Evolutionary Intelligence (2020) 13:71–82
https://doi.org/10.1007/s12065-018-0157-1

SPECIAL ISSUE

Energy consumption laxity-based quorum selection for distributed object-based systems

Tomoya Enokido[1] · Dilawaer Duolikun[2] · Makoto Takizawa[2]

Received: 7 May 2018 / Revised: 12 June 2018 / Accepted: 18 June 2018 / Published online: 29 June 2018

Abstract

In object based systems, an object is an unit of computation resource. Distributed applications are composed of multiple objects. Objects in an application are replicated to multiple servers in order to increase reliability, availability, and performance. On the other hand, the large amount of electric energy is consumed in a system compared with non-replication systems since multiple replicas of each object are manipulated on multiple servers. In this paper, the energy consumption laxity-based quorum selection (ECLBQS) algorithm is proposed to construct a quorum for each method issued by a transaction so that the total electric energy consumption of servers to perform methods can be reduced in the quorum based locking protocol. The total electric energy consumption of servers, the average execution time of each transaction, and the number of aborted transactions are shown to be more reduced in the ECLBQS algorithm than the random algorithm in evaluation.

Keywords Quorum-based locking protocol · Data management · Energy-aware information systems · Object-based systems · Replication

1 Introduction

In current information systems, various kinds of distributed applications like data centers [1, 2] are realized on scalable, high performance, and fault-tolerant computing systems like cloud computing systems [2–4]. These distributed applications are composed of multiple objects in object-based frameworks [5] like CORBA [6]. Each object is an unit of computation resource like a file. An object is an encapsulation of data and methods to manipulate the data in the object. An object is allowed to manipulate only through methods supported by the object. A transaction is an atomic sequence of methods [7] to manipulate objects. Once a server which performs a method issued by a transaction on an object stops by fault [7], the transaction aborts. In order to realize reliable and available application services, objects are replicated [8] on multiple servers. Replicas of each object are distributed on multiple servers and have to be mutually consistent. In order to keep replicas of each object mutually consistent, conflicting methods issued by multiple transactions are required to be serializable [9]. In the two-phase locking (2PL) protocol [7], all the replicas of an object for a *write* method and one of the replicas for a *read* method are locked before manipulating the object according to the *read-one-write-all* scheme [8] to keep the replicas of each object mutually consistent. However, every replica of each object has to be locked for every write method issued in a system, the 2PL protocol is not efficient for write-dominant applications. In order to reduce the overhead to perform write methods, the quorum-based locking (QBL) protocols [5, 10] are proposed. In the quorum-based locking protocol, some numbers nQ^r and nQ^w of replicas of an object, called *quorum numbers*, are locked for *read* and *write* methods, respectively. The quorum numbers nQ^r and nQ^w for each object have to be "$nQ^r + nQ^w > N$" where N is the total number of replicas. Subsets of replicas locked for read and write methods are referred to as *read* and *write quorums*, respectively. The more number of write methods are issued

✉ Tomoya Enokido
eno@ris.ac.jp

Dilawaer Duolikun
dilewerdolkun@gmail.com

Makoto Takizawa
makoto.takizawa@computer.org

1 Faculty of Business Administration, Rissho University, 4-2-16, Osaki, Shinagawa, Tokyo 141-8602, Japan

2 Department of Advanced Sciences, Faculty of Science and Engineering, Hosei University, 3-7-2, Kajino-cho, Koganei-shi, Tokyo 184-8584, Japan

in a system, the smaller number of write quorum can be taken in the QBL protocol. As a result, the overhead to perform write methods can be reduced in the QBL protocol than the 2PL protocol. On the other hand, the large amount of electric energy is consumed in a system than non-replication systems since methods issued to each object are performed on multiple replicas stored in multiple servers. It is critical to not only provide the reliable and available application service but also reduce the total electric energy consumption of an object-based system as discuss in the Green computing [1, 2, 11, 12].

In this paper, the *energy consumption laxity-based quorum selection* (*ECLBQS*) algorithm is proposed to construct a quorum for each method issued by a transaction in the quorum based locking protocol so that the total electric energy consumption of servers to perform methods can be reduced. The ECLBQS algorithm is evaluated in terms of the total electric energy consumption of servers, the average execution time of each transaction, and the number of aborted transactions compared with the random algorithm in homogeneous and heterogeneous server clusters. The evaluation results show the total electric energy consumption of servers, the average execution time of each transaction, and the number of aborted transactions in the ECLBQS algorithm can be maximumly reduced to 38.7, 54.1, and 47.6% of the random algorithm in a homogeneous server cluster, respectively. In addition, the total electric energy consumption of servers, the average execution time of each transaction, and the number of aborted transactions in the ECLBQS algorithm can be maximumly reduced to 38.1, 41.6, and 45.1% of the random algorithm in a heterogeneous server cluster, respectively.

In Sect. 2, we show related studies on energy-efficient information systems. In Sect. 3, we discuss the system model, data access model, and power consumption model of a server. In Sect. 4, we discuss the ECLBQS algorithm. In Sect. 5, we evaluate the ECLBQS algorithm compared with the random algorithm.

2 Related works

Various kinds of approaches are proposed to realize energy aware information systems [1, 2, 11, 13–15]. In order to realize energy aware information systems, it is necessary to define the power consumption model and the computation model of a server to perform application processes. The electric power of a server depends on not only hardware components [14] but also types of application processes performed on the server. In our previous studies, application processes are classified into computation [16, 17, 19–21], communication [23], storage [18], and general types [22]. The electric power of a server to perform each type of application processes is measured and the power consumption models to perform each type of application processes are proposed by abstracting parameters which dominate the electric power of a server based on the experiments. The *power consumption model for a storage server* (*PCS* model) [18] to concurrently perform storage and computation processes are proposed. Storage processes read and write data in objects stored in a server. Computation processes mainly consume CPU resources of a server. In this paper, a transaction issues read and write methods to manipulate replicas of objects. We assume only read and write methods are performed on a server. Read and write methods are performed as storage processes [18] in a server. Therefore, the electric power consumption model of a server to perform multiple read and write methods issued by transactions is defined based on the PCS model in this paper.

The *quorum-based locking* (*QBL*) protocol [10] is proposed to not only keep replicas of objects mutually consistent but also reduce the overhead to perform write methods. In the QBL protocol, each object supports simple read and write methods. The *quorum based object locking* (*QOL*) protocol [5] which extends the traditional QBL protocol with simple read and write methods to the object-based system with abstract methods is proposed to not only keep the replicas of objects mutually consistent but also reduce the number of replicas to be locked in a system. By using the QOL protocol, the total number of replicas to be locked in a system can be reduced than the traditional QBL protocol. However, the QOL and QBL protocols do not consider to reduce the total electric energy consumption of servers to perform methods on multiple replicas of objects. In this paper, we propose the *energy consumption laxity-based quorum selection* (*ECLBQS*) algorithm which extends the traditional QBL protocol to not only keep the replicas of objects mutually consistent but also reduce the total electric energy consumption of servers to perform read and write methods on replicas of objects.

3 System model

3.1 Objects and transactions

A system is composed of multiple servers $s_1, \ldots, s_n$ ($n \geq 1$) interconnected in reliable networks. This means messages can be delivered to their destinations in the sending order and without message loss. Let S be a cluster of servers $s_1, \cdots, s_n$ ($n \geq 1$), i.e. $S = \{s_1, \cdots, s_n\}$. Let O be a set of objects $o_1, \cdots, o_m$ ($m \geq 1$), i.e. $O = \{o_1, \cdots, o_m\}$. Each object o_h is a unit of computation resource like a file and is an encapsulation of data and methods to manipulate the data in the object o_h. In this paper, we assume each object o_h supports *read* (r) and *write* (w) methods for manipulating data in the object o_h. Let

$op(o_h)$ be a state obtained by performing a method op ($\in \{r, w\}$) on an object o_h. A pair of methods op_1 and op_2 on an object o_h are *compatible* if and only if (iff) a result obtained by performing the methods op_1 and op_2 does not depend on the computation order, i.e. $op_1 \circ op_2(o_h) = op_2 \circ op_1(o_h)$. Otherwise, a method op_1 *conflicts* with another method op_2. For example, a pair of read methods r_1 and r_2 are compatible on an object o_h. On the other hand, a write method conflicts with read and write methods on an object o_h.

Each object o_h is replicated on multiple servers to make a system more reliable and available. Replicas of each object o_h are distributed on multiple servers in a server cluster S. Let $R(o_h)$ be a set of replicas $o_h^1, \cdots, o_h^l$ ($l \geq 1$) [8] of an object o_h. Let $nR(o_h)$ be the total number of replicas of an object o_h, i.e. $nR(o_h) = |R(o_h)|$. Let S_h be a subset of servers which hold a replica of an object o_h in a server cluster S ($S_h \subseteq S$). For example, a server cluster S is composed of five servers $s_1, \cdots, s_5$ as shown in Fig. 1. There are three objects o_1, o_2, and o_3. There are three replicas of each object o_h, i.e. $nR(o_h) = 3$ ($h = 1, 2, 3$). Here, $S_1 = \{s_1, s_2, s_5\}$ since replicas o_1^1, o_1^2, and o_1^3 of the object o_1 are stored in the servers s_1, s_2, and s_5.

A *transaction* is an atomic sequence of methods [7]. A transaction T_i is initiated in a client cl_i and issues read and write methods to manipulate replicas of objects. Multiple conflicting transactions are required to be *serializable* [7, 9] to keep replicas of each object mutually consistent. Let $\mathbf{T}$ be a set $\{T_1, \cdots, T_k\}$ ($k \geq 1$) of transactions initiated in a system. Let H be a schedule of the transactions in a set $\mathbf{T}$ of transactions, i.e. a sequence of methods performed in a set $\mathbf{T}$ of transactions. A transaction T_i *precedes* another transaction T_j ($T_i \rightarrow_H T_j$) in a schedule H iff a method op_i issued by the transaction T_i is performed before another method op_j issued by the transaction T_j and the method op_i conflicts with the method op_j. A schedule H is serializable iff the precedent relation $\rightarrow_H$ is acyclic.

3.2 Quorum-based locking protocol

In this paper, multiple conflicting transactions are serialized by using the *quorum-based locking* protocol [5, 10]. 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. Q_h^{op} is referred to as a *quorum* of the method op on the object o_h ($Q_h^{op} \subseteq R(o_h)$). Let nQ_h^{op} be the *quorum number* of a method op on a object o_h, i.e. $nQ_h^{op} = |Q_h^{op}|$. The quorums have to satisfy the following constraints:

[*Quorum 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. $nQ_h^w > nR(o_h)/2$.

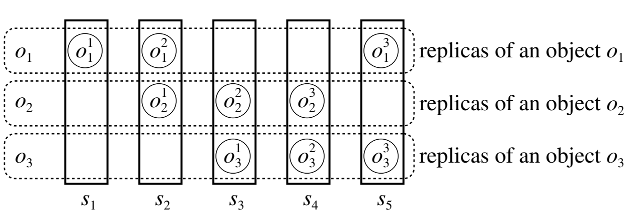

Fig. 1 A server cluster S and replicas of objects

Every quorum surly includes at least one newest replica o_h^q of each object o_h by satisfying the quorum constraints. Let $\mu(op)$ be a *lock mode* of a method op ($\in \{r, w\}$). If a method op_1 is compatible with another method op_2 on an object o_h, a lock mode $\mu(op_1)$ is compatible with another lock mode $\mu(op_2)$. Otherwise, a lock mode $\mu(op_1)$ conflicts with another lock mode $\mu(op_2)$.

A transaction T_i locks replicas of an object o_h by using the following quorum-based locking (QBL) protocol [5, 10] before manipulating the replicas with a method op.

[*Quorum-based locking protocol]*

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

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

3.3 Data access model

Methods which are being performed and already terminate on a server 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. Data in a replica o_h^q is read at rate $RR_{ti}(\tau)$ [B/sec] by each method $r_{ti}(o_h^q)$ in a set $RP_t(\tau)$ at time τ. Data in a replica o_h^q is written at rate $WR_{ti}(\tau)$ [B/sec] by each method $w_{ti}(o_h^q)$ in a set $WP_t(\tau)$ 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. At time τ, the read rate $RR_{ti}(\tau)$ ($\leq maxRR_t$) and write rate $WR_{ti}(\tau)$ ($\leq maxWR_t$) for each read and write method performed on a server s_t are given as follows:

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

Here, $fr_t(\tau)$ and $fw_t(\tau)$ are degradation ratios of the read rate $RR_{ti}(\tau)$ and write rate $WR_{ti}(\tau)$ at time τ, respectively. Here, $0 \leq fr_t(\tau) \leq 1$ and $0 \leq fw_t(\tau) \leq 1$. The degradation ratios $fr_t(\tau)$ and $fw_t(\tau)$ depends on the number of current read and write methods performed on a server s_t at time τ. The degradation ratios $fr_t(\tau)$ and $fw_t(\tau)$ at time τ are given as follows:

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

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

The *read laxity* $lr_{ti}(\tau)$ [B] and *write laxity* $lw_{ti}(\tau)$ [B] of methods $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$ show how much amount of data are read and written in a replica o_h^q by the methods $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$ at time τ, respectively. Suppose that methods $r_{ti}(o_h^q)$ and $w_{ti}(o_h^q)$ start on a server s_t at time st_{ti}, respectively. At time st_{ti}, the read laxity $lr_{ti}(\tau) = rb_h^q$ [B] where rb_h^q is the size of data in a replica o_h^q to be read by a method $r_{ti}(o_h^q)$. 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 by a method $w_{ti}(o_h^q)$. The read laxity $lr_{ti}(\tau)$ and write laxity $lw_{ti}(\tau)$ at time τ are given as $lr_{ti}(\tau) = rb_h^q - \Sigma_{\tau=st_{ti}}^{\tau} RR_{ti}(\tau)$ and $lw_{ti}(\tau) = wb_h^q - \Sigma_{\tau=st_{ti}}^{\tau} WR_{ti}(\tau)$, respectively.

3.4 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 $minE_t$ denote the maximum and minimum electric power (W) of the server s_t, respectively. The *power consumption model for a storage server* (*PCS* model) [18] to concurrently perform storage and computation processes on a server is proposed. In this paper, we assume only read and write 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 read and write methods at time τ is given as follows:

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

A server s_t consumes the minimum electric power $minE_t$ (W) if no method is performed on the server s_t, i.e. the electric power in the idle state of the server s_t. The server s_t consumes the electric power RE_t (W) if $|WP_t(\tau)| = 0$ and $|RP_t(\tau)| \geq 1$, i.e. only read methods are performed on the server s_t. The server s_t consumes the electric power WE_t (W) if $|WP_t(\tau)| \geq 1$ and $|RP_t(\tau)| = 0$, i.e. only write methods are performed on the server s_t. The server s_t consumes the electric power $WRE_t(\alpha)$ (W) $= \alpha \cdot RE_t + (1-\alpha) \cdot WE_t$ (W) where $\alpha = |RP_t(\tau)|/(|RP_t(\tau)| + |WP_t(\tau)|)$ if $|WP_t(\tau)| \geq 1$ and $|RP_t(\tau)| \geq 1$, i.e. both at least one read method and at least one write 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 between time τ_1 an τ_2 is given as follows:

$$TE_t(\tau_1, \tau_2) = \sum_{\tau=\tau 1}^{\tau_2} E_t(\tau). \tag{4}$$

The processing power $PE_t(\tau)$ (W) of a server s_t at time τ is $E_t(\tau) - minE_t$. The total processing energy $TPE_t(\tau_1, \tau_2)$ (J) of a server s_t between time τ_1 and τ_2 is given as follows:

$$TPE_t(\tau_1, \tau_2) = \sum_{\tau=\tau 1}^{\tau_2} PE_t(\tau) = \sum_{\tau=\tau 1}^{\tau_2} (E_t(\tau) - minE_t). \tag{5}$$

The total processing energy consumption laxity $tpecl_t(\tau)$ shows how much electric energy a server s_t has to consume to perform every current read and write 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 $TPECL_t$ procedure:

```
TPECL_t(τ) {
   if RP_t(τ) = φ and WP_t(τ) = φ, return(0);
   laxity = E_t(τ) - minE_t; /* PE_t(τ) of a server s_t at time τ */
   for each read method r_ti(o_h^q) in RP_t(τ), {
      lr_ti(τ + 1) = lr_ti(τ) - RR_ti(τ);
      if lr_ti(τ + 1) = 0, RP_t(τ + 1) = RP_t(τ) - {r_ti(o_h^q)};
   } /* for end. */
   for each write method w_ti(o_h^q) in WP_t(τ), {
      lw_ti(τ + 1) = lw_ti(τ) - WR_ti(τ);
      if lw_ti(τ + 1) = 0, WP_t(τ + 1) = WP_t(τ) - {w_it(o_h^q)};
   } /* for end. */
   return(laxity + TPECL_t(τ + 1));
}
```

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

4 The ECLBQS algorithm

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

```
ECLBQS(op, o_h, τ) { /* op ∈ {r, w} */
   S_h^op = φ;
   while (nQ_h^op > 0) {
      for each server s_t in S_h, {
         if op = read method, RP_t(τ) = RP_t(τ) ∪ {op};
         else WP_t(τ) = WP_t(τ) ∪ {op}; /* op = write method*/
         TPE_t(τ) = TPECL_t(τ);
         } /* for end. */
         server = a server s_t where TPE_t(τ) is the minimum;
         S_h^op = S_h^op ∪ {server};
         S_h = S_h - {server};
         nQ_h^op = nQ_h^op - 1;
   } /* while end. */
   return(S_h^op);
}
```

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

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

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

Table 1 Parameters of each server s_t in a server cluster S

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

(t = 1,..., 5)

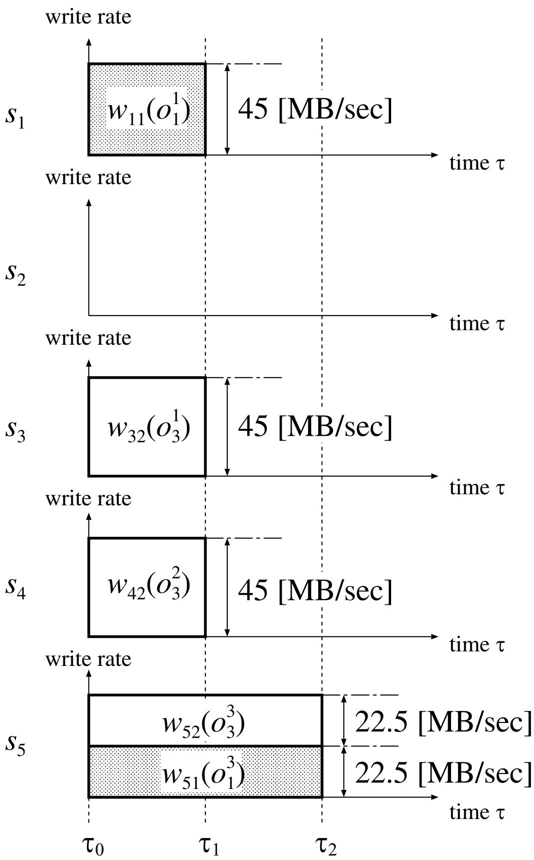

Fig. 2 Execution of methods

laxity $lw_{52}(\tau_1)$ is 22.5 MB at time τ_1. At time τ_1, a pair of the write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ are still concurrently performed on the server s_5 at write rate 22.5 MB/s. The write laxity $lw_{51}(\tau_2)$ gets 0 at time time τ_2 since $lw_{51}(\tau_1) - WR_{51}(\tau_1)$ = 22.5 MB − 22.5 MB = 0. Similarly, the write laxity lw_{52} (τ_2) gets 0 at time τ_1. Here, a pair of write methods $w_{51}(o_1^3)$ and $w_{52}(o_3^3)$ terminate at time τ_2 on the server s_5.

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

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

5 Evaluation

5.1 Environment

The ECLBQS algorithm is evaluated in terms of the total processing energy consumption of a server cluster, the average execution time of each transaction, and the average number of aborted transactions compared with the random algorithm. In the random algorithm, a quorum for each method is randomly selected. In this evaluation, we consider a homogeneous (S) and heterogeneous (H) server clusters which are composed of fifteen servers $s_1, \dots, s_{15}$ ($n = 15$), respectively, as shown in Tables 2 and 3.

In the homogeneous server cluster $S = \{s_1, \dots, s_{15}\}$, every server s_t ($t = 1, \dots, 15$) 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 [18]. The maximum read and write rates (B/s) on every server s_t are 80 and 45 MB/s, respectively, i.e. $maxRR_t$ = 80 MB/s and $maxWR_t = 45$ MB/s. The parameters rw_t and wr_t in the read and write degradation ratios $fr_t(\tau)$ and fw_t (τ) of every server s_t are 0.5, respectively. The minimum electric power $minE_t$ (W) of every server s_t is 39 W. The electric power WE_t where only write methods are performed on every server s_t is 53 W. The electric power RE_t where only read methods are performed on every server s_t is 43 W.

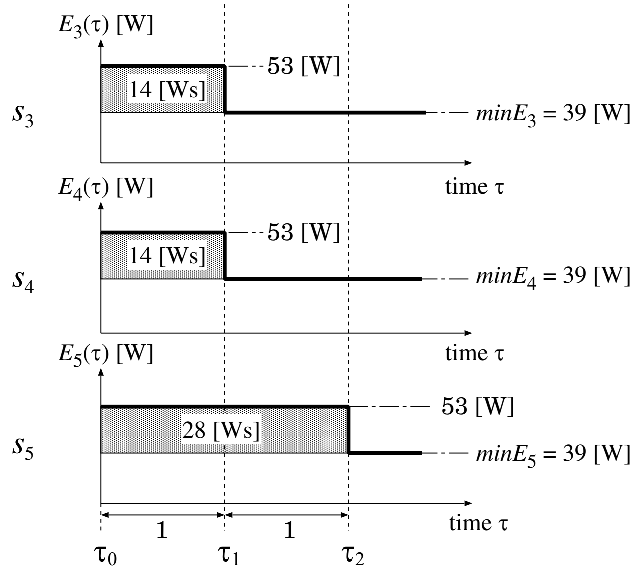

Fig. 3 Total processing energy laxity (J)

In the heterogeneous server cluster $H = \{s_1, \cdots, s_{15}\}$, we consider three types of servers, $Type_1 = \{s_1, \cdots, s_5\}$, $Type_2 = \{s_6, \cdots, s_{10}\}$, $Type_3 = \{s_{11}, \cdots, s_{15}\}$ as shown in Table 3. In each server type, every parameter of data access model and power consumption model is the same. For example, parameters of five servers $s_1, \cdots, s_5$ in $Type_1$ are the same as a server s_t ($t = 1, \cdots, 15$) in the homogeneous server cluster S. The maximum read and write rates (B/s) of the $Type_2$ server s_6 are 120 and 67 MB/s, respectively, i.e. $maxRR_6 = 120$ MB/s and $maxWR_6 = 67$ MB/s. $fr_6(\tau) = 0.5$, $fw_6(\tau) = 0.5$, $minE_6 = 59$ W, $WE_6 = 80$ W, and $RE_6 = 64$ W. Parameters of $Type_2$ servers $s_7, \cdots, s_{10}$ are the same as the server s_6. The maximum read and write rates (B/s) of the $Type_3$ server s_{11} are 136 and 76 MB/s, respectively, i.e. $maxRR_{11} = 136$ MB/s and $maxWR_{11} = 76$ MB/s. $fr_{11}(\tau) = 0.5$, $fw_{11}(\tau) = 0.5$, $minE_{11} = 45$ W, $WE_{11} = 60$ W, and $RE_{11} = 49$ W. Parameters of $Type_3$ servers $s_{12}, \cdots, s_{15}$ are the same as the server s_{11}.

There are fifty objects $o_1, \cdots, o_{50}$ in a system, i.e. $O = \{o_1, \cdots, o_{50}\}$. Each object o_h supports *read* (r) and *write* (w) methods. The size of data in each object o_h is randomly selected between 50 and 250 MB. The total number of replicas for every object is seven, i.e. $R(o_h) = \{o_h^1, \cdots, o_h^7\}$ and $nR(o_h) = 7$ ($h = 1, \cdots, 50$). Replicas of each object are randomly distributed on fifteen servers in the homogeneous S and heterogeneous H server clusters, respectively. The quorum number nQ_h^w of a write method on every object o_h is four, i.e. $nQ_h^w = 4$. The quorum number nQ_h^r of a read method on every object o_h is four, $nQ_h^r = 4$.

The total number m of transactions $T_1, \cdots, T_m$ ($0 \leq m \leq 600$) are issues to manipulate objects in a system. Each transaction issues three methods randomly selected from one-hundred methods on the fifty objects. By each read and write method issued by a transaction T_i to a replica o_h^q of an object o_h, the total amount of data of the replica o_h^q are fully read and written, respectively. The starting time of each transaction T_i is randomly selected in a unit of 1 s between 1 and 360 s.

5.2 The average execution time of each transaction

The ECLBQS algorithm is evaluated in terms of the average execution time (s) of each transaction in the homogeneous S and heterogeneous H server clusters, respectively, compared with the random algorithm. Let ET_i^β be the execution time (s) of a transaction T_i in a server cluster $\beta \in \{S$ (homogeneous), H (heterogeneous)$\}$ where the transaction T_i commits. For example, suppose a transaction T_i starts at time st_i and commits at time et_i in a server cluster β. Here, the execution time ET_i^β of the transaction T_i is $et_i - st_i$ (s). In this evaluation, the execution time ET_i^β for each transaction T_i is measured five times for each total number m of transactions ($0 \leq m \leq 600$). Let $ET_i^{\beta,tm}$ be the execution time ET_i^β obtained in tmth simulation. The average execution time AET^β (s) of each transaction for each total number m of transactions is calculated as $\sum_{tm=1}^{5} \sum_{i=1}^{m} ET_i^{\beta,tm}/(m \cdot 5)$.

Table 2 Homogeneous server cluster S

Server s_t	$maxRR_t$	$maxWR_t$	rw_t	wr_t	$minE_t$	WE_t	RE_t
$s_1, \cdots, s_{15}$	80 MB/s	45 MB/s	0.5	0.5	39 W	53 W	43 W

($t = 1, \ldots, 15$)

Table 3 Heterogeneous server cluster H

Server s_t	$maxRR_t$	$maxWR_t$	rw_t	wr_t	$minE_t$	WE_t	RE_t
$s_1, \cdots, s_5$	80 MB/s	45 MB/s	0.5	0.5	39 W	53 W	43 W
$s_6, \cdots, s_{10}$	120 MB/s	67 MB/s	0.5	0.5	59 W	80 W	64 W
$s_{11}, \cdots, s_{15}$	136 MB/s	76 MB/s	0.5	0.5	45 W	60 W	49 W

($t = 1, \ldots, 15$)

Figure 4 shows the average execution time AET^S (s) of each transaction in the homogeneous server cluster S to perform the total number m of transactions in the ECLBQS and random algorithms. In the ECLBQS and random algorithms, the average execution time AET^S increases as the total number m of transactions increases since more number of transactions are concurrently performed. For $0 < m \leq 600$, the average execution time AET^S of each transaction can be more shorter in the ECLBQS algorithm than the random algorithm. This means that the data access resources in the server cluster S can be more efficiently utilized in the ECLBQS algorithm than the random algorithm. For example, for $m = 100$, the average execution time AET^S of each transaction in the homogeneous server cluster S in the ECLBQS and random algorithms are 9.1 and 19.9 s, respectively. This means the average execution time AET^S of each transaction in the homogeneous server cluster S in the ECLBQS algorithm can be maximumly reduced to 54.1% of the random algorithm. In Fig. 4, the average execution time AET^S of each transaction in the homogeneous server cluster S in the ECLBQS algorithm can be averagely reduced to 25.1% of the random algorithm for $0 \leq m \leq 600$.

Figure 5 shows the average execution time AET^H (s) of each transaction in the heterogeneous server cluster H to perform the total number m of transactions in the ECLBQS and random algorithms. For $0 < m \leq 600$, the average execution time AET^H of each transaction can be more shorter in the ECLBQS algorithm than the random algorithm as similar to the results obtained in the homogeneous server cluster S. For $m = 150$, the average execution time AET^H of each transaction in the heterogeneous server cluster H in the ECLBQS and random algorithms are 14.8 and 25.4 s, respectively. This means the average execution time AET^H of each transaction in the heterogeneous server cluster H in the ECLBQS algorithm can be maximumly reduced to 41.6% of the random algorithm. In Fig. 5, the average execution time AET^H of each transaction in the heterogeneous server cluster H in the ECLBQS algorithm can be averagely reduced to 24.3% of the random algorithm for $0 \leq m \leq 600$.

5.3 The average number of aborted instances of each transaction

The ECLBQS algorithm is evaluated in terms of the average number of aborted transactions in the homogeneous S and heterogeneous H server clusters, respectively, compared with the random algorithm. A transaction T_i aborts if the transaction T_i could not lock every replica in an r-quorum Q_h^r or w-quorum Q_h^w. If a transaction T_i aborts, the transaction T_i is restarted after δ time units in this evaluation. The time units δ (s) is randomly selected between 20 and 30 s ($20 \text{ s} \leq \delta \leq 30 \text{ s}$) 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 T_i commits. Let AT_i^β be the number of aborted instances of a transaction T_i in a server cluster $\beta \in \{S \text{ (homogeneous)}, H \text{ (heterogeneous)}\}$. The number of aborted instances AT_i^β for each transaction T_i is measured five times for each total number m of transactions ($0 \leq m \leq 600$). Let $AT_i^{\beta,tm}$ be the number of aborted transaction instances AT_i^β of a transaction T_i in a server cluster β obtained in tmth simulation. The average number of aborted instances AAT^β of each transaction in a server cluster β for each total number m of transactions is calculated as $\sum_{tm=1}^{5} \sum_{i=1}^{m} AT_i^{\beta,tm}/(m \cdot 5)$.

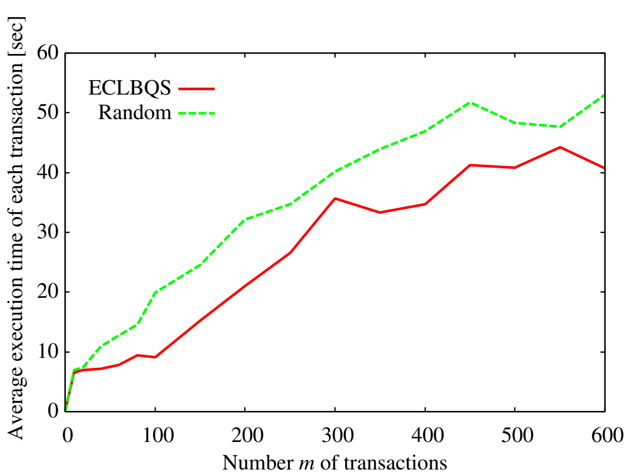

Fig. 4 Average execution time AET^S (s) of each transaction in the homogeneous server cluster S

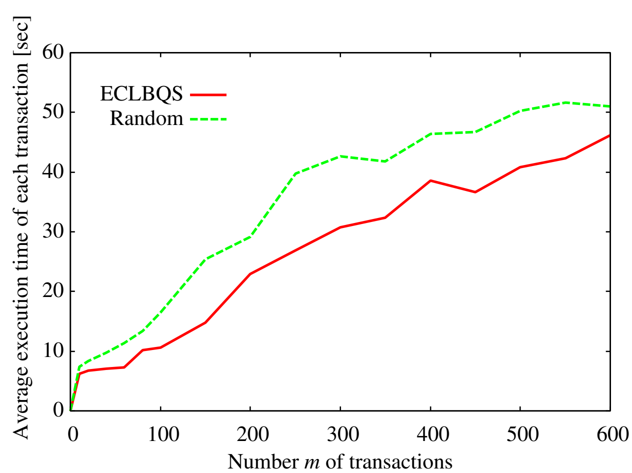

Fig. 5 Average execution time AET^H (s) of each transaction in the heterogeneous server cluster H

Figure 6 shows the average number of aborted instances AAT^S of each transaction in the homogeneous server cluster S to perform the total number m of transactions in the ECLBQS and random algorithms. In the ECLBQS and random algorithms, the average number of aborted instances AAT^S of each transaction increases as the total number m of transactions increases. The more number of transactions are concurrently performed, the more number of transactions cannot lock replicas. Hence, the number of aborted instances of each transaction increases in the ECLBQS and random algorithms. For $80 \leq m \leq 600$, the average number of aborted instances AAT^S of each transaction can be more reduced in the ECLBQS algorithm than the random algorithm. The data access resources in the homogeneous server cluster S can be more efficiently utilized in the ECLBQS algorithm than the random algorithm. Hence, the average execution time of each transaction in the homogeneous server cluster S can be shorter in the ECLBQS algorithm than the random algorithm as shown in Fig. 4. As a result, the number of aborted instances AAT^S of each transaction can be more reduced in the ECLBQS algorithm than the random algorithm since the number of transaction to be concurrently performed can be reduced in the ECLBQS algorithm than the random algorithm. For example, for $m = 400$, the average number of aborted instances AAT^S of each transaction in the homogeneous server cluster S in the ECLBQS and random algorithms are 108 and 205, respectively. This means the average number of aborted instances AAT^S of each transaction in the ECLBQS algorithm can be maximumly reduced to 47.6% of the random algorithm. In Fig. 6, the average number of aborted instances AAT^S of each transaction in the ECLBQS algorithm can be averagely reduced to 38.9% of the random algorithm for $0 \leq m \leq 600$.

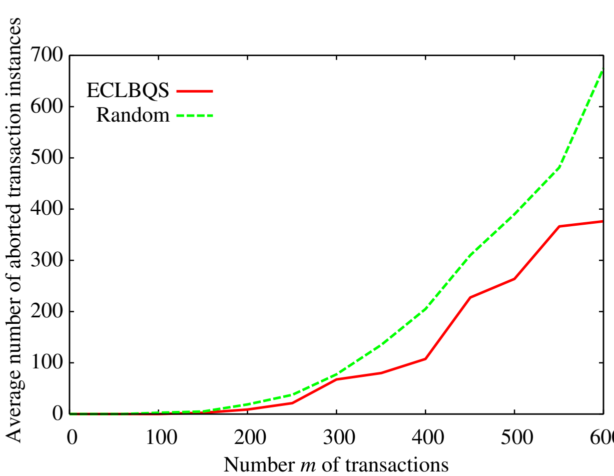

Fig. 6 Average number of aborted transaction instances AAT^S in the homogeneous server cluster S

Figure 7 shows the average number of aborted instances AAT^H of each transaction in the heterogeneous server cluster H to perform the total number m of transactions in the ECLBQS and random algorithms. For $100 \leq m \leq 600$, the average number of aborted instances AAT^H of each transaction can be more reduced in the ECLBQS algorithm than the random algorithm as similar to the results obtained in the homogeneous server cluster S. For $m = 450$, the average number of aborted instances AAT^H of each transaction in the heterogeneous server cluster H in the ECLBQS and random algorithms are 158 and 288, respectively. This means the average number of aborted instances AAT^H of each transaction in the ECLBQS algorithm can be maximumly reduced to 45.1% of the random algorithm. In Fig. 7, the average number of aborted instances AAT^H of each transaction in the ECLBQS algorithm can be averagely reduced to 41.2% of the random algorithm for $0 \leq m \leq 600$.

5.4 The average total processing energy of a server cluster

The ECLBQS algorithm is evaluated in terms of the average total processing energy (J) of the homogeneous S and heterogeneous H server clusters, respectively, to perform the total number m of transactions. Let $TEC^{\beta,tm}$ be the total processing energy (J) of a server cluster $\beta \in \{S$ (homogeneous), H (heterogeneous)$\}$ to perform the number m of transactions ($0 \leq m \leq 600$) obtained in the tmth simulation. The total processing energy $TEC^{\beta,tm}$ is measured five times for each number m of transactions. The average total processing energy $ATEC^{\beta}$ (J) of a server cluster β is calculated as $\sum_{tm=1}^{5} TEC^{\beta,tm}/5$ for each number m of transactions.

Figure 8 shows the average total processing energy $ATEC^S$ of the homogeneous server cluster S to perform the total number m of transactions in the ECLBQS and random algorithms. In the ECLBQS and random algorithms, the average total processing energy $ATEC^S$ of the homogeneous server cluster S increases as the number m of transactions increases. For $0 < m \leq 600$, the average total processing energy $ATEC^S$ of the homogeneous server cluster S can be more reduced in the ECLBQS algorithm than the random algorithm. In the ECLBQS algorithm, each time a transaction T_i issues a method op ($\in \{r, w\}$) to manipulate an object o_h, the transaction T_i selects a subset nS_h^{op} ($\subseteq S_h$) of nQ_h^{op} servers which hold a replica o_h^q of the object o_h to construct a quorum Q_h^{op} for the method op so that the total processing energy laxity of a server cluster S is the minimum. In addition, the processing energy to perform each transaction and aborted instances of each transaction can be more reduced in the ECLBQS algorithm than the random algorithm since the average execution time and the number of aborted instances of each transaction can be more reduced in the ECLBQS algorithm than the random algorithm. As a result, the average total processing

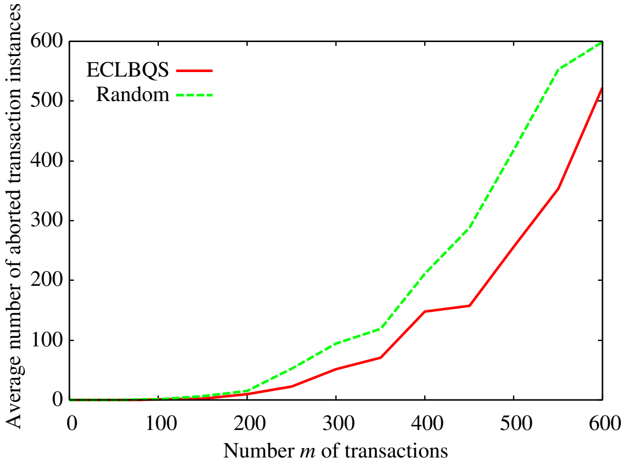

Fig. 7 Average number of aborted transaction instances AAT^H in the heterogeneous server cluster H

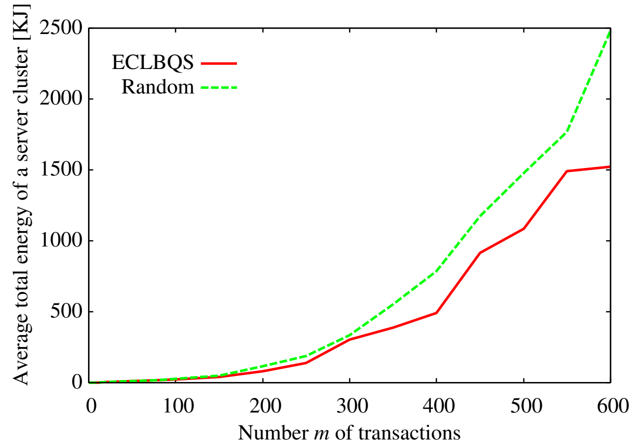

Fig. 8 Average total processing energy $ATEC^S$ (KJ) in the homogeneous server cluster S

energy $ATEC^S$ of the server cluster S to perform the number m of transactions can be more reduced in the ECLBQS algorithm than the random algorithm. For example, for $m = 600$, the average total processing energy of the homogeneous server cluster S in the ECLBQS and random algorithm are 1521 and 2483 KJ, respectively. This means the average processing energy of the homogeneous server cluster S in the ECLBQS algorithm can be maximumly reduced to 38.7% of the random algorithm. In Fig. 8, the average processing energy of the homogeneous server cluster S in the ECLBQS algorithm can be averagely reduced to 18.3% of the random algorithm for $0 \leq m \leq 600$.

Figure 9 shows the average total processing energy $ATEC^H$ of the heterogeneous server cluster H to perform the total number m of transactions in the ECLBQS and random algorithms. For $0 < m \leq 600$, the average total processing energy $ATEC^H$ of the heterogeneous server cluster H can be more reduced in the ECLBQS algorithm than the random algorithm as similar to the results obtained in the homogeneous server cluster S. For example, for $m = 250$, the average total processing energy of the heterogeneous server cluster H in the ECLBQS and random algorithm are 150 and 243 KJ, respectively. This means the average processing energy of the heterogeneous server cluster H in the ECLBQS algorithm can be maximumly reduced to 38.1% of the random algorithm. In Fig. 9 the average processing energy of the heterogeneous server cluster H in the ECLBQS algorithm can be averagely reduced to 15.5% of the random algorithm for $0 \leq m \leq 600$.

5.5 Summary of evaluation

In the evaluation, the average total processing energies of the homogeneous S and heterogeneous H server clusters to perform the total number m ($0 \leq m \leq 600$) of transactions are shown to be more reduced in the ECLBQS algorithm than the random algorithm. The average total processing energies of the homogeneous S and heterogeneous H server clusters in the ECLBQS algorithm can be maximumly reduced to 38.7 and 38.1% of the random algorithm, respectively, for $0 \leq m \leq 600$. The average total processing energies of the homogeneous S and heterogeneous H server clusters in the ECLBQS algorithm can be averagely reduced to 18.3 and 15.5% of the random algorithm, respectively, for $0 \leq m \leq 600$. The average execution time and number of aborted instances of each transaction in the homogeneous S and heterogeneous H server cluster are shown to be more reduced in the ECLBQS algorithm than the random algorithm. The average execution time and number of aborted instances of each transaction in the homogeneous server cluster S in the ECLBQS algorithm can be maximumly reduced to 54.1 and 47.6% of the random algorithm, respectively, for $0 \leq m \leq 600$. The average execution time and number of aborted instances of each transaction in the homogeneous server cluster S in the ECLBQS algorithm can be averagely reduced to 25.1 and 38.9% of the random algorithm, respectively, for $0 \leq m \leq 600$. The average execution time and number of aborted instances of each transaction in the heterogeneous server cluster H in the ECLBQS algorithm can be maximumly reduced to 41.6 and 45.1% of the random algorithm, respectively, for $0 \leq m \leq 600$. The average execution time and number of aborted instances of each transaction in the heterogeneous server cluster H in the ECLBQS algorithm can be maximumly reduced to 24.3 and 41.2% of the random algorithm, respectively, for $0 \leq m \leq 600$. Following the evaluation, the ECLBQS algorithm is more useful than the random algorithm.

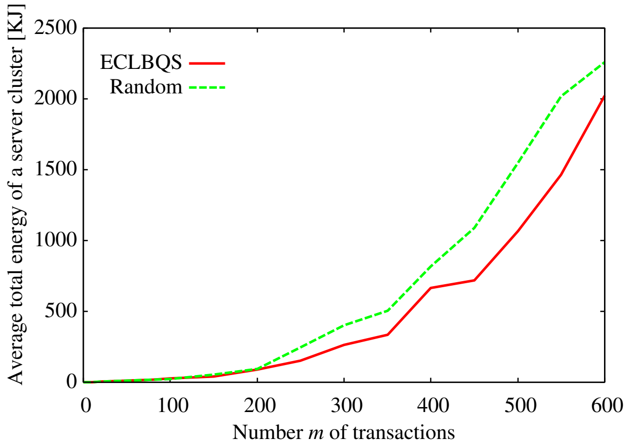

Fig. 9 Average total processing energy $ATEC^H$ (KJ) in the heterogeneous server cluster H

6 Concluding remarks

In this paper, we newly proposed the *energy consumption laxity - based quorum selection* (*ECLBQS*) algorithm to select a quorum for each method issued by a transaction in the quorum based locking (QBL) protocol so that the total electric energy consumption of a server cluster to perform read and write methods issued by transactions can be reduced. We evaluated the ECLBQS algorithm in terms of the average total processing energy consumption of a server cluster, the average execution time of each transaction, and the number of aborted instances of each transaction in homogeneous and heterogeneous server clusters compared with the random algorithm. The evaluation results show the average total processing energy consumption of a server cluster, the average execution time of each transaction, and the average number of aborted instances of each transaction can be more reduced in the ECLBQS algorithm than the random algorithm. Following the evaluation, the ECLBQS algorithm is more useful than the random algorithm.

We are now defining meaningless methods which are not required to be performed on each replica of an object based on the precedent relation and semantics of methods. In future works, we improve the ECLBQS algorithm to furthermore reduce the total electric energy consumption of a server cluster by omitting meaningless methods.

References

1. Natural Resources Defense Council (NRDS) (2014) Data center efficiency assessment—scaling up energy efficiency across the data center industry: evaluating key drivers and barriers. http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf. Accessed 3 Apr 2015
2. Natural Resources Defense Council (NRDS) (2012) Is cloud computing always greener? Finding the most energy and carbon efficient information technology solutions for small- and medium-sized organizations. http://www.nrdc.org/energy/files/cloud-computing-efficiency-IB.pdf. Accessed 6 Apr 2015
3. Enokido T, Duolikun D, Takizawa M (2015) An extended improved redundant power consumption laxity-based (EIRPCLB) algorithm for energy efficient server cluster systems. World Wide Web 18(6):1629–1630
4. Tomimori M, Sugawara S (2017) Content sharing method using expected acquisition rate in hybrid peer-to-peer networks with cloud storages. Int J Space Based Situated Comput 7(4):187–196
5. Tanaka K, Hasegawa K, Takizawa M (2000) Quorum-based replication in object-based systems. J Inf Sci Eng 16(3):317–331
6. Object Management Group Inc. (2012) Common object request broker architecture (CORBA) specification, version 3.3, Part 1—Interfaces. http://www.omg.org/spec/CORBA/3.3/Interfaces/PDF. Accessed 24 Apr 2017
7. Bernstein PA, Hadzilacos V, Goodman N (1987) Concurrency control and recovery in database systems. Addison-Wesley, Boston
8. Schneider FB (1993) Replication management using the state-machine approach. Distributed systems, 2nd edn. ACM Press, New York
9. Gray JN (1978) Notes on database operating systems. Lect Notes Comput Sci 60:393–481
10. Garcia-Molina H, Barbara D (1985) How to assign votes in a distributed system. J ACM 32(4):814–860
11. Khan S, Kolodziej J, Li J, Zomaya AY (2013) Evolutionary based solutions for green computing. Springer, New York
12. Serhan Z, Diab WB (2016) Energy efficient QoS routing and adaptive status update in WMSNs. Int J Space Based Situated Comput 6(3):129–146
13. Qu X, Peng X (2017) An energy-efficient virtual machine scheduler based on CPU share-reclaiming policy. Int J Grid Util Comput (IJGUC) 6(2):113–120
14. Intel Corporation (2010) Intel Xeon Processor 5600 Series: the next generation of intelligent server processors. http://www.intel.com/content/www/us/en/processors/xeon/xeon-5600-brief.html. Accessed 24 Apr 2017
15. Kaushik A, Vidyarthi DP (2018) A hybrid heuristic resource allocation model for computational grid for optimal energy usage. Int J Grid Util Comput (IJGUC) 9(1):51–74
16. Kataoka H, Nakamura S, Duolikun D, Enokido T, Takizawa M (2017) Multi-level power consumption model and energy-aware server selection algorithm. Int J Grid Util Comput (IJGUC) 8(3):201–210
17. Duolikun D, Enokido T, Takizawa M (2017) An energy-aware algorithm to migrate virtual machines in a server cluster. Int J Grid Util Comput (IJGUC) 7(1):32–42
18. Sawada A, Kataoka H, Duolikun D, Enokido T, Takizawa M (2016) 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
19. Enokido T, Aikebaier A, Takizawa M (2010) A model for reducing power consumption in peer-to-peer systems. IEEE Syst J 4(2):221–229
20. Enokido T, Aikebaier A, Takizawa M (2011) Process allocation algorithms for saving power consumption in peer-to-peer systems. IEEE Trans Ind Electron 58(6):2097–2105
21. Enokido T, Aikebaier A, Takizawa M (2014) An extended simple power consumption model for selecting a server to perform

computation type processes in digital ecosystems. IEEE Trans Ind Inform 10(2):1627–1636
22. Enokido T, Takizawa M (2013) Integrated power consumption model for distributed systems. IEEE Trans Ind Electron 60(2):824–836
23. Enokido T, Takizawa M (2013) The evaluation of the extended transmission power consumption (ETPC) model to perform communication type processes. Computing 95(10–11):1019–1037