i$ of an object $o_h$ written by the transaction $T^i$. $T^i \parallel_H T^j$ iff neither $T^i \rightarrow_H T^j$ nor $T^j \rightarrow_H T^i$. A schedule $H$ is $\langle \mathbf{T}, \rightarrow_H \rangle$ ($\subseteq \mathbf{T}^2$).

**[One-Copy Serial].** A schedule $H = \langle \mathbf{T}, \rightarrow_H \rangle$ is *one-copy serial* [9] iff (if and only if) for every pair of different transactions $T^i$ and $T^j$ in $\mathbf{T}$, either $T^i \rightarrow_H T^j$, $T^j \rightarrow_H T^i$, or $T^i \parallel_H T^j$.

In an one-copy serial schedule $OH = \langle \mathbf{T}, \rightarrow_{OH} \rangle$ ($\subseteq \mathbf{T}^2$), if $T^i \rightarrow_H T^j$, $T^i \rightarrow_{OH} T^j$, and the relation, $\rightarrow_{OH}$ is acyclic.

Let $r_t^i(d_h^j)$ be a read method issued by a transaction $T^i$ to read a version $d_h^j$, which is written by a transaction $T^j$, of an object $o_h$ on a server $s_t$. Let $w_t^i(d_h^i)$ be a write method issued by a transaction $T^i$ to write a version $d_h^i$ in an object $o_h$ on a server $s_t$.

A *multi-version schedule* $MVS$ is $\langle \mathbf{T}, \rightarrow_{MVS} \rangle$ ($\subseteq \mathbf{T}^2$) where for every pair of transactions $T^i$ and $T^j$ in $\mathbf{T}$, the following conditions hold:

(1) If $T^i \rightarrow_{OH} T^j$, $T^i \rightarrow_{MVS} T^j$.
(2) If $T^i$ writes a version $d_h^i$, $T^j$ reads a version $d_h^k$, and $T^i \rightarrow_{MVS} T^j$, $d_h^i \ll_h d_h^k$ or $d_h^i = d_h^k$.

**[One-Copy Serializability].** A multi-version schedule $MVS = \langle \mathbf{T}, \rightarrow_{MVS} \rangle$ is *one-copy serializable* [9] iff for every pair of transactions $T^i$ and $T^j$ in $\mathbf{T}$, either $T^i \rightarrow_{MVS} T^j$, $T^j \rightarrow_{MVS} T^i$, or $T^i \parallel_{MVS} T^j$.

The *MVTO* (*Multi-Version Timestamp Ordering*) algorithm [9,10] is proposed to make transactions one-copy serialize. Each transaction $T^i$ is given an unique timestamp $TS(T^i)$ which shows time when the transaction $T^i$ is created. Suppose a transaction $T^i$ issues a method *op* to manipulate an object $o_h$ in a server $s_t$. In the MVTO algorithm, a method *op* issued by a transaction $T^i$ is performed by the following procedure [9,10]:

1. If a method *op* is a read method $r_t^i(d_h^k)$, the read method *op* reads a version $d_h^k$ written by a transaction $T^k$ whose timestamp $TS(T^k)$ is the maximum in $TS(T^k) < TS(T^i)$.
2. If a method *op* is a write method $w_t^i(d_h^i)$, the write method *op* is rejected if a read method $r_t^j(d_h^k)$ is performed on the object $o_h$ such that $TS(T^k) < TS(T^i) < TS(T^j)$. Otherwise, the write method $w_t^i(d_h^i)$ is performed.

By using the MVTO algorithm, each read method reads the latest committed version of an object $o_h$. In addition, each read method is not blocked by the other methods.

### 2.3 Data Access Model

Methods which are being performed and already terminate are *current* and *previous* at time $\tau$, 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 $\tau$, respectively. A notation $P_t(\tau)$ shows a set of current read and write methods on a server $s_t$ at time $\tau$, i.e. $P_t(\tau) = RP_t(\tau) \cup WP_t(\tau)$. Each read method $r_t^i(d_h^j)$ in a set $RP_t(\tau)$ reads a version $d_h^j$ in an object $o_h$ at rate $RR_t^i(\tau)$ [Byte/sec (B/sec)] at time $\tau$. Each write method $w_t^i(d_h^i)$ in a set $WP_t(\tau)$ writes a version $d_h^i$ in an object $o_h$ at rate $WR_t^i(\tau)$ [B/sec] at time $\tau$. Let $maxRR_t$ and $maxWR_t$ be the maximum read and write rates [B/sec] of read and write methods on a server $s_t$, respectively. The read rate $RR_t^i(\tau)$ ($\leq maxRR_t$) and write rate $WR_t^i(\tau)$ ($\leq maxWR_t$) are $dr_t(\tau) \cdot maxRR_t$ and $dw_t(\tau) \cdot maxWR_t$, respectively. Here, $dr_t(\tau)$ and $dw_t(\tau)$ are degradation ratios. $1 \ / \ (|RP_t(\tau)| + rw_t \cdot |WP_t(\tau)|)$ and $1 \ / \ (wr_t \cdot |RP_t(\tau)| + |WP_t(\tau)|)$, respectively, where $0 \leq rw_t \leq 1$ and $0 \leq wr_t \leq 1$. $0 \leq dr_t(\tau) \leq 1$ and $0 \leq dw_t(\tau) \leq 1$.

The *read laxity* $rl_t^i(\tau)$ [B] and *write laxity* $wl_t^i(\tau)$ [B] of methods $r_t^i(d_h^j)$ and $w_t^i(d_h^i)$ show the amount of data to be read and written in an object $o_h$ by the methods $r_t^i(d_h^j)$ and $w_t^i(d_h^i)$ at time $\tau$, respectively. Suppose that methods $r_t^i(d_h^j)$ and $w_t^i(d_h^i)$ start on a server $s_t$ at time $st_t^i$. At time $st_t^i$, the read laxity $rl_t^i(\tau) = rb_h^j$ [B] where $rb_h^j$ is the size of the version $d_h^j$ in an object $o_h$. The write laxity $wl_t^i(\tau) = wb_h^i$ [B] where $wb_h^i$ is the size of the version to be written in an object $o_h$. The read laxity $rl_t^i(\tau)$ and write laxity $wl_t^i(\tau)$ at time $\tau$ are $rb_h^j$ - $\Sigma_{\tau=st_t^i}^{\tau} RR_t^i(\tau)$ and $wb_h^i$ - $\Sigma_{\tau=st_t^i}^{\tau} WR_t^i(\tau)$, respectively.

### 2.4 Power Consumption Model of a Server

In our previous studies, the *PCS* model (*Power Consumption model for a Storage server*) [17] to perform storage and computation processes are proposed. Let

$E_t(\tau)$ be the electric power [W] of a server $s_t$ at time $\tau$. $maxE_t$ and $minE_t$ show the maximum and minimum electric power [W] of the server $s_t$, respectively. In this paper, we assume only read and write methods are performed on a server $s_t$. According to the PCS model [17], the electric power $E_t(\tau)$ [W] of a server $s_t$ to perform multiple read and write methods at time $\tau$ is given as follows:

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

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 $TEE_t(\tau_1, \tau_2)$ [J] of a server $s_t$ from time $\tau_1$ to $\tau_2$ is $\Sigma_{\tau=\tau_1}^{\tau_2} E_t(\tau)$. The processing electric power $PEP_t(\tau)$ [W] of a server $s_t$ at time $\tau$ is $E_t(\tau)$ - $minE_t$. The total processing electric energy $TPEE_t(\tau_1, \tau_2)$ of a server $s_t$ from time $\tau_1$ to $\tau_2$ is given as $TPEE_t(\tau_1, \tau_2) = \Sigma_{\tau=\tau_1}^{\tau_2} PEP_t(\tau)$.

## 3 Improved EEMVTO (IEEMVTO) Algorithm

### 3.1 Meaningless Methods

Let $MH_h$ be a *local schedule* of methods which are performed on an object $o_h$ in a multi-version schedule $MH$. A method $op^1$ of a transaction $T^1$ *locally precedes* another method $op^2$ of a transaction $T^2$ in a local schedule $MH_h$ ($op^1 \rightarrow_{MH_h} op^2$) iff $T^1 \rightarrow_{MH} T^2$ and $op^1$ is performed before $op^2$ on an object $o_h$. Suppose a partial write method $pw^i(d_h^i)$ issued by a transaction $T^i$ locally precedes another full write method $fw^j(d_h^j)$ issued by a transaction $T^j$ in a local schedule $MH_h$ ($pw^i(d_h^i) \rightarrow_{MH_h} fw^j(d_h^j)$) on an object $o_h$. Here, the partial write method $pw^i(d_h^i)$ is not required to be performed on the object $o_h$ if the full write method $fw^j(d_h^j)$ is surely performed on the object $o_h$ just after the partial write method $pw^i(d_h^i)$, i.e. the full write method $fw^j(d_h^j)$ can *absorb* the partial write method $pw^i(d_h^i)$.

**[Absorption of Write Methods].** A full write method $op^1$ *absorbs* another partial or full write method $op^2$ in a local subschedule $MH_h$ on an object $o_h$ iff one of the following conditions is hold:

1. $op^2 \rightarrow_{MH_h} op^1$ and there is no read method $op'$ such that $op^2 \rightarrow_{MH_h} op' \rightarrow_{MH_h} op^1$.
2. $op^1$ absorbs $op^3$ and $op^3$ absorbs $op^2$ for some method $op^3$.

[**Absorption of Read Methods**]. A read method $op^1$ absorbs another read method $op^2$ in a local subschedule $H_h$ of an object $o_h$ iff one of the following conditions is hold:

1. $op^1 \rightarrow_{H_h} op^2$ and there is no write method $op'$ such that $op^1 \rightarrow_{H_h} op' \rightarrow_{H_h} op^2$.
2. $op^1$ absorbs $op^3$ and $op^3$ absorbs $op^2$ for some method $op^3$.

[**Meaningless Methods**]. A method $op$ is *meaningless* iff the method $op$ is absorbed by another method $op'$ in the local subschedule $MH_h$ on an object $o_h$.

### 3.2 IEEMVTO Algorithm

In this paper, the *IEEMVTO* (*Improved EEMVTO*) algorithm is newly proposed to furthermore reduce not only the total electric energy consumption of a cluster of servers but also the average execution time of each transaction by not performing meaningless read and write methods on each object. In this paper, we assume transactions are serialized based on the MVTO algorithm [9,10].

Suppose a read method $r^i_t(d^k_h)$ issued by a transaction $T^i$ is performed on the object $o_h$ as shown in Fig. 1. A transaction $T^j$ issues a read method $r^j_t(o^k_h)$ to the object $o_h$ while the read method $r^i_t(d^k_h)$ is being performed on the object $o_h$. In the MVTO algorithm, the read method $r^j_t(o^k_h)$ is performed on the object $o_h$ as soon as the object $o_h$ receives the read method $r^j_t(o^k_h)$. In the IEEMVTO algorithm, the read method $r^j_t(o^k_h)$ is meaningless since the read method $r^i_t(o^k_h)$ issued by the transaction $T^i$ is being performed on the object $o_h$ and the read method $r^i_t(o^k_h)$ absorbs the read method $r^j_t(o^k_h)$. Hence, the read method $r^j_t(o^k_h)$ is not performed on the object $o_h$ and a result obtained by performing the read method $r^i_t(o^k_h)$ is sent to a pair of transactions $T^i$ and $T^j$.

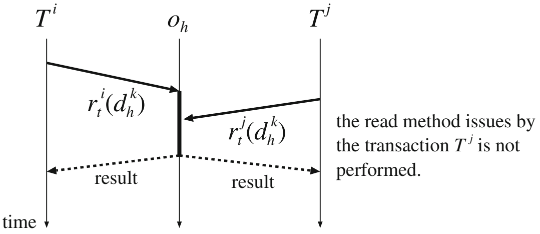

**Fig. 1.** A meaningless read method.

Suppose a transaction $T^i$ issues a partial write method $pw^i_t(d^i_h)$ to an object $o_h$ allocated to a server $s_t$ as shown in Fig. 2. In the MVTO algorithm, the partial write method $pw^i_t(d^i_h)$ is performed on the object $o_h$ as soon as the object $o_h$ receives the partial write method $pw^i_t(d^i_h)$. In the EEMVTO algorithm, the object $o_h$ sends a termination notification of the partial write method $pw_t^i(d_h^i)$ to the transaction $T^i$ as soon as the object $o_h$ receives the partial write method $pw_t^i(d_h^i)$. However, the partial write method $pw_t^i(d_h^i)$ is not performed until the object $o_h$ receives a method *op* which is performed just after the partial write method $pw_t^i(d_h^i)$ on the object $o_h$, i.e. the partial write method $pw_t^i(d_h^i)$ is delayed. Suppose a transaction $T^j$ issues a full write methods $fw_t^j(d_h^j)$ to the object $o_h$ after the transaction $T^i$ commits. Here, the partial write method $pw_t^i(d_h^i)$ issued by the transaction $T^i$ is meaningless since the full write method $fw_t^j(d_h^j)$ issued by the transaction $T^j$ absorbs the partial write method $pw_t^i(d_h^i)$ on the object $o_h$. Hence, the full write method $fw_t^j(d_h^j)$ can be performed on the object $o_h$ without performing the partial write method $pw_t^i(d_h^i)$. This means that the meaningless write method $pw_t^i(d_h^i)$ is not performed on the object $o_h$.

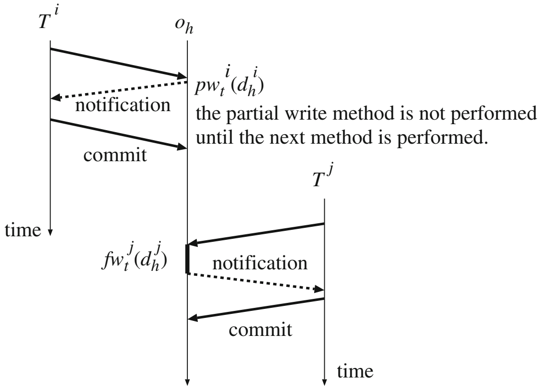

**Fig. 2.** Omission of a meaningless write method.

Suppose a transaction $T^j$ issues a read method $r_t^j(d_h^i)$ after another transaction $T^i$ commits. Here, the partial write method $pw_t^i(d_h^i)$ issued by the transaction $T^i$ has to be performed before the read method $r_t^j(d_h^i)$ is performed since the read method $r_t^j(d_h^i)$ has to read a version $d_h^i$ written by the partial write method $pw_t^i(d_h^i)$.

Let $o_h.Cr$ be a read method $r_t^i(d_h^k)$ issued by a transaction $T^i$, which is being performed on a object $o_h$. A notation $o_h.Dw$ is a write method $w_t^i(d_h^i)$ issued by a transaction $T^i$ to write data $d_h^i$ of an object $o_h$ in a server $s_t$, which is waiting for a method *op* to be performed on the object $o_h$ after $w_t^i(d_h^i)$. Suppose a transaction $T^i$ issues a method *op* to an object $o_h$. In the IEEMVTO algorithm, the method *op* is performed on the object $o_h$ by the following **IEEMVTO** procedure:

```
IEEMVTO($op$) {
    if $op = r$, { /* $op$ is a read method. */
        if $o_h.Dw = \phi$, {
            if $o_h.Cr = \phi$, {
```

```
            $o_h.Cr = op(d_h^k)$;
            **perform**($op(d_h^k)$); /* $d_h^k$ is the latest committed data. */
            $o_h.Cr = \phi$;
         }
         **else** a result of $o_h.Cr$ is sent to a transaction $T^i$;
      }
      **else** {
         **perform**($o_h.Dw$);
         $o_h.Dw = \phi$;
         $o_h.Cr = op(d_h^k)$;
         **perform**($op(d_h^k)$); /* $d_h^k$ is the latest committed data. */
         $o_h.Cr = \phi$;
      }
   }
   **else** { /* $op$ is a write method. */
      **if** $o_h.Dw = \phi$, $o_h.Dw = op(d_h^i)$;
      **else** { /* $o_h.Dw \neq \phi$ */
         **if** $op(d_h^i)$ absorbs $o_h.Dw$, $o_h.Dw = op(d_h^i)$; /* $o_h.Dw$ is not performed. */
         **else** {
            **perform**($o_h.Dw$);
            $o_h.Dw = op(d_h^i)$;
         }
      }
   }
}
```

In the IEEMVTO algorithm, the total electric energy consumption of a cluster $S$ of servers can be furthermore reduced than the EEMVTO algorithm since the number of read and write methods performed on each object can be more reduced. In addition, the computation resources which are used to perform meaningless read and write methods can be used to perform the other methods in each server $s_t$. As a result, the execution time of each transaction can be more reduced in the IEEMVTO algorithm than the EEMVTO algorithm. This means that the throughput of a system can increase in the IEEMVTO algorithm than the EEMVTO algorithm.

## 4 Evaluation

### 4.1 Environment

We evaluate the IEEMVTO algorithm in terms of the total processing electric energy of a cluster $S$ of homogeneous servers and the average execution time of each transaction compared with the EEMVTO algorithm [16]. The cluster $S$ of servers is composed of ten homogeneous servers $s_1$, ..., $s_{10}$ ($n = 10$), where every server $s_t$ ($t = 1$, ..., 10) follows the same data access model and power consumption model. Parameters of each server $s_t$ are shown in Table 1, which are obtained based on the experimentations [17]. There are thirty objects $o_1$, ..., $o_{30}$ in a system. The size of data in each object $o_h$ is randomly selected between 50 and 100 [MByte]. Each object $o_h$ supports *read* ($r$), *full write* ($fw$), and *partial write* ($pw$) methods. Each object is randomly allocated to a server $s_t$ in the cluster $S$.

**Table 1.** Homogeneous cluster $S$ of servers ($t$ = 1, ..., 10)

| 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 $nt$ ($0 \leq nt \leq 500$) of transactions are issued 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 ($fw$) method. On the other hand, a half size of data of an object $o_h$ is written and read by each partial write ($pw$) and read ($r$) methods, 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 Total Processing Electric Energy Consumption

Figure 3 shows the total processing electric energy consumption [KJ] of the cluster $S$ of servers to perform the number $nt$ of transactions in the IEEMVTO and EEMVTO algorithms. For $0 \leq nt \leq 500$, the total processing electric energy consumption of the cluster $S$ of servers can be more reduced in the IEEMVTO algorithm than the EEMVTO algorithm. In the IEEMVTO algorithm, meaningless read and write methods are not performed on each object. As a result, the total processing electric energy consumption of the cluster $S$ of servers can be more reduced in the IEEMVTO algorithm than the EEMVTO algorithm.

### 4.3 Average Execution Time of Each Transaction

Figure 4 shows the average execution time [sec] of the $nt$ transactions in the IEEMVTO and EEMVTO algorithms. In the IEEMVTO and EEMVTO algorithms, the average execution time increases as the total number $nt$ of transactions increases since more number of transactions are concurrently performed. For $0 < nt \leq 500$, the average execution time of each transaction can be more reduced in the IEEMVTO algorithm than the EEMVTO algorithm. In the IEEMVTO algorithm, each transaction can commit without waiting for performing meaningless methods. Hence, the average execution time of each transaction is shorter in the IEEMVTO algorithm than the EEMVTO algorithm.

Following the evaluation, the total processing electric energy consumption of a homogeneous cluster $S$ of servers and the average execution time of each

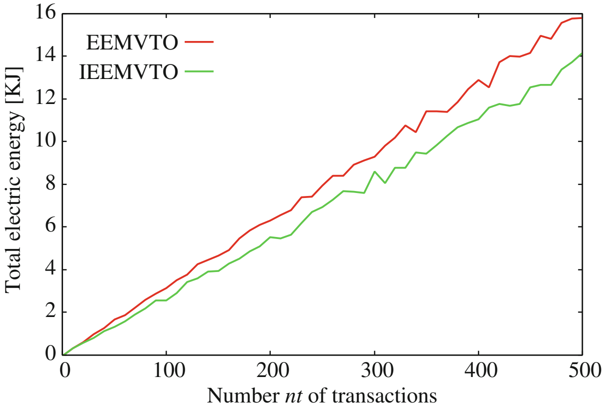

**Fig. 3.** Total processing electric energy consumption [KJ] of a cluster $S$ of servers.

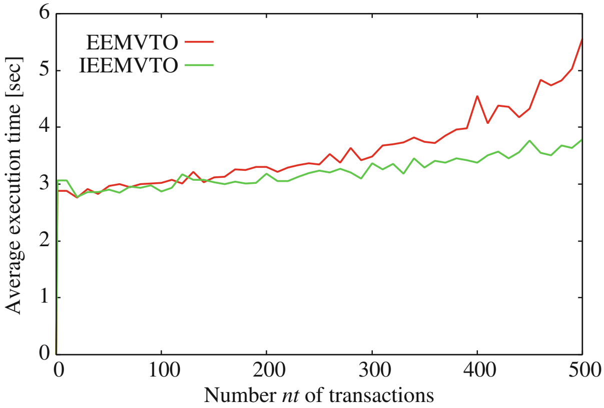

**Fig. 4.** Average execution time [sec] of each transaction.

transaction can be more reduced in the IEEMVTO algorithm than the EEMVTO algorithm. Hence, the IEEMVTO algorithm is more useful than the EEMVTO algorithm.

## 5 Concluding Remarks

In this paper, we newly proposed the IEEMVTO algorithm to reduce not only the total processing electric energy consumption of a cluster of servers but also the average execution time of each transaction by not performing meaningless read and write methods. We evaluated the IEEMVTO algorithm compared with the EEMVTO algorithm. The evaluation results showed the total processing electric energy consumption of a cluster of servers and the average execution time of each transaction can be more reduced in the IEEMVTO algorithm than the EEMVTO algorithm. Following the evaluation, the IEEMVTO algorithm is more useful than the EEMVTO algorithm.

## References

1. Nakamura, S., Enokido, T., Takizawa, M.: Implementation and evaluation of the information flow control for the Internet of Things. Concurr. Comput. Practice Exp. **33**(19), e6311 (2021)
2. Enokido, T., Takizawa, M.: The redundant energy consumption laxity based algorithm to perform computation processes for IoT services. Internet Things **9** (2020). https://doi.org/10.1016/j.iot.2020.100165
3. Object Management Group Inc.: Common object request broker architecture (CORBA) specification, version 3.3, Part 1 - interfaces (2012). https://www.omg.org/spec/CORBA/3.3/Interfaces/PDF
4. Tanaka, K., Hasegawa, K., Takizawa, M.: Quorum-based replication in object-based systems. J. Inf. Sci. Eng. **16**(3), 317–331 (2000)
5. Enokido, T., Duolikun, D., Takizawa, M.: An energy-efficient quorum-based locking protocol by omitting meaningless methods on object replicas. J. High Speed Netw. **28**(3), 181–203 (2022)
6. Enokido, T., Duolikun, D., Takizawa, M.: Energy-efficient concurrency control by omitting meaningless write methods in object-based systems. In: Proceedings of the 36th International Conference on Advanced Information Networking and Applications (AINA-2022), pp. 129–139 (2022)
7. 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). https://doi.org/10.1007/3-540-08755-9_9
8. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, Boston (1987)
9. Bernstein, P.A., Goodman, N.: Multiversion concurrency control - theory and algorithms. ACM Trans. Database Syst. **8**(4), 465–483 (1983)
10. Reed, D.: Naming and synchronization in a decentralized computer system. Technical report, MIT/LCS/TR-205, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1978). https://hdl.handle.net/1721.1/16279
11. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. J. ACM **32**(4), 814–860 (1985)
12. Natural Resources Defense Council (NRDS): Data center efficiency assessment - scaling up energy efficiency across the data center industry. Evaluating key drivers and barriers (2014). https://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf
13. Enokido, T., Duolikun, D., Takizawa, M.: Energy consumption laxity-based quorum selection for distributed object-based systems. Evol. Intel. **13**(1), 71–82 (2018). https://doi.org/10.1007/s12065-018-0157-1
14. Enokido, T., Duolikun, D., Takizawa, M.: The improved redundant active time-based (IRATB) algorithm for process replication. In: Barolli, L., Woungang, I., Enokido, T. (eds.) AINA 2021. LNNS, vol. 225, pp. 172–180. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75100-5_16

15. Enokido, T., Duolikun, D., Takizawa, M.: The redundant active time-based algorithm with forcing meaningless replica to terminate. In: Barolli, L., Yim, K., Enokido, T. (eds.) CISIS 2021. LNNS, vol. 278, pp. 206–213. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79725-6_20
16. Enokido, T., Duolikun, D., Takizawa, M.: Energy-efficient multi-version concurrency control (EEMVCC) for object-based systems. accepted for publication. In: Barolli, L., Miwa, H., Enokido, T. (eds.) NBiS 2022. LNNS, vol. 526. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14314-4_2
17. 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)