Energy-Efficient Purpose Ordering Scheduler

Tomoya Enokido[1(✉)] 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
makoto.takizawa@computer.org

Abstract. Distributed applications are composed of multiple objects. An object is an unit of computation resource. Conflicting transactions have to be serialized to keep objects mutually consistent. In this paper, the energy-efficient purpose ordering (EEPO) scheduler is proposed to not only serialize multiple conflicting transactions in the significant order of purposes assigned to the transactions but also reduce the total electric energy consumption of servers by omitting meaningless methods.

Keywords: Energy-aware information systems · Transactions · Purposes · The EEPO scheduler · The EERO scheduler · The RO scheduler

1 Introduction

A subject doing a job function plays a *role* [1,2] in an enterprise. In the role-based access control (RBAC) model [1–3], a role is a set of access rights. An access right is given in a pair $\langle o, op \rangle$ of an abject o [4] and a method *op*. A subject granted a role including an access right $\langle o, op \rangle$ can manipulate the object o through the method *op* by issuing a transaction. A transaction [5,6] is an atomic sequence of methods issued by a subject to manipulate objects. Conflicting methods [6] issued by multiple transactions have to be serialized on an object to keep the object mutually consistent. There are various ways to serialize multiple conflicting methods like timestamp ordering (TO) [5] and FIFO [5,6].

In our previous studies, the role ordering (RO) scheduler [3] is proposed to serialize multiple conflicting transactions in the significant order of roles granted to subjects and *authorization relation* [1,2,7] of roles. The RO scheduler does not consider to reduce the total electric energy consumption of servers to perform methods on objects. The energy-efficient role ordering (EERO) scheduler [8] is proposed to not only serialize multiple conflicting transactions in the significant order of roles granted to subjects but also reduce the total electric energy consumption of servers by omitting meaningless methods. In the RO and EERO schedulers, a subject granted more significant roles is more significant than other

L. Barolli et al. (Eds.): BWCCA 2019, LNNS 97, pp. 137–149, 2020.
https://doi.org/10.1007/978-3-030-33506-9_13

subjects granted less significant roles. Then, a method issued by a more significant subject is performed prior to other methods issued by less significant subjects. However, this assumption is not true in some types of applications. For example, suppose a *president* would like to access to a bank account just for checking but a *manager* would like to access to the bank account for transferring money to make payment. The purpose *payment* is more significant than the purpose *check* in a purpose point of view. Hence, a method issued by the *manager* should be performed prior to a method issued by the *president*.

In this paper, a subject assigns a transaction with a purpose which is a subset of roles granted to the subject. We first define the *purpose-oriented dominant* relation among subjects. Then, the *energy-efficient purpose ordering* (*EEPO*) scheduler is proposed to not only serialize multiple conflicting transactions in the significant order of purposes but also reduce the total electric energy consumption of servers by omitting meaningless methods. We evaluate the EEPO scheduler in terms of the total electric energy consumption of servers and the execution time of each transaction compared with the RO and EERO schedulers.

In Sect. 2, we discuss the significancy of transactions, meaningless methods, and power consumption model of a server. In Sect. 3, we propose the EEPO scheduler. In Sect. 4, we evaluate the EEPO scheduler compared with the RO and EERO schedulers.

2 System Model

2.1 Object-Based Systems with RBAC Model

A server cluster S is composed of multiple servers s_1, ..., s_n $(n \geq 1)$ and multiple clients $cl_1, ..., cl_l$ $(l \geq 1)$ interconnected in reliable networks. Let O be a set of objects $o_1, ..., o_m$ $(m \geq 1)$ [4]. 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. Objects are distributed on multiple servers. A pair of methods op_1 and op_2 *conflict* if and only if (iff) the result obtained by performing the methods depends on the computation order. Otherwise, op_1 and op_2 are *compatible*. A *transaction* is an atomic sequence of methods [5]. A transaction T_i is initiated in a client cl_s and issues methods to servers to manipulate objects. In this paper, we assume each transaction T_i serially issues methods. Each transaction T_i initiated in a client cl_s is given an identifier $tid(T_i) = \langle V(T_i), id(cl_s)\rangle$ where $V(T_i)$ is a logical time of the client cl_s when T_i is initiated and $id(cl_s)$ is an identifier of the client cl_s. For every pair of transaction identifiers $tid(T_i)\,(= \langle V(T_i), id(cl_1)\rangle)$ and $tid(T_j)\,(= \langle V(T_j), id(cl_2)\rangle)$, $tid(T_i) < tid(T_j)$ iff 1) $V(T_i) < V(T_j)$ or 2) $id(cl_1) < id(cl_2)$ and $V(T_i) \parallel V(T_j)$. A role R is a collection of *access rights* in the *role-based access control* (*RBAC*) model [1,2]. An access right is specified in a pair $\langle o, op\rangle$ of an object o and a method op. If a subject Sub is granted a role R including $\langle o, op\rangle$, the subject Sub is allowed to invoke a method op on an object o. Let $Srole$ be a family $\{R_1, ..., R_q\}$ of roles granted to a subject Sub. Let Sub_i denote a subject which initiates a transaction T_i.

2.2 Significancy of Methods

Class methods are ones for *creating* and *dropping* an object. *Object* methods are ones for manipulating data in an object. Object methods are furthermore classified into *change* and *output* types. In an *output* type method, data is derived from an object. Change type methods are furthermore classified into *full* and *partial* types. In a full type method, whole data in an object is fully changed. In a partial type method, a part of data in an object is changed. A method op_1 *semantically dominates* op_2 on an object o ($op_1 \succeq\succeq op_2$) iff an application considers op_1 to be more significant than op_2. op_1 is *semantically equivalent* with op_2 ($op_1 \cong op_2$) if $op_1 \succeq\succeq op_2$ and $op_2 \succeq\succeq op_1$. op_1 is *more semantically significant* than op_2 ($op_1 \succ\succ op_2$) if $op_1 \succeq\succeq op_2$ and $op_1 \ncong op_2$. op_1 and op_2 are *semantically uncomparable* ($op_1 \,\|\, op_2$) iff neither $op_1 \succeq\succeq op_2$ nor $op_2 \succeq\succeq op_1$.

Definition. A method op_1 is *more significant* than another method op_2 on an object o ($op_1 \succ op_2$) iff (1) op_1 is a *class* type and op_2 is an *object* type, (2) op_1 is a *change* type and op_2 is an *output* one, (3) op_1 is a *full change* type and op_2 is an *partial* one, or (4) op_1 and op_2 are a same object type and $op_1 \succ\succ op_2$.

A method op_1 is *significantly equivalent* with op_2 ($op_1 \equiv op_2$) iff op_1 and op_2 are a same type and $op_1 \cong op_2$. op_1 *significantly dominates* op_2 ($op_1 \succeq op_2$) iff $op_1 \succ op_2$ or $op_1 \equiv op_2$. op_1 and op_2 are *significantly uncomparable* ($op_1 \,\|\, op_2$) iff neither $op_1 \succeq op_2$ nor $op_2 \succeq op_1$.

Suppose a file object F supports six methods *create*, *drop*, *modify*, *insert*, *delete*, and *read* as shown in Fig. 1. *modify* $\succ$ *insert* since *modify* is a full change type method and *insert* is a partial change type method.

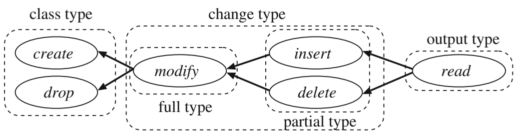

Fig. 1. Significancy of methods.

2.3 Significancy of Roles

In object-based systems, subjects and objects are referred to as entities. Each entity e_i is given one security class $sc(e_i)$ [9]. A security class sc_1 can *flow into* sc_2 ($sc_1 \mapsto sc_2$) iff the information in an entity e_1 of a security class sc_1 can flow into another entity e_2 of a security class sc_2. sc_1 and sc_2 are *equivalent* ($sc_1 \equiv sc_2$) iff $sc_1 \mapsto sc_2$ and $sc_2 \mapsto sc_1$. sc_1 precedes sc_2 ($sc_1 \prec sc_2$) iff $sc_1 \mapsto sc_2$ but $sc_2 \not\mapsto sc_1$. sc_2 dominates sc_1 ($sc_1 \preceq sc_2$) iff $sc_1 \prec sc_2$ or $sc_1 \equiv sc_2$.

Definition. An object o_1 is *more significant* than o_2 ($o_1 \succ o_2$) iff $sc(o_1) \succ sc(o_2)$.

A pair of objects o_1 and o_2 are *significantly equivalent* ($o_1 \equiv o_2$) iff $sc(o_1) \equiv sc(o_2)$. o_1 *significantly dominates* o_2 ($o_1 \succeq o_2$) iff $o_1 \succ o_2$ or $o_1 \equiv o_2$. o_1 and o_2 are *significantly uncomparable* ($o_1 \parallel o_2$) iff neither $sc(o_1) \succeq sc(o_2)$ nor $sc(o_2) \succeq sc(o_1)$.

Definition. Let α_1 and α_2 be a pair of access rights $\langle o_1, op_1 \rangle$ and $\langle o_2, op_2 \rangle$. An access right α_1 is *more significant* than α_2 ($\alpha_1 \succ \alpha_2$) iff (1) $o_1 \succ o_2$, (2) $op_1 \succ op_2$ and $o_1 \equiv o_2$, or (3) $\alpha_1 \succ \alpha_3$ and $\alpha_3 \succ \alpha_2$ for some access right α_3.

A pair of access rights α_1 and α_2 are *significantly equivalent* ($\alpha_1 \equiv \alpha_2$) iff (1) $op_1 \equiv op_2$ and $o_1 = o_2$, or (2) $o_1 \equiv o_2$ and $o_1 \neq o_2$. α_1 *significantly dominates* α_2 ($\alpha_1 \succeq \alpha_2$) iff $\alpha_1 \succ \alpha_2$ or $\alpha_1 \equiv \alpha_2$. α_1 and α_2 are *significantly uncomparable* ($\alpha_1 \parallel \alpha_2$) iff neither $\alpha_1 \succeq \alpha_2$ nor $\alpha_2 \succeq \alpha_1$.

Let **A** be a set of access rights. An access right β is *maximally reachable* from another access right α ($\beta \leftharpoonup \alpha$) iff $\beta \succeq \alpha$ and there is no access right γ such that $\gamma \succeq \beta$ in **A**.

Definition. A role R_1 *significantly dominates* R_2 ($R_1 \succeq R_2$) iff (1) for some access right α in R_2, there is an access right $\beta \in R_1$ - R_2 such that $\beta \leftharpoonup \alpha$ in $R_1 \cup R_2$ and (2) for every access right $\beta \in R_1$, there is no access right $\alpha \in R_2$ such that $\alpha \leftharpoonup \beta$ in $R_1 \cup R_2$.

A role R_1 is *significantly equivalent* with R_2 ($R_1 \equiv R_2$) iff $R_1 \succeq R_2$ and $R_2 \succeq R_1$. R_1 and R_2 are *significantly uncomparable* ($R_1 \parallel R_2$) iff neither $R_1 \succeq R_2$ nor $R_2 \succeq R_1$. A *least upper bound* $R_1 \sqcup R_2$ is a role R_3 such that $R_3 \succeq R_1$ and $R_3 \succeq R_2$ and there is no role R_4 such that $R_3 \succeq R_4 \succeq R_1$ and $R_3 \succeq R_4 \succeq R_2$. A *greatest lower bound* $R_1 \sqcap R_2$ is similarly defined. Here, $R_1 \sqcap \cdots \sqcap R_m \preceq R_i \preceq R_1 \sqcup \cdots \sqcup R_m$ holds but $R_1 \cap \cdots \cap R_m \preceq R_i \preceq R_1 \cup \cdots \cup R_m$ may not hold.

Definition. Let $\mathbf{R}_1$ and $\mathbf{R}_2$ be families of roles. $\mathbf{R}_1$ *significantly dominates* $\mathbf{R}_2$ ($\mathbf{R}_1 \succeq \mathbf{R}_2$) iff $\sqcap_{R \in \mathbf{R}_1} R \succeq \sqcup_{R \in \mathbf{R}_2} R$. $\mathbf{R}_1$ and $\mathbf{R}_2$ are *significantly equivalent* ($\mathbf{R}_1 \equiv \mathbf{R}_2$) iff $\mathbf{R}_1 \succeq \mathbf{R}_2$ and $\mathbf{R}_2 \succeq \mathbf{R}_1$. $\mathbf{R}_1$ and $\mathbf{R}_2$ are *significantly uncomparable* ($\mathbf{R}_1 \parallel \mathbf{R}_2$) iff neither $\mathbf{R}_1 \succeq \mathbf{R}_2$ nor $\mathbf{R}_2 \succeq \mathbf{R}_1$.

2.4 Significancy of Transactions

We first define the dominant relation of subjects with respect to the significancy of roles and authorized relation:

Definition. A subject Sub_i *precedes* Sub_j on a role R ($Sub_i \Rightarrow_R Sub_j$) iff Sub_i grants R to Sub_j or $Sub_i \Rightarrow_R Sub_k \Rightarrow_R Sub_j$ for some subject Sub_k.

A pair of subjects Sub_i and Sub_j are *equivalent* on R ($Sub_i \equiv_R Sub_j$) iff $Sub_i \Rightarrow_R Sub_j$ and $Sub_j \Rightarrow_R Sub_i$. A pair of subjects Sub_i and Sub_j are *independent* with respect to R ($Sub_i \parallel_R Sub_j$) iff neither $Sub_i \Rightarrow_R Sub_j$ nor $Sub_j \Rightarrow_R Sub_i$.

Subject-oriented (SO) Dominant Relation. A subject Sub_i *subject-oriented* (*SO*) *dominates* Sub_j ($Sub_i \succeq^{SO} Sub_j$) iff (1) $Srole_i \succeq Srole_j$, (2) $Sub_i \Rightarrow_R Sub_j$ for some role $R \in Srole_{ij}$ and $Sub_j \not\Rightarrow_R Sub_i$ for every $R \in Srole_{ij}$ if $Srole_i \,\|\, Srole_j$, or (3) $Sub_i \succeq^{SO} Sub_k \succeq^{SO} Sub_j$ for some subject Sub_k.

Suppose each subject Sub_i issues a transaction T_t with purpose $Prole_t$ ($\subseteq Srole_i$). We define the dominant relation of subjects with respect to the significancy of purposes of transactions.

Purpose-oriented (PO) Dominant Relation. For a pair of transactions T_t and T_u issued by subjects Sub_i and Sub_j, respectively, the subject Sub_i *purpose-oriented* (*PO*) dominates Sub_j ($Sub_i \succeq^{PO}_{tu} Sub_j$) with respect to the purposes of transactions T_t and T_u iff (1) $Prole_t \succeq Prole_u$ or (2) $Sub_i \Rightarrow_R Sub_j$ for some role $R \in Prole_{tu}$ and $Sub_j \not\Rightarrow_R Sub_i$ for every $R \in Prole_{tu}$ if $Prole_t \,\|\, Prole_u$.

The SO-dominant relation $\succeq^{SO}$ is transitive. However, the PO-dominant relation $\succeq^{PO}_{tu}$ is not transitive since the PO-dominant relation $\succeq^{PO}_{tu}$ is only defined for a pair of transactions T_t and T_u.

We define the SO- and PO-dominant relations of transactions based on the SO- and PO-dominant relations of subjects issuing the transactions, respectively.

Definition. For a pair of conflicting transactions T_i and T_j,

- T_i *SO-dominates* T_j ($T_i \succeq^{SO} T_j$) iff $Sub_i \succeq^{SO} Sub_j$.
- T_i *PO-dominates* T_j ($T_i \succeq^{PO} T_j$) iff $Sub_i \succeq^{PO}_{tu} Sub_j$.

Let $\succeq^D$ show a dominant relation of transactions for a dominant type $D \in$ {SO, PO}. A pair of transactions T_i and T_j are *equivalent* ($T_i \equiv^D T_j$) iff $T_i \succeq^D T_j$ and $T_j \succeq^D T_i$. A pair of transactions T_i and T_j are *independent* ($T_i \,\|^D\, T_j$) iff neither $T_i \succeq^D T_j$ nor $T_j \succeq^D T_i$.

2.5 Meaningless Methods

Let $\mathbf{T}$ be a set $\{T_1, ..., T_k\}$ ($k \geq 1$) of transactions. Let SH be a *schedule* of the transactions in a set $\mathbf{T}$ where every transaction in the schedule SH is serially performed in the following serial precedent relation:

Definition. A transaction T_i *serially precedes* T_j in a schedule SH ($T_i \rightarrow_{SH} T_j$) iff (1) $T_i \succeq^D T_j$, or (2) $tid(T_i) < tid(T_j)$ if $T_i \,\|^D\, T_j$ or $T_i \equiv^D T_j$.

A schedule SH is a totally ordered set $\langle \mathbf{T}, \rightarrow_{SH} \rangle$. A schedule SH is serializable iff the serial precedent relation $\rightarrow_{SH}$ is acyclic. A schedule $SH = \langle \mathbf{T}, \rightarrow_{SH} \rangle$ is *legal* iff $T_1 \rightarrow_{SH} T_2$ if $T_1 \succeq^D T_2$, or $tid(T_1) < tid(T_2)$ if $T_1 \,\|^D\, T_2$ or $T_1 \equiv^D T_2$ for every pair of T_1 and T_2 in $\mathbf{T}$. In order to make a schedule *legal*, methods from transactions are required to be buffered until all the transactions are initiated.

Definition. A schedule $SH = \langle \mathbf{T}, \rightarrow_{SH} \rangle$ is *RS-partitioned* into the subschedules $SH_f = \langle \mathbf{T}_f, \rightarrow_{SH_f} \rangle$ (f = 1, ..., d):

1. $\mathbf{T}_f \cap \mathbf{T}_g = \phi$ for every pair of subschedules H_f and H_g and $\mathbf{T}_1 \cup \cdots \cup \mathbf{T}_d = \mathbf{T}$.

2. $T_1 \rightarrow_{SH_f} T_2$ if $T_1 \succeq^D T_2$, or $tid(T_1) < tid(T_2)$ if $T_1 \|^D T_2$ or $T_1 \equiv^D T_2$ for every pair of transactions T_1 and T_2 in each SH_f.
3. $T_1 \rightarrow_{SH} T_2$ if $T_1 \rightarrow_{SH_f} T_2$ for every pair of transactions T_1 and T_2 in each SH_f.
4. For every pair of subschedules SH_f and SH_g, if $T_{f1} \rightarrow_{SH} T_{g1}$ for some pair of transactions T_{f1} in SH_f and T_{g1} in SH_g, there is no pair of transactions T_{f2} in SH_f and T_{g2} in SH_g such that $T_{g2} \rightarrow_{SH} T_{f2}$.

Definition. A schedule SH of $\mathbf{T}$ is *RS-serializable* with respect to subschedules SH_1, ..., SH_d iff SH is RS-partitioned into the subschedules SH_1, ..., SH_d.

It is straightforward for the following theorem to hold.

Theorem. A history SH is serializable if SH is RS-serializable with respect to some RS-partition SH_1, ..., SH_d of SH.

Suppose a schedule SH is RS-partition into the subschedules SH_1, ..., SH_d.

Definition. A method op_1 *serially precedes* op_2 in a subschedule SH_f ($op_1 \rightarrow_{SH_f} op_2$) iff (1) the methods op_1 and op_2 are issued by a same transaction T_i and op_1 is issued before op_2, (2) the methods op_1 and op_2 are issued by a pair of transactions T_i and T_j, respectively, and $T_i \rightarrow_{SH_f} T_j$, or (3) $op_1 \rightarrow_{SH_f} op_3 \rightarrow_{SH_f} op_2$ for some method op_3.

Let $SH_f^{o_h}$ be a *local subschedule* of methods which are performed on an object o_h in a subschedule SH_f.

Definition. A method op_1 *serially precedes* another method op_2 in a local subschedule $SH_f^{o_h}$ ($op_1 \rightarrow_{SH_f}^{o_h} op_2$) iff $op_1 \rightarrow_{SH_f} op_2$.

Suppose an object o_h supports six methods as shown in Fig. 1 and a method *insert* serially precedes another method *modify* in a local subschedule $SH_f^{o_h}$ (*insert* $\rightarrow_{SH_f}^{o_h}$ *modify*) on the object o_h. Here, the *insert* method is not required to be performed on the object o_h if the *modify* method is surely performed on the object o_h, i.e. the *modify* method can *absorb* the *insert* method.

Definition

– A full change method op_1 *absorbs* another partial change method op_2 in a local subschedule $SH_f^{o_h}$ on an object o_h if $op_2 \rightarrow_{SH_f}^{o_h} op_1$, and there is no class or output method op' such that $op_2 \rightarrow_{SH_f}^{o_h} op' \rightarrow_{SH_f}^{o_h} op_1$, or op_1 absorbs op'' and op'' absorbs op_2 for some method op''.
– An output method op_1 *absorbs* another output method op_2 in a local subschedule $SH_f^{o_h}$ on an object o_h if $op_2 \rightarrow_{SH_f}^{o_h} op_1$, and there is no class or change method op' such that $op_2 \rightarrow_{SH_f}^{o_h} op' \rightarrow_{SH_f}^{o_h} op_1$, or op_1 absorbs op'' and op'' absorbs op_2 for some method op''.
– A class method op_1 for dropping an object o_h *absorbs* another change method op_2 in a local subschedule $SH_f^{o_h}$ on an object o_h if $op_2 \rightarrow_{SH_f}^{o_h} op_1$, and there is no class or output method op' such that $op_2 \rightarrow_{SH_f}^{o_h} op'' \rightarrow_{SH_f}^{o_h} op_1$, or op_1 absorbs op'' and op'' absorbs op_2 for some method op''.

A method op is not required to be performed on an object o_h if the method op is absorbed by another method op' in a local subschedule $SH_f^{o_h}$.

Definition. A method op is *meaningless* iff the method op is absorbed by another method op' in the local subschedule $sh_f^{o_h}$ of an object o_h.

2.6 Power Consumption Model of a Server

In class methods and change type methods, data is written into an object. On the other hand, in output type methods, data is read from an object. In this paper, methods are classified into *read* (r) and *write* (w) types of methods. 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 r and w methods on a server s_t at time τ, respectively. Here, $P_t(\tau) = RP_t(\tau) \cup WP_t(\tau)$. Let $r_{ti}(o_h)$ and $w_{ti}(o_h)$ be methods issued by a transaction T_i to read and write data in an object o_h on a server s_t, respectively. By each method $r_{ti}(o_h)$ in a set $RP_t(\tau)$, data is read in an object o_h at rate $RR_{ti}(\tau)$ [B/sec] at time τ. By each method $w_{ti}(o_h)$ in a set $WP_t(\tau)$, data is written in an object o_h 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 $fr_t(\tau) \cdot maxRR_t$ and $fw_t(\tau) \cdot maxWR_t$, respectively. Here, $fr_t(\tau)$ and $fw_t(\tau)$ are degradation ratios for read and write methods, respectively. $0 \leq fr_t(\tau) \leq 1$ and $0 \leq fw_t(\tau) \leq 1$. The degradation ratios $fr_t(\tau)$ and $fw_t(\tau)$ are given as $\frac{1}{|RP_t(\tau)|+rw_t \cdot |WP_t(\tau)|}$ and $\frac{1}{wr_t \cdot |RP_t(\tau)|+|WP_t(\tau)|}$, respectively. 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 $r_{ti}(o_h)$ and $w_{ti}(o_h)$ show how much amount of data are read and written in an object o_h by the methods $r_{ti}(o_h)$ and $w_{ti}(o_h)$ at time τ, respectively. Suppose that a pair of methods $r_{ti}(o_h)$ and $w_{ti}(o_h)$ start on a server s_t at time st_{ti}, respectively. At time st_{ti}, the read laxity $lr_{ti}(\tau)$ is rb_h [B] where rb_h is the size of data in an object o_h. The write laxity $lw_{ti}(\tau)$ is wb_h [B] where wb_h is the size of data to be written in an object o_h. Here, $lr_{ti}(\tau) = rb_h$ - $\Sigma_{\tau=st_{ti}}^{\tau} RR_{ti}(\tau)$ and $lw_{ti}(\tau) = wb_h$ - $\Sigma_{\tau=st_{ti}}^{\tau} WR_{ti}(\tau)$.

Let $E_t(\tau)$ be the electric power consumption [W] of a server s_t at time τ. $maxE_t$ and $minE_t$ show the maximum and minimum electric power consumption [W] of the server s_t, respectively. *The power consumption model for a storage server* (*PCS* model) [10] is proposed. 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:

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

The server s_t consumes the electric power RE_t [W] if $|WP_t(\tau)| = 0$ and $|RP_t(\tau)| \geq 1$. The server s_t consumes the electric power WE_t [W] if $|WP_t(\tau)| \geq 1$ and $|RP_t(\tau)| = 0$. 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 $|RP_t(\tau)| \geq 1$. Otherwise, a server s_t consumes the minimum electric power $minE_t$. Here, $minE_t \leq RE_t \leq WRE_t(\alpha) \leq WE_t \leq maxE_t$. The processing power consumption $PE_t(\tau)$ [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 $TPE_t(\tau_1, \tau_2) = \Sigma_{\tau=\tau 1}^{\tau_2} PE_t(\tau)$.

3 Energy-Efficient Purpose Ordering (EEPO) Scheduler

We discuss *energy efficient purpose ordering* (*EEPO*) scheduler to not only make transactions RS-serializable with PO-dominant relation but also reduce the total energy consumption of a server cluster S. A transaction T_i first sends a *begin* request b_i to every target object. Then, the transaction T_i issues methods and lastly issues either a *commit* (cm_i) or *abort* (ab_i) request to the objects. Each client cl_s manipulates a variable cf_s where initially $cf_s = 1$. Each client cl_s periodically sends a *fence* message k to make an RS-partition, which carries $k.f$ ($= cf_s$). Each time a client cl_s sends a fence message k, $cf_s = cf_s + 1$ in the client cl_s. Each object o_h has a variable f_h where initially $f_h = 1$. Each time an object o_h receives a fence message k where $k.f = f_h$ from every client, $f_h = f_h + 1$ in the object o_h. Transactions whose begin requests are received before a fence message k compose an RS-partition and are sorted in the serial precedence relation $\rightarrow_{SH_f}$.

There are a set RQ_h of *local receipt queues* RQ_{h1}, ..., RQ_{hl}, a *global receipt queue* GRQ_h, and an *auxiliary global receipt queue* $AGRQ_h$ for each object o_h. On receipt of a method op_i from a transaction T_i initiated on a client cl_s, the method op_i is enqueued into a local receipt queue RQ_{hs} for the client cl_s ($s = 1, ..., l$) on an object o_h. Begin requests and fence messages are moved to $AGRQ_h$ to make an RS-partition. Transactions in an RS-partition are serialized in the serial precedence relation $\rightarrow_{SH_f}$. Methods are moved to GRQ_h and are performed in the serial precedence relation $\rightarrow_{SH_f}$. The following conditions have to be satisfied to realize the RS-serializability:

Role-Based Serializability (RS) Conditions

1. Methods in every global receipt queue GRQ_h are sorted in the serial precedence relation $\rightarrow_{SH_f}$.
2. For a method op_i from a transaction T_i, if op_i precedes a method op_j conflicting with op_i from another transaction T_j in some GRQ_h, op'_i from T_i precedes a method op'_j conflicting with op'_i from T_j in every $GRQ_{h'}$.

The EEPO scheduler for an object o_h handles methods to realize the RS-serializability by the **RS** procedure as shown in Algorithm 1.

Suppose an RS-partition SH_f is composed of begin requests preceding a fence k where $k.f$ is the minimum in $AGRQ_h$ of an object o_h. Each begin request b_i

Algorithm 1. RS procedure

Input: GRQ_h, $AGRQ_h$, $\{RQ_{h1}, ..., RQ_{hl}\}$.
Output: RS-partitioned GRQ_h.
/* The following procedures are used to manipulate a queue Q for a method *op*. */
– **top**(Q): a top element in Q. – **enqueue**(op, Q): op is enqueued into Q.
– **tail**(Q): a tail element in Q. – **dequeue**(Q): a top element in Q is dequeued.
– **RSsort**(op, Q, e_1, e_2): op is inserted between elements e_1 and e_2 in Q, and requests between elements e_1 and e_2 in Q are sorted in the serial precedence relation $\rightarrow_{SH_f}$.

```
procedure RS(GRQ_h, AGRQ_h, {RQ_h1, ..., RQ_hl})
    if there is a fence k where k.f = f_h in every RQ_hs then
        for every local receipt queue RQ_hs do
            while top(RQ_hs) ≠ k do
                op_i ← dequeue(RQ_hs);
                if op_i = b_i then enqueue(op_i, AGRQ_h);
                else /* op_i is not a begin request b_i. */
                    if b_i is between the top of AGRQ_h and a fence k' then
                        RSsort(op_i, GRQ_h, top(GRQ_h), k');
                    else if b_i is between a pair of fences k' and k'' then
                        RSsort(op_i, GRQ_h, k', k'');
                    else if b_i is between a fence k' and the tail of AGRQ_h)  then
                        RSsort(op_i, GRQ_h, k', tail(GRQ_h));
                    end if
                end if
            end while
            op_i ← dequeue(RQ_hs);  op_i is removed; /* fence k is removed. */
        end for
        enqueue(k, AGRQ_h);  enqueue(k, GRQ_h);  f_h = f_h + 1;
    end if
end procedure
```

of a transaction T_i holds a transaction identifier $tid(T_i)$ and list L_i of methods issued by the transaction T_i. Here, begin requests in the RS-partition SH_f can be totally ordered in the serial precedent relation $\rightarrow_{SH_f}$ of transactions. Hence, a local subschedule $SH_f^{o_h}$ of methods can be created on an object o_h by sorting lists of methods held in begin requests according to the serial precedent relation $\rightarrow_{SH_f}$. Methods in GRQ_h are performed on an object o_h by the **Delivery** procedure as shown in Algorithm 2.

4 Evaluation

We evaluate the *EEPO* scheduler in terms of the total electric energy consumption [J] of a server cluster S and the average execution time [msec] of each transaction compared with the RO [3] and EERO [8] schedulers. We consider a homogeneous server cluster S composed of five servers $s_1, ..., s_5$. Every server s_t $(t = 1, ..., 5)$ follows the same data access model and power consumption

Algorithm 2. Delivery procedure

Input: GRQ_h. $\mathbf{E}_h$ and $\mathbf{TE}_h$ are sets of current methods and transactions on o_h.
Output: Performing methods on an object o_h.

```
/* Procedures to check methods and transactions being performed on o_h. */
– Mcompatible(op, E_h): true if E_h = φ or a method op does not conflict with
  every method in E_h, otherwise false.
– Tcompatible(T(op), TE_h): true if TE_h = φ or a transaction T(op) issuing a
  method op does not conflict with every transaction in TE_h, otherwise false.
– Meaningless(op): true if a method op is meaningless in the local subschedule
  SH_f^{o_h} and there is a method op' in a global receipt queue GRQ_h where the method
  op' absorbs the method op, otherwise false.

procedure DELIVERY(GRQ_h)
    op ← top(GRQ_h);
    if op ≠ fence then
        if Mcompatible(op, E_h) and Tcompatible(T(op), TE_h then
            op ← dequeue(GRQ_h);  E_h ← E_h ∪ {op};
            if T(op) ∉ TE_h then TE_h = TE_h ∪ {T(op)}; end if
            if Meaningless(op) then
                E_h = E_h - {op};
                if op = cm_i or op = ab_i then TE_h = TE_h - {T(op)}; end if
            else perform(op);
            end if
        end if
    else
        if E_h = φ and TE_h = φ then
            every begin request b_i preceding the fence op in AGRQ_h is removed;
            op is removed from GRQ_h and AGRQ_h; /* the fence op is removed. */
        end if
    end if
end procedure
```

model as shown in Table 1. Parameters of each server s_t are given based on the experimentations [10]. There are five objects o_1, ..., o_5 in a system. Each server s_t holds one object o_h ($t = h$). The size of data in each object o_h is randomly selected between 50 and 80 [MB]. Each object o_h supports six types of methods as shown in Fig. 1. There are five subjects Sub_1, ..., Sub_5. There are three roles R_1, R_2, and R_3 owned by Sub_1, where $R_1 \succeq R_2 \succeq R_3$. Here, $Sub_1 \succeq_{R_i} Sub_2$, $Sub_1 \succeq_{R_i} Sub_3$, $Sub_1 \succeq_{R_i} Sub_4$, $Sub_1 \succeq_{R_i} Sub_5$ for every role R_i (i = 1, ..., 3). $Sub_2 \succeq_{R_3} Sub_4$ and $Sub_3 \succeq_{R_3} Sub_5$. $Srole_1 = \{R_1, R_2, R_3\}$, $Srole_2 = Srole_3 = \{R_2, R_3\}$, and $Srole_4 = Srole_5 = \{R_3\}$. Here, $Srole_1 \succeq^{SO} Srole_2 = Srole_3 \succeq^{SO} Srole_4 = Srole_5$. The subject Sub_1 issues transactions with a purpose $Prole_1 = \{R_3\}$ ($\subseteq Srole_1$). Other transactions are assigned with purposes as $Prole_2 = Prole_3 = \{R_2, R_3\}$ and $Prole_4 = Prole_5 = \{R_3\}$. Here, $Prole_2 = Prole_3 \succeq^{PO} Prole_1 = Prole_4 = Prole_5$. Each subject Sub_i (i = 1, ..., 5) initiates a same number l ($0 \leq l \leq$ 1,200) of transactions on each of five clients cl_1, ..., cl_5. The total number tn ($= l \cdot 5$) ($0 \leq tn \leq$ 6,000)

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]

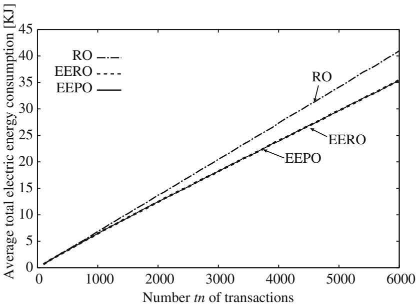

Fig. 2. Average total energy consumption [KJ] of a server cluster.

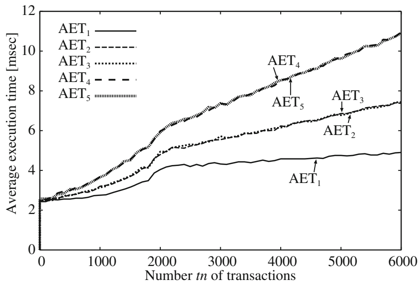

Fig. 3. Average execution time AET_i in the RO scheduler.

of transactions are issued to manipulate objects. We assume each transaction issues full change, partial change, and output methods. The total amount of data of an object o_h are fully written and read by each full change and output methods, respectively. On the other hand, a half size of data of an object o_h are written into the object o_h by partial change methods. Each transaction issues three methods randomly selected from twenty methods on the five objects. The starting time of each transaction T_i is randomly selected in a unit of one second between 1 and 3600 [sec].

Figure 2 shows the average total electric energy consumption [KJ] of the server cluster S to perform the number tn of transactions in the RO, EERO, and EEPO schedulers. The average total electric energy consumption of the server cluster S in the EEPO algorithm is almost the same as the EERO scheduler. In the EERO and EEPO schedulers, meaningless methods which are not required to be performed on each object are omitted. As a results, the average total electric energy consumption of the server cluster S can be more reduced in the EERO and EEPO schedulers than the RO scheduler.

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 [msec]. Figures 3, 4, and 5 show the average execution time AET_i of each transaction issued by the same subject Sub_i in the server cluster S to perform the total number tn of transactions in the RO, EERO, and EEPO schedulers, respectively. In the RO and EERO schedulers, transactions are ordered based on the SO-dominant relations. As a result, the average execution time AET_1 of transactions issued by the subject Sub_1 is the minimum in the RO and EERO schedulers since the subject Sub_1 is more significant than the other subjects. Following Figs. 3 and 4, the more significant subject is, the shorter average execution time a transaction

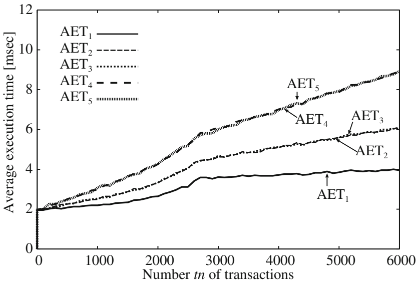

Fig. 4. Average execution time AET_i in the EERO scheduler.

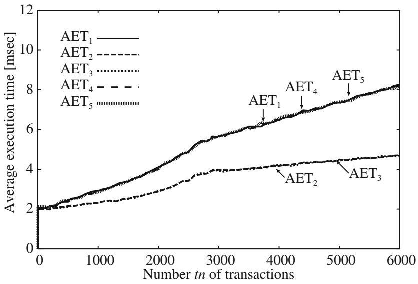

Fig. 5. Average execution time AET_i in the EEPO scheduler.

issued by the subject implies in the RO and EERO schedulers. On the other hand, transactions are ordered based on the PO-dominant relations in the EEPO scheduler. As a result, the average execution time of AET_1 of transactions issued by the subject Sub_1 is the same as the average execution times AET_4 and AET_5 issued by subjects Sub_4 and Sub_5, respectively, since $Prole_1 = Prole_4 = Prole_5$. Following Fig. 5, the more significant transaction with respect to purpose is, the shorter average execution time a transaction implies in the EEPO scheduler.

Following the evaluation, the more significant transactions with respect to purposes, the earlier performed in the EEPO scheduler. The average total electric energy consumption of a server cluster can be more reduced in the EEPO and EERO schedulers than the RO scheduler. The average total electric energy consumption of a server cluster in the EEPO scheduler is the same as the EERO scheduler.

5 Concluding Remarks

In this paper, we newly proposed the EEPO scheduler to not only serialize multiple conflicting transactions in the significant order of purposes assigned to transactions but also reduce the total electric energy consumption of a server cluster by omitting meaningless methods. We evaluated the EEPO scheduler compared with the RO and EERO schedulers. The evaluation results show the total electric energy consumption of a server cluster in the EEPO scheduler is the same as the EERO scheduler. The total electric energy consumption of a server cluster can be more reduced in the EEPO and EERO scheduler than the RO scheduler. In addition, the more significant transactions with respect to purposes, the earlier performed in the EEPO scheduler.

References

1. Ferraiolo, D.F., Kuhn, D.R., Chandramouli, R.: Role Based Access Control. Artech House, Norwood (2005)

2. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. IEEE Comput. **29**(2), 38–47 (1996)
3. Enokido, T.: Role-based serializability using role ordering schedulers. J. Interconnect. Netw. **7**(4), 437–450 (2006)
4. 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
5. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, Boston (1987)
6. Gray, J.N.: Notes on database operating systems. Lect. Notes Comput. Sci. **60**, 393–481 (1978)
7. Sandhu, R.S.: Lattice-based access control models. IEEE Comput. **26**(11), 9–19 (1993)
8. Enokido, T., Duolikun, D., Takizawa, M.: Energy-efficient role ordering scheduler. In: Proceedings of the 20th International Conference on Network-Based Information Systems (NBiS-2017), pp. 78–90 (2017)
9. Denning, D.E., Denning, P.J.: Cryptography and Data Security. Addison-Wesley Publishing Company, Boston (1982)
10. 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)