Noname manuscript No.
(will be inserted by the editor)

Dynamic Grid Quorum: A Reconfigurable Grid Quorum and Its Power Optimization Algorithm

Munetoshi Ishikawa · Koji Hasebe · Akiyoshi Sugiki · Kazuhiko Kato

Received: date / Accepted: date

Abstract In this paper, we present the dynamic grid quorum, a method for reducing the power consumption of large-scale distributed storage systems. The basic principle of our approach is to skew the workload towards a small number of quorums. This can be realized by the following three techniques. First, our system allows reconfiguration by exchanging nodes without any data migration, so that high-capacity nodes can be reallocated to busier quorums. Second, for more effective skewing of the workload, we introduce the notion of dual allocation, which makes it possible to consider two distinguished allocations in the same grid for write and read quorums. Finally, we present an optimization algorithm to find a pair of strategy and an allocation of nodes, which minimizes power for a given system setting and its workload. We also demonstrate that the dynamic grid quorum saves, on average, 14%–25% energy compared with static configurations, when the intensity of the total workload was changed.

Keywords distributed storage, quorum system, grid quorum, power saving, optimization algorithm, reconfiguration

Correspondence to:M. Ishikawa · K. Hasebe · A. Sugiki · K. Kato
Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
E-mail: munetoshi@osss.cs.tsukuba.ac.jp

K. Hasebe
E-mail: hasebe@iit.tsukuba.ac.jp

A. Sugiki
E-mail: sugiki@cs.tsukuba.ac.jp

K. Kato
E-mail: kato@cs.tsukuba.ac.jp

1 Introduction

Service-oriented computing is a key paradigm in achieving the long-awaited, richer environments in computer science. To achieve such platforms for large-scale business computing systems, reliable, available, and cost-effective data management is required.

The use of quorum systems is a promising approach for achieving reliable data management in such platforms. Compared with the traditional read-one write-all technique, the quorum approach allows flexibility in storage configuration and decentralized management for consistency control. Since the first quorum paper [7] was presented, much research has focused on investigating the possibilities of quorum systems. These studies are wide-ranging, from desired configurations in quorum systems [13,1,3,14] to performance optimization methods that minimize the communication delays [6, 17,12]. On the other hand, little attention has been paid to reducing power consumption in quorum systems, which is necessary in terms of green computing, which has recently become of prime interest to practitioners.

In this paper, we present the *dynamic grid quorum*, a technique for reducing storage power consumption. As its name implies, our approach is based on a grid quorum [3], but the basic principle of our approach is to skew the workload towards a small number of quorums. This can be realized using the following three concepts, which are extensions of our previous work [10]. First, the dynamic grid quorum provides reconfiguration by exchanging nodes without any data migration. Although the node exchange idea comes from a study on tree-based quorums [5], much of the work has had to be adapted for grid quorums. Since we also consider heterogeneity in nodes, this reconfiguration method allows

high-capacity nodes to be allocated to busier quorums to reduce the number of active nodes. Even when exchanging nodes in a grid quorum, our reconfiguration method preserves the consistency of replicated data, namely the *one-copy serializability* (cf. [2]). Second, for more effective skewing of the workload, we introduce the notion of *dual allocation*. In the original grid quorum system [3], the nodes are allocated in a rectangular grid then grouped into write and read quorums in a specific way according to this allocation. The dual allocation is an extension of this grouping method. That is, to group the nodes into quorums, we consider two distinguished allocations in the same grid for write and read quorums. In addition, the one-copy serializability still holds for this extension. Finally, we present an optimization algorithm to find a pair of a strategy (i.e., probability distribution over the quorums, determining which quorum is selected for each write/read request) and an allocation, which minimizes power for a given system setting and its workload. Despite the exponential growth of the possible combinations of quorums and allocations with respect to the number of nodes, our algorithm reduces the computational complexity to find a solution using several techniques.

To evaluate the effectiveness of the proposed dynamic grid quorum with dual allocation, we compared our approach under three alternative configurations: static write-optimized, static read-optimized, and dynamic reconfiguration with no dual allocation (i.e., single allocation). From the simulations, we observed that our dynamic grid quorum with dual allocation saved, on average, 14.1–25.2% energy compared with the static configurations, when the intensity of the total workload was changed. In addition, compared with the case of single allocation, our proposed system saved, on average, 10.4% energy, which indicated that our dual allocation skews workload more effectively.

This paper is organized as follows. Section 2 presents related work. Sections 3 and 4 introduce the dynamic grid quorum and its power consumption model. Section 5 gives the definition of our reconfiguration. Section 6 introduces the notion of dual allocation and its reconfiguration. Section 7 introduces the algorithm and proves that it minimizes the power consumption of the dynamic grid quorum. Section 8 presents the simulation results. Finally, Section 9 concludes the paper and discusses future research.

2 Related Work

There have been a number of attempts to reduce storage power consumption. A commonly-observed feature in many of these techniques is that they adopt the approach of skewing the workload, which is also used in this paper. In MAID [4] and PDC [15], popular data are concentrated on specific disks. In DIV [16], original and redundant data are separated onto different disks, thereby allowing write/read requests to be concentrated on the disks with original data. In RIMAC [20], eRAID [18], and EERAID [11], a data access on a disk in standby mode is transformed into accesses on active disks or caches, and then the required data are reconstructed from the parities obtained during these accesses. In Hibernator [21] and PARAID [19], data are collected or spread to adapt to changes in operational loads.

Although these studies in the literature restrict their scope to storage with a specific kind of central controller to manage the data access, recent work such as Harnik et al., [8] addresses power-saving in large-scale distributed storage system, whose prime application is cloud computing. Our target application is similar to this study, but our main motivation is to achieve reliable, available, and cost-effective distributed data management by taking the quorum approach, which has not been thoroughly investigated.

3 Dynamic Grid Quorum

Our proposed system is based on the grid quorum [3], which is a special kind of write-read coterie [9], in which the nodes are allocated in a rectangular grid. A write-read coterie $\mathcal{C}$ is a pair $\langle \mathcal{C}_W, \mathcal{C}_R \rangle$ of collections of node groups, called *quorums*, where $\mathcal{C}_W$ and $\mathcal{C}_R$ represent *write quorums* and *read quorums*. For readability, we use the notation $\mathcal{C}_A$ to denote $\mathcal{C}_W$ or $\mathcal{C}_R$. (Similar notations are also used for other notions.) Throughout this paper, we fix the numbers of columns and rows of the grid as n and m. Each *intersection point* (often just called *point*) of the i-th column (from the left) and the j-th row (from the bottom) is denoted by (i, j). The set of all points in the grid is denoted by D. We also fix the underlying set of nodes as $U = \{u_1, \ldots, u_{n \cdot m}\}$ whose elements are allocated to the points in the grid by bijective mapping. We refer to this allocation by the mapping $v : D \to U$. We thus define the node at point (i, j) by $v(i, j)$. In addition, the set $\{(i, i) \mid 1 \leq i \leq \min\{n, m\}\}$ is called the *diagonal line*. The point (j, i) is called the *diagonal point* of (i, j).

To formally define the dynamic grid quorum, we introduce the notion of *degree* as follows. Degree, denoted by f, is a pair $\langle c, r \rangle$ of natural numbers with $0 \leq c < \min\{n, m\}$ and $2 \leq r \leq \min\{n, m\}$, respectively. For given degree f, the set of *exchangeable pairs* (denoted by E^f) is defined by $\{\langle (i, j), (j, i) \rangle \mid i \leq c \wedge r \leq$

$j \wedge i < j\}$. In addition, for given E^f, the sets of the first and the second projections of all the elements of E^f (i.e., $\{p \mid \langle p, p' \rangle \in E^f\}$ and $\{p' \mid \langle p, p' \rangle \in E^f\}$, resp.) are called *left-side exchangeable points* (denoted by E^f_{left}) and *lower exchangeable points* (denoted by E^f_{low}). The symbol f may be omitted if it is clear from the context.

We present an example of the exchangeable points below.

Example 1 Fig. 1 depicts the exchangeable points of 6×6 grid with $f = \langle c, r \rangle = \langle 4, 3 \rangle$. In this figure, the left side and the lower exchangeable areas of this case are indicated as the area within the heavy-lines and the area within the dashed-lines.

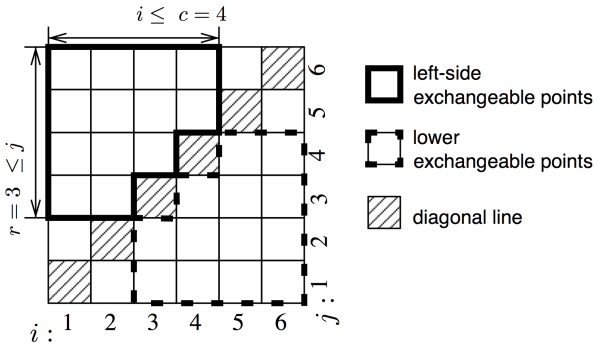

Fig. 1 Exchangeable points with $f = \langle c, r \rangle = \langle 4, 3 \rangle$

Finally, for readability we use the following notations. $Col(i)$ and $Row(j)$ are respectively used to denote the sets of points on the i-th column (i.e., $Col(i) = \{(i, j) \mid 1 \leq j \leq m\}$) and on the j-th row (i.e., $Row(j) = \{(i, j) \mid 1 \leq i \leq n\}$). Notations $v(Col(i))$ and $v(Row(j))$ are used to denote the sets of nodes on $Col(i)$ and $Row(j)$ determined by allocation v. Also, $[i, j]$ is used to denote the set $\bigcup_{k=i}^{j} Col(k)$.

In terms of the notions previously introduced, the dynamic grid quorum is defined as follows.

Definition 1 (Dynamic grid quorum) Let $\mathcal{C}^{v,f} = \langle \mathcal{C}^{v,f}_W, \mathcal{C}^{v,f}_R \rangle$ be a pair of collections of node groups. $\mathcal{C}^{v,f}$ is a *dynamic grid quorum* if

- $\mathcal{C}^{v,f}_W$ is the set of all possible node groups, each of which (say, $Q^{v,f}_W$) satisfies the following condition:

  $Q^{v,f}_W = \{v(i, j) \mid (i, j) \in Col(k) \vee \forall i \exists! j (1 \leq i < k \wedge 1 \leq j \leq m)\}$ for some $k\,(1 \leq k \leq n)$ such that
  - if $(i, j) \in Col(k) \cap E^f_{low}$ then $v(j, i) \in Q^{v,f}_W$;
  - if $v(i, j) \in Q^{v,f}_W$ and $(i, j) \cap E^f_{left}$ then $v(j, i) \in Q^{v,f}_W$.
- $\mathcal{C}^{v,f}_R$ is the set of all possible node groups, each of which (say, $Q^{v,f}_R$) satisfies the following condition:

  $Q^{v,f}_R = \{v(i, j) \mid \forall i \exists! j (1 \leq i \leq n \wedge 1 \leq j \leq m)\}$ such that if $v(i, j) \in Q^{v,f}_R$ and $(i, j) \in E^f_{left}$ then $v(j, i) \in Q^{v,f}_R$. □

Throughout this paper, we may omit v as well as f if these are clear from the context.

To help the reader understand this definition, we present an example of write and read quorums below.

Example 2 Fig. 2 depicts a write quorum and a read quorum in the 6×6 dynamic grid quorum with $f = \langle 2, 4 \rangle$. In this figure, the points of nodes in the write quorum are denoted by vertical striped patterns. This write quorum consists of the following two sets: the set of all nodes on the fifth column (i.e., $v(Col(5))$), nodes chosen singly from the i-th column for each $i = 1, \ldots, 4$ (i.e., $\{v(1, 5), v(2, 5), v(3, 1), v(4, 4)\}$) where $v(1, 5)$ and $v(2, 5)$ must be chosen because $(5, 1), (5, 2) \in E_{low}$. On the other hand, nodes in the read quorum are denoted by gray shading. The read quorum consists of nodes chosen singly from each column (i.e., $\{v(1, 4), v(2, 1), v(3, 6), v(4, 1), v(5, 5), v(6, 2)\}$) where $v(4, 1)$ must be included because $(1, 4) \in E_{left}$.

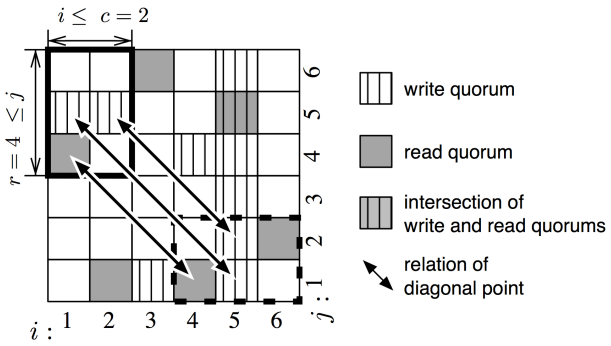

Fig. 2 Write and read quorums with $f = \langle c, r \rangle = \langle 2, 4 \rangle$

4 Power Consumption Model

We introduce a power consumption model that includes the following: *strategy*, *load*, *capacity* of nodes, *state-function*, and *power consumption*.

Strategy. Each write/read request is assigned to a write/read quorum by the probability distribution $P = \langle P_W, P_R \rangle$ over $\langle \mathcal{C}_W, \mathcal{C}_R \rangle$ satisfying the following properties:

- $0 \leq P_A(Q_A) \leq 1$ for any $Q_A \in \mathcal{C}_A$.
- $\sum_{Q_A \in \mathcal{C}_A} P_A(Q_A) = 1$.

We call this probability distribution the strategy.

Load. We define the load of a system, $ld = \langle ld_W, ld_R \rangle$, as the number of write/read requests per unit time. We

also define functions $load_W$ and $load_R$ which represent the loads of the write/read quorums (say, Q_W and Q_R) as follows.

$$load_A(Q_A) = ld_A \cdot P_A(Q_A). \tag{1}$$

In addition, we extend these functions to represent the load of a node (say, u) as follows.

$$- \ load_A(u) = \sum_{Q_A \in \mathcal{C}_A, Q_A \ni u} load_A(Q_A). \tag{2}$$

$$- \ load(u) = load_W(u) + load_R(u). \tag{3}$$

Capacity. We assume that each node $u \in U$ has a capacity $cap(u)$, which represents the upper bound of $load(u)$ defined by the following condition.

$$load(u) \leq cap(u). \tag{4}$$

We also introduce the notion of *overload* as follows. Node u is overloaded if $load(u) > cap(u)$. A system is overloaded if there is at least one overloaded node. A system is overloaded with the nodes up to the k-th column if the system is overloaded for any strategy such that the load of every node from the right of the k+1-th column is 0 (i.e., $load_W(v(i,j)) = 0$ for any $i > k$ and for any j).

State-function. We introduce the state-function On, a Boolean function over the set of nodes, which satisfies the following condition.

$$\text{If } load(u) \neq 0 \text{ then } On(u) = 1. \tag{5}$$

We model the state of the nodes by this function. That is, if $On(u)=1$ then node u is *active*, whereas if $On(u)=0$ then u is on *standby* and consumes no power. Note that $On(u)$ may be 1 even if $load(u) = 0$. This means that node u is active even though it is not executing any operations.

Power consumption. We assume that every node in an active state consumes the same power. Thus we define the power consumption of a system by the number of active nodes (i.e., $\sum_{u \in U} On(u)$) denoted by $|On|$.

5 Reconfiguration

To reduce the power consumption in an environment where the load varies, our system enables reconfiguration by exchanging a node with the corresponding diagonal point. This can be defined as the change of allocation mapping.

Definition 2 (Reconfiguration) For any $(i,j) \in E_{left}$, let $\xrightarrow{(i,j)}$ be a binary relation over the allocations such that if $v \xrightarrow{(i_1,j_1)} v'$ then for any i_2 and j_2

$$v(i_2,j_2) = \begin{cases} v'(j_2,i_2) & \text{if } (i_1{=}i_2 \wedge j_1{=}j_2) \vee (i_1{=}j_2 \wedge j_1{=}i_2) \\ v'(i_2,j_2) & \text{otherwise.} \end{cases}$$ □

In addition, for any $E' \subseteq E_{left}$, the notation $v \xrightarrow{E'} v'$ is used to represent that v' is obtained from v by successive exchanges of all the points in E' with their diagonals. In this case, v' is called the *reconfigured allocation* from v. Here we note that, obviously, for any v and v', if $v \xrightarrow{E'} v'$ then $v' \xrightarrow{E'} v$. In addition, v' can be obtained from v regardless of the order of exchanges. For the sake of convenience, as a special case, we introduce the expression $v \xrightarrow{\phi} v'$ to represent that v' can be obtained from v without any operation, i.e. $v = v'$. Note that any system obtained through reconfigurations is also a dynamic grid quorum.

We present an example of this reconfiguration below.

Example 3 Fig. 3 shows the capacity of all nodes in the 4×4 dynamic grid quorums $\mathcal{C}^v$ and $\mathcal{C}^{v'}$, where v' is obtained from v by reconfiguration with $E' = \{(1,2),(1,3),(1,4)\}$. (Thus, v is also obtained from v' by reconfiguration with the same set, E'.) Here we consider the case that $ld_W = 1$ and $ld_R = 3$. In this instance, as the upper two systems in this figure indicate, the minimum number of active nodes in system $\mathcal{C}^v$ and its reconfigured system $\mathcal{C}^{v'}$ are 9 and 7. On the other hand, where $ld_W = 3$ and $ld_R = 1$, as the lower two systems indicate, the minimum numbers of active nodes in system $\mathcal{C}^{v'}$ and its reconfigured system $\mathcal{C}^v$ are 10 and 7. This shows that our reconfiguration may remap nodes with heterogeneous capacity, thereby reducing the power consumption of the system when the ratio of write/read requests varies.

Although such reconfigurations are indeed useful for reducing the power, it may violate the consistency of the replicated data. Our reconfiguration, however, preserves the property of consistency called *one-copy serializability* (cf. [2]), which guarantees that any read operation returns the data installed by the last committed write operation. This can be guaranteed by the following proposition.

Proposition 1 Let v and v' be allocations which satisfy $v \xrightarrow{E'} v'$ for some $E' \subseteq E_{left}$. $\mathcal{C}^v$ and $\mathcal{C}^{v'}$ are dynamic grid quorums consisting of the same nodes. For

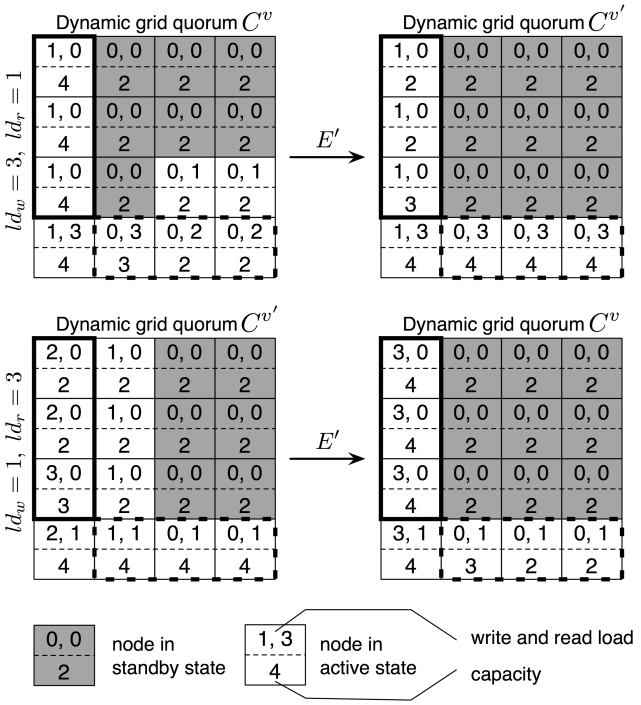

Fig. 3 Reconfigurations $\xrightarrow{E'}$ with $E' = \{(1,2),(1,3),(1,4)\}$

any $Q_W \in \mathcal{C}_W^v$, $Q'_W \in \mathcal{C}_W^{v'}$, and $Q'_R \in \mathcal{C}_R^{v'}$, the following two properties hold: $Q_W \cap Q'_R \neq \phi$ and $Q_W \cap Q'_W \neq \phi$.

Proof Due to space limitations, we show only the first property. From the definition of a dynamic grid quorum, for any $Q_W \in \mathcal{C}_W$ and $Q'_R \in \mathcal{C}'_R$, there exist i and j such that $v(Col(i)) \subseteq Q_W$ and $v'(i,j) \in Q'_R$. Here we consider the following three cases: (a) $(i,j) \in E_{low}$; (b) $(j,i) \in E_{left}$; (c) otherwise.

Case (a): By the definition of reconfiguration, if $(i,j) \in E'$ then $v'(i,j) = v(j,i)$, otherwise $v'(i,j) = v(i,j)$. By the definition of a dynamic grid quorum, $v(i,j), v(j,i) \in Q_W$, because Q_W has the corresponding diagonal node if Q_W has a node in the lower exchangeable area. Thus $Q_W \cap Q'_R \ni v'(i,j)$.

Case (b): By the definition of reconfiguration, if $(i,j) \in E'$ is exchanged then $v(i,j) = v'(j,i)$, otherwise $v(i,j) = v'(i,j)$. By the definition of a dynamic grid quorum, $v'(i,j), v'(j,i) \in Q'_R$, because Q'_R has the corresponding diagonal node if Q'_R has a node in the left-side exchangeable area. On the other hand, $v(i,j) \in Q_W$. Thus $Q_W \cap Q'_R \ni v(i,j)$.

Case (c): Because $v'(i,j)$ is never exchanged, $v(i,j) = v'(i,j)$. Thus $Q_W \cap Q'_R \ni v(i,j)$. □

We note that this proposition also guarantees that our reconfiguration can be done without any data migration to preserve the consistency of replicated data.

6 Dual Allocation

As defined in Section 3, in the dynamic grid quorum, the nodes are allocated to the points in an underlying grid, and the nodes are grouped into write and read quorums in a specific way according to this allocation. Then, as explained in Section 5, by changing the allocation we can gather high-capacity nodes to a busier area while preserving the one-copy serializability, that may reduce the number of nodes in the active mode. Here we note that this allocation is just a logical mapping, so our reconfiguration does not change the physical position of any node. From this consideration, we can separate allocations for write and read quorums, and then the reconfigurations for write and read quorums independently. This is called the *dual allocation*, which reduces power in a more effective way than with the single allocation.

We explain the idea behind this dual allocation by using a simple example below. (See also Fig. 4 for the graphical presentation.)

Example 4 Let us consider a system consisting of 4×4 grid, whose initial allocation is v_0. Now we assume that the system load for both write and read requests is 2, and the capacity of every node is 4. In this case, to minimize power consumption, the best strategy is to gather all the write requests to the write quorum $Q_W = \{u_1, u_2, u_3, u_4\}$ (i.e., the quorum consisting of the leftmost column) and all the read requests to the read quorum $Q_R = \{u_4, u_8, u_{12}, u_{16}\}$ (i.e., the quorum consisting of the lowest row). As the upper system in the figure indicates, with this strategy the three nodes $\{u_1, u_2, u_3\}$ are used for write operations, the three nodes $\{u_8, u_{12}, u_{16}\}$ are used for read operations, and the node u_4 is used for both. Thus, the total number of active nodes is 7. Here we consider the two distinct allocations, v_W and v_R, to reconfigure the write and read quorums, where $v_0 \xrightarrow{\phi} v_W$ and $v_0 \xrightarrow{E'} v_R$ with $E' = \{u_1, u_2, u_3\}$. The result is presented in the lower two systems in the figure. As this indicates, because only Q_R is changed to $\{u_1, u_2, u_3, u_4\}$ from $\{u_4, u_8, u_{12}, u_{16}\}$, after the reconfiguration, both Q_W and Q_R consist of the same nodes $\{u_1, u_2, u_3, u_4\}$. Thus, the number of active nodes becomes 4.

The formal definition of the dual allocation is as follows.

Definition 3 (Dynamic grid quorum with dual allocation) For any initial allocation v_0 and degree f, let $\langle v_W, v_R \rangle$ be a pair of (possibly different) allocations ranging over the same nodes and satisfying the following condition:

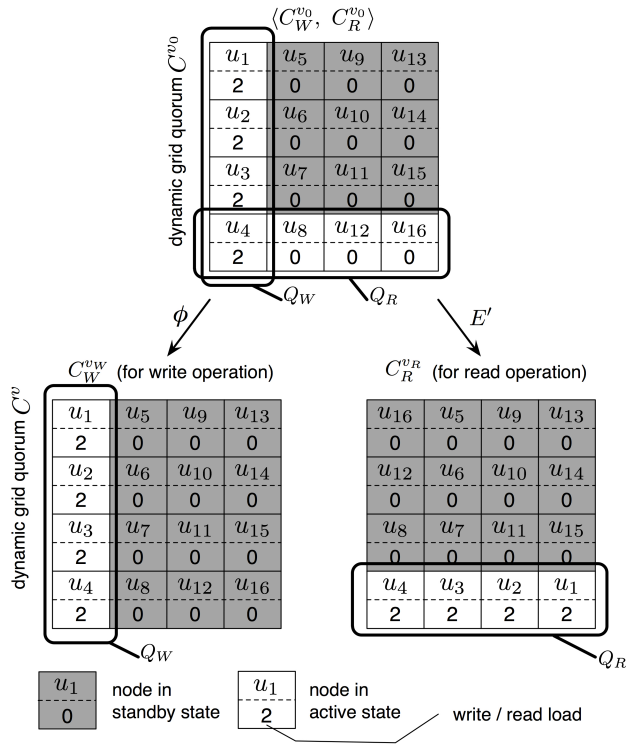

Fig. 4 Example of dual allocation (with $E' = \{(1,2),(1,3),(1,4)\}$)

There exist $E', E'' \subseteq E^f_{left}$ such that $v_0 \xrightarrow{E'} v_W$ and $v_0 \xrightarrow{E''} v_R$.

We call this pair a *dual allocation*. A dynamic grid quorum with dual allocation $\langle v_W, v_R \rangle$ is a pair $\langle \mathcal{C}_W^{v_W,f}, \mathcal{C}_R^{v_R,f} \rangle$ of collections of node groups obtained from $\langle \mathcal{C}_W^{v,f}, \mathcal{C}_R^{v,f} \rangle$ by replacing v with v_W (to determine $\mathcal{C}_W^{v_W,f}$) and v_R (to determine $\mathcal{C}_R^{v_R,f}$), in Definition 1. □

Clearly, for any dual allocation, v_W can be obtained from v_R by reconfigurations and vice versa. In addition, any dynamic grid quorum defined as Definition 1 is a special case of a dynamic grid quorum with dual allocation such that $v_W = v_R$.

We can naturally extend the definition of reconfiguration for a dynamic grid quorum (presented in Definition 1) to the one with dual allocation by separately considering the allocations for write and read operations. The related notions can be defined in a similar way. We say that $v' = \langle v'_W, v'_R \rangle$ is a reconfigured dual allocation obtained from $v = \langle v_W, v_R \rangle$, if there exist $E', E'' \subseteq E_{left}$ such that $v_W \xrightarrow{E'} v'_W$ and $v_R \xrightarrow{E''} v'_R$. The notation $v \xrightarrow{\langle E', E'' \rangle} v'$ is used to represent that v' is obtained from v by reconfiguration.

We note that, in our definition of dynamic grid quorum with dual allocation, the one-copy serializability still holds. To guarantee this consistency, it is sufficient to show that for any dual allocation $v = \langle v_W, v_R \rangle$ and for any write/read quorums Q_W, $Q'_W \in \mathcal{C}_W^{v_W}$ and for any $Q_R \in \mathcal{C}_R^{v_R}$, $Q_W \cap Q_R \neq \phi$ and $Q_W \cap Q'_W \neq \phi$ hold. The former one follows from Proposition 1 and the definition of dual allocation, while the latter one from the definition of dynamic grid quorum. Moreover, the reconfigurations for the dynamic grid quorum with dual allocation also preserve the one-copy serializability. This can be shown by Proposition 1.

In the next section, throughout we consider the dual allocation.

7 Optimization Algorithm

Our idea for power optimization is to skew the load towards a small number of quorums. This is realized by our optimization algorithm which is introduced below. To make our discussion simpler, we consider the following restrictions throughout this section.

(R1) For any $(i,j),(i,j') \in E^f_{left}$, $cap(v_0(i,j)) = cap(v_0(i,j'))$.

(R2) For any $(i,j),(i',j) \in E^f_{low}$, $cap(v_0(i,j)) = cap(v_0(i',j))$.

(R3) If there exists (i,j) with $On(v_0(i,j)) = 1$ then $On(v_0(i,j')) = 1$ for any $j' \leq j$.

(R4) If there exists (i,j) with $load_W(v_W(i,j)) \neq 0$ then $On(v_W(i',j)) = 1$ for any $i' \leq i$.

(R5) For any $(i,j),(i,j') \in E^f_{left}$, v_W, and E' such that $v_0 \xrightarrow{E'} v_W$, if $(i,j) \in E'$ then $(i,j') \in E'$.

(R6) For any $(i,j),(i',j) \in E^f_{left}$, v_R, and E' such that $v_0 \xrightarrow{E'} v_R$, if $(i,j) \in E'$ then $(i',j) \in E'$.

Intuitively, (R1) and (R2) respectively represent that in the left-side and in the lower exchangeable areas, the capacity of any node in the same column and in the same row is the same for the initial allocation. (R3) and (R4) respectively represent that if there is an active node, then all nodes below this node in the same column and all nodes in the same row on the left of this node are also active for the initial allocation and for any write allocation. (R5) and (R6) respectively represent that reconfigurations for any write and read allocations simultaneously exchange all the exchangeable nodes on the left in the i-th column, with the corresponding diagonal nodes.

Under these restrictions, we consider the following problem.

Problem 1 For a given initial allocation v_0, a degree f, a capacity cap, and a system load ld, find a pair of a strategy P and a reconfigured dual allocation v that minimizes the power consumption of the system.

Our optimization algorithm (called OA), includes the sub-algorithm (called SA), where algorithm SA is used to find an optimal strategy for a given allocation and system load. By means of SA, algorithm OA finds an optimal solution to Problem 1 among all the possible pairs each of which consists of a reconfigured allocation from the initial allocation and its optimal strategy. More precisely, our algorithm solves the problem by the following procedure.

First, for a given initial allocation v_0 and a given system load ld, algorithm OA considers all the possible reconfigured allocations obtained from v_0. Then, for each of these allocations, algorithm SA checks all the possible state-functions one-by-one to find one which minimizes power consumption and does not overload the system. From this optimal state-function, SA finds an optimal strategy which minimizes power consumption for this allocation by using linear programming. Here we would like to note the following point about the computational complexity. In general, the number of state-functions increases exponentially with the number of nodes in the system. Thus, to improve the problem of this search space explosion, algorithm SA checks the state-functions in ascending order of power consumption with a pruning technique. In addition, when using linear programming to find an optimal strategy, another problem of search space explosion emerges. That is, the number of quorums generally increases exponentially with the number of nodes, so this results in an explosion of the search space for this optimization. On the other hand, the power consumption of the system is determined only by the total load of each node. From this observation, the key idea in optimizing the strategy is to divide each quorum into its components and consider only the loads of each component. Then we can limit the search space to a subset created from a special type of load, called *component load*, from which we obtain our desired strategy.

Next, for a given set of pairs of reconfigured allocations from v_0 and their optimal strategy (which is obtained by the previous step), algorithm OA finds an optimal pair, namely a solution to Problem 1. In this step, since the number of allocations increases exponentially with the value of degree for columns c, the computational complexity of this step also increases exponentially with c. To address this problem, our algorithm also uses a pruning technique.

Here P^{opt} and v^{opt} are used respectively to denote the strategy and the allocation obtained by our algorithm OA. Then the following theorem holds.

Theorem 1 The pair of P^{opt} and v^{opt} is an optimal solution to Problem 1.

The outline of the proof of Theorem 1 is as follows. For the preparation, we first define the component load. Next, we prove Lemmas 1 and 2 which guarantee that an optimal strategy can be obtained from an optimal component load. Then we prove Lemma 3, which confirms that we can find an optimal component load, if any, for a given allocation. Furthermore, we prove Lemma 4, which confirms that we can find an optimal pair of a component load and a reconfigured allocation from the initial allocation. Finally, we prove Theorem 1 by using Lemmas 1–4.

Definition 4 (Component load) For a given dual allocation v, a degree f and system load ld, we define component load consisting of the following five functions:

- $c\text{-}load_{WN} : V_W \times \{(i,j) \mid (i,j) \notin E_{left} \wedge 1 \leq i < n\} \to \mathbb{R}^+$,
- $c\text{-}load_{WT} : V_W \times T \to \mathbb{R}^+$,
- $c\text{-}load_{WC} : V_W \times \{i \mid 1 \leq i \leq n\} \to \mathbb{R}^+$,
- $c\text{-}load_{RN} : V_R \times \{(i,j) \mid (i,j) \notin E_{left}\} \to \mathbb{R}^+$,
- $c\text{-}load_{RT} : V_R \times T \to \mathbb{R}^+$.

Here V_A is the set of write/read allocations; $\mathbb{R}^+$ is the set of non-negative real numbers; T is the set of tuples of natural numbers (each of which is denoted by $\mathbf{t} = \langle t_1, t_2, \ldots, t_c \rangle$) defined as follows:

- $t_i \in \{0\} \cup \{j \mid (i,j) \in E_{left}\}$;
- If $0 < t_i = j \leq c$ for some j, then $t_j = 0$ for any i;
- If $0 < t_i = j$ for some j, then $t_{i'} \neq j$ for any $i \neq i'$.

In addition, *c-load* previously defined satisfies conditions (6) to (13) given below.

$$- \sum_{i=1}^{n} c\text{-}load_{WC}(v_W, i) = ld_W. \quad (6)$$

$$- \sum_{j=1}^{m} c\text{-}load_{WN}(v_W, (i,j)) = \sum_{i'=i+1}^{n} c\text{-}load_{WC}(v_W, i') \quad \text{for any } i\,(c < i < n). \quad (7)$$

$$- \sum_{\mathbf{t} \in T} c\text{-}load_{WT}(v_W, \mathbf{t}) = \sum_{i=m+1}^{n} c\text{-}load_{WC}(v_W, i). \quad (8)$$

$$- \sum_{\mathbf{t} \in T, t_i = 0} c\text{-}load_{WT}(v_W, \mathbf{t}) = \sum_{(i,j) \notin E_{left}} c\text{-}load_{WN}(v_W, (i,j)) - \sum_{i'=i+1}^{r-1} c\text{-}load_{WC}(v_W, i') \quad \text{for any } i\,(1 \leq i \leq c). \quad (9)$$

$$- \sum_{\mathbf{t} \in T, t_i = j} c\text{-}load_{WT}(v_W, \mathbf{t}) \leq c\text{-}load_{WN}(v_W, (i,j)) \quad \text{for any } i\,(1 \leq i < n). \quad (10)$$

$$- \sum_{\mathbf{t}\in T} \textit{c-load}_{RT}(v_R, \mathbf{t}) = ld_R. \quad (11)$$

$$- \sum_{\mathbf{t}\in T,\ t_i=0} \textit{c-load}_{RT}(v_R, \mathbf{t}) = \sum_{(i,j)\notin E_{left}} \textit{c-load}_{RN}(v_R, (i,j)) \quad \text{for any } i\,(1 \le i \le n). \quad (12)$$

$$- \sum_{\mathbf{t}\in T,\ t_i=j} \textit{c-load}_{RT}(v_R, \mathbf{t}) \le \textit{c-load}_{RN}(v_R, (j,i)) \quad \text{for any } (i,j) \in E_{left}\ . \quad (13)$$

□

The intuitive meaning of this definition is as follows. First, to consider the write load of components, we divide each write quorum into the following three components: (a) the set of nodes on $Col(i)$, (b) the set of nodes on the left-side exchangeable points, and (c) the other nodes. The loads of components (a) and (c) are represented by $\textit{c-load}_{WC}$ and $\textit{c-load}_{WN}$, while the load of component (b) is represented by $\textit{c-load}_{WC}(v_W, i)$ (in the case of $i \le m$) or $\textit{c-load}_{WT}(v_W, \mathbf{t})$ (in the case of $i > m$). That is, in the case of $i > m$, t_i is included in $\mathbf{t}$ iff node $v_W(i,j)$ is included in the components with $t_i = j$.

On the other hand, to consider the read load of components, we divide each read quorum into the following two components: (d) the set of nodes on the left-side exchangeable points and (e) the other nodes. Functions $\textit{c-load}_{RT}$ and $\textit{c-load}_{RN}$ represent the total loads of (d) and (e). Vector $\mathbf{t}$ in $\textit{c-load}_{RT}$ indicates a component of (e), namely t_i is included in $\mathbf{t}$ iff node $v_R(i,j)$ is included in the components with $t_i = j$.

Instead of $load_W$ and $load_R$ (introduced by Eq. (2)), in terms of the five functions introduced by Definition 4, we define $\textit{c-load}_A : V \times U \to \mathbb{R}^+$ (where V is the set of dual allocations) that represents the total write/read load of each node, as follows.

$$- \textit{c-load}_W(v, v_W(i,j)) = \begin{cases} \textit{c-load}_{WC}(v_W, i) + \textit{c-load}_{WC}(v_W, j) + \sum_{\mathbf{t}\in T,\ t_i=j} \textit{c-load}_{WT}(v_W, \mathbf{t}) & \text{if } (i,j) \in E_{left} \\ \textit{c-load}_{WC}(v_W, i) + \textit{c-load}_{WN}(v_W, (i,j)) & \text{otherwise.} \end{cases} \quad (14)$$

$$- \textit{c-load}_R(v, v_R(i,j)) = \begin{cases} \sum_{\mathbf{t}\in T,\ t_i=j} \textit{c-load}_{RT}(v_R, \mathbf{t}) & \text{if } (i,j) \in E_{left} \\ \textit{c-load}_{RN}(v_R, (i,j)) & \text{otherwise.} \end{cases} \quad (15)$$

As in the case of function load, we also define *c-load* as follows.

$$\textit{c-load}(v, u) = \textit{c-load}_W(v, u) + \textit{c-load}_R(v, u).$$

For readability, throughout this paper we may omit the first parameter of $\textit{c-load}_A$ and *c-load*, if it is clear from the context.

In terms of the definition of *c-load*, instead of *load* we can also define the same condition of the capacity as Eq. (4) as well as the state-function as follows.

$$- \textit{c-load}(u) \le cap(u) \qquad \text{for any } u \in U. \quad (16)$$

$$- \text{If } \textit{c-load}(u) \ne 0 \text{ then } On(u) = 1 \quad \text{for any } u \in U. \quad (17)$$

We now introduce Lemmas 1 and 2. For readability, we consider the collection $\mathcal{C}_{W,Col(k)}$ $(k \le n)$ of write quorums and functions π_A and π'_A as follows.

$$- \mathcal{C}_{W,Col(k)} = \{Q_W \mid Q_W \in \mathcal{C}_W \wedge \forall j \le m\ (v_W(k,j) \in Q_W)\} \quad \text{for any } k \le n.$$

$$- \pi_A(\mathbf{t}) = \{Q_A \mid Q_A \in \mathcal{C}_A \wedge (t_i = j \ne 0 \text{ iff } v_A(i,j) \in Q_A \wedge (i,j) \in E_{left}) \wedge \forall k \le m (Q_A \notin \mathcal{C}_{W,Col(k)})\} \quad \text{for any } \mathbf{t} \in T.$$

$$- \pi'_A(Q_A) = \mathbf{t} \quad \text{s.t. } Q_A \in \pi_A(\mathbf{t}) \text{ for any } Q_A \in \mathcal{C}_A \text{ except for } Q_A \in \mathcal{C}_{W,Col(k)} \text{ for some } k \le m.$$

Intuitively, $\mathcal{C}_{W,Col(k)}$ means the collection of write quorums each of which includes all the nodes in the k-th column for the write allocation. Function $\pi_A(\mathbf{t})$ returns the set of quorums whose components are indicated by input $\mathbf{t}$, while π'_A is similar to the inverse of π_A.

Lemma 1 For any allocation v, any degree f, any system load ld, any capacity *cap*, and any component load *c-load* satisfying (16), we can obtain a strategy P such that it satisfies (4) and its power consumption is the same as *c-load*.

Proof To show this lemma, it is sufficient to show that $load_W(u) = \textit{c-load}_W(u)$ and $load_R(u) = \textit{c-load}_R(u)$ for any $u \in U$. To obtain these equations, we define P_W and P_R as (18) and (19), respectively.

– $P_W(Q_W) =$

$$\begin{cases} \dfrac{\textit{c-load}_{WC}(v_W,k)\cdot \textit{c-load}_{WT}(v_W,\pi'_W(Q_W))}{ld_W\cdot\sum_{k'=m+1}^{n}\textit{c-load}_{WC}(v_W,k')}\cdot g_1(Q_W) \\ \qquad\qquad\qquad\qquad \text{for some } k>m \\ \dfrac{\textit{c-load}_{WC}(v_W,k)}{ld_W}\cdot g_1(Q_W) \\ \qquad\qquad\qquad\qquad \text{otherwise} \end{cases}$$

where k satisfies $Q_W \in \mathcal{C}_{W,Col(k)}$. (18)

– $P_R(Q_R) = \dfrac{\textit{c-load}_{RT}(v_R,\pi'_R(Q_R))}{ld_R}\cdot g_1(Q_R)$. (19)

Here g_1 is defined as follows.

$$g_1(Q_A) = \prod_{(i,j)\in E'} \frac{g_2(i,j)}{\sum_{(i,j')\notin E_{left}} g_2(i,j')}$$

for $i \neq k$ if $A = W$,

where $E' = \{(i,j) \mid v_A(i,j) \in Q_A \wedge v_A(j,i) \notin Q_A\}$, and k satisfies $Q_W \in \mathcal{C}_{W,Col(k)}$. In addition, g_2 is defined as follows.

$$g_2(i,j) = \textit{c-load}_{AN}(v_A,(i,j)) - \sum_{\mathbf{t}\in T,\ t_j=i} \textit{c-load}_{AT}(v_A,\mathbf{t}).$$

Then by the definitions of *c-load* (i.e., Eqs. (6)–(10), we can derive the following equation from (18).

$$\sum_{Q_W\ni v_W(i,j)} P_W(Q_W)\cdot ld_W =$$

$$\begin{cases} \textit{c-load}_{WC}(v_W,i) + \textit{c-load}_{WC}(v_W,j) \\ \qquad + \sum_{\mathbf{t}\in T,\ t_i=j}\textit{c-load}_{WT}(v_W,\mathbf{t}) \\ \qquad\qquad\qquad \text{if } (i,j)\in E_{left} \\ \textit{c-load}_{WC}(v_W,i) + \textit{c-load}_{WN}(v_W,(i,j)) \\ \qquad\qquad\qquad \text{otherwise.} \end{cases} \quad (20)$$

Hence, from (14), which is the condition of $\textit{c-load}_W$, and (20), we can obtain $load_W(v_W(i,j)) = \textit{c-load}_W(v_W(i,j))$ for any (i,j).

Similarly, we can also derive $load_R(v_R(i,j)) = \textit{c-load}_R(v_R(i,j))$ from (11)–(13), (15), and (19). □

In addition, the following lemma, which is the converse of Lemma 1, is provable.

Lemma 2 For any allocation v, any degree f, any system load ld, any capacity cap, and any strategy P, we can obtain a component load whose power consumption is the same as P.

Proof To show this lemma, it is sufficient to show that $\textit{c-load}_W(u) = load_W(u)$ and $\textit{c-load}_R(u) = load_R(u)$ for any $u \in U$, where $load_W$ and $load_R$ are determined by P. To obtain these equations, we define $\textit{c-load}_{WN}$, $\textit{c-load}_{WT}$, $\textit{c-load}_{WC}$, $\textit{c-load}_{RN}$, and $\textit{c-load}_{RT}$ as (21), (22), (23), (24), and (25), respectively.

– $\textit{c-load}_{WN}(v_W,(i,j)) = \sum_{Q_W\ni v_W(i,j), Q_W\notin\mathcal{C}_{W,Col(i)}} load_W(Q_W)$. (21)

– $\textit{c-load}_{WT}(v_W,\mathbf{t}) = \sum_{Q_W\in\pi_W(\mathbf{t})} load_W(Q_W)$. (22)

– $\textit{c-load}_{WC}(v_W,i) = \sum_{Q_W\in\mathcal{C}_{W,Col(i)}} load_W(Q_W)$. (23)

– $\textit{c-load}_{RN}(v_R,(i,j)) = \sum_{Q_R\ni v_R(i,j)} load_R(Q_R)$. (24)

– $\textit{c-load}_{RT}(v_R,\mathbf{t}) = \sum_{Q_R\in\pi_R(\mathbf{t})} load_R(Q_R)$. (25)

The equation $\textit{c-load}_W(v_W(i,j)) = load_W(v_W(i,j))$ can be obtained by (22) and (23) for any $(i,j) \in E_{left}$. In addition, the same equation can be obtained by (21) and (23) for any $(i,j) \notin E_{left}$.

Similarly, we can also derive $\textit{c-load}_R(v_R(i,j)) = load_R(v_R(i,j))$ for any (i,j) from (24) and (25). □

Next, we introduce algorithm SA presented in Fig. 5. For any allocation, any degree, any system load, and any capacity, this algorithm finds a component load, if any, which minimizes power consumption among all the component loads satisfying (16) and (17).

The algorithm SA consists of three sub-algorithms named LP1, LP2, and LP3, which are based on linear programming. For given k, $i \leq n$, LP1 and LP2 respectively estimate the lower- and upper-bound of the number of active nodes in the i-th column for read allocations such that the system is not overloaded with the nodes up to the k-th column (i.e., $load_W(v_W(k',j)) = 0$ for all $k' > k$ and for all j). That is, the outputs of LP1 and LP2 are defined as follows.

– $\text{LP1}(i,k) = \min\{j \mid \sum_{j'\leq j} cap(v_R(i,j')) \geq ld_R + \min L(i,j)\}$,

– $\text{LP2}(i,k) = \min\{j \mid \sum_{j'\leq j} cap(v_R(i,j')) \geq ld_R + \max L(i,j)\}$,

where $L(i,j)$ is the set of all possible values of $\sum_{j'\leq j}\textit{c-load}_W(v_W(i,j'))$ such that $\textit{c-load}_W$ satisfies $\textit{c-load}_W(v_W(i',j'))=0$ for any $i' > k$ and for any j'.

Here each element of $L(i,j)$ means the total write load of nodes below the j-th row and in the i-th column. The values of $\min L(i,j)$ and $\max L(i,j)$ can be solved using linear programming whose constraints are represented by the following conditions.

- Eqs. (6)–(13).
- For any $i > k$ and for any j, $c\text{-}load_W(v_W(i,j)) = 0$.

On the other hand, for given k and state-function, LP3 returns a component load, if any, satisfying the following conditions.

- Eqs. (6)–(13) and (17).
- For any $i > k$ and for any j, $c\text{-}load_W(v_W(i,j)) = 0$.

By using these sub-algorithms, SA finds an optimal state-function among the set $S = \cup_{k=0}^{n} S_k$ of state-functions such that each $On \in S_k$ satisfies the following conditions.

(P1) $On(v_W(k',j)) = 1$ for any j and any $k' \leq k$, and $On(v_W(k+1,j)) = 0$ for some j.
(P2) For any (i,j), $On(v_R(i,j)) = 1$ if $j \leq \mathrm{LP1}(k,i)$.
(P3) $On(v_R(i,j)) = 0$ for any (i,j) satisfying the following conditions:
  (P3.1) $j > \mathrm{LP2}(k,i)$,
  (P3.2) $(i,j) \notin E_{low}$ or $On(v_R(j,i)) = 0$,
  (P3.3) $On(v_0(i',h)) = 0$ for any $h > j'$ with $v_0(i',j') = v_R(i,j)$,
  (P3.4) $i' > k$ with $v_W(i',j') = v_R(i,j)$.
(P4) $On(v_R(i,j)) = 1$ for any (i,j) and (i',j') satisfying the following six conditions:
  (P4.1) $i, i' > k$,
  (P4.2) $(i,j),(i',j') \in [r,m]$ or $(i,j),(i',j') \notin [r,m]$,
  (P4.3) $On(v_R(i',j')) = 1$,
  (P4.4) $(i',j') \notin E_{low}$ or $On(v_R(j',i')) = 0$,
  (P4.5) $On(v_0(h,l')) = 0$ for any $l' > l$ with $v_0(h,l) = v_R(i',j')$,
  (P4.6) $j \leq \min\{h \mid \sum_{h' \leq h} cap(v_R(i,h')) \geq \sum_{l<j'} cap(v_R(i',l))\}$.

To find an optimal state-function, the execution process of SA is as follows.

Step 1: Find α such that for any $k < \alpha$, LP3 has no solution for On^{all} and k, where On^{all} is the state function with $On^{all}(u) = 1$ for any $u \in U$.
Step 2: For each S_k with $k \geq \alpha$, find $On \in S_k$ such that $|On|$ is the minimum among all state-functions which do not cause system overload.
Step 3: If the power consumption of every state-function in S_{k+1} is more than the one of state-function On such that $|On|$ is the minimum for all the functions found in Step 2, then return On. (We call this output On^{opt}.)

```
1:input: U, v, E^f, cap, ld
2:On^opt ← U ;
3:k ← 0 ;
4:while LP3(α, On^opt) returns no strategy
5:   if k = n then
6:     exit "NO SOLUTION" ;
7:   k ← k + 1 ;
8:loop
9:   On ← {v_W(i,j) | i ≤ k} ;
10:  On ← On ∪ {v_R(i,j) | j ≤ LP1(i,k)} ;
11:  loop
12:    On' ← On ∪ {v_0(i,j) | ∃i'(v_0(i',j) ∈ On ∧ i < i')} ;
13:    On' ← On' ∪ {v_R(i,j) | v_R(j,i) ∈ On ∧ (j,i) ∈ E_left} ;
14:    if On = On' then
15:      break ;
16:    On ← On' ;
17:  if |On| ≥ |On^opt| then
18:    return LP3(n, On^opt) ;
19:  S ← {On} ;
20:  while S ≠ φ
21:    On ← arg min_{On'} {|On'| | On' ∈ S} ;
22:    if |On| ≥ |On^opt| then
23:      break ;
24:    if LP3(n, On) then
25:      On^opt ← On ;
26:      break ;
27:    for i = 1 to n
28:      On' ← On ∪ {v_R(i,j) | j = max{j' | v_R(i,j') ∈ On} + 1} ;
29:      loop
30:        On' ← On ∪ {v_0(i,j) | ∃i'(v_0(i',j) ∈ On ∧ i < i')} ;
31:        On' ← On' ∪ {v_R(i,j) | v_R(j,i) ∈ On ∧ (j,i) ∈ E_left} ;
32:        add nodes to On' for satisfying (P4) ;
33:        if On = On' then
34:          break ;
35:        On ← On' ;
36:      if |On| < |On^opt| and On satisfies (P4) then
37:        S ← S ∪ On ;
38:  if k = n then
39:    return LP3(n, On^opt) ;
40:  k ← k + 1 ;
```

Fig. 5 Algorithm SA

Clearly, when algorithm SA terminates, $|On^{opt}| \leq |On|$ for any On satisfying that $On \in S_\alpha \cup \cdots \cup S_n$.

Now we show that algorithm SA finds an optimal component load for a given system setting and its load.

Lemma 3 For any allocation v, any degree f, any system load ld, and any capacity cap, algorithm SA returns a component load, if any, which minimizes power consumption.

Proof To show this lemma, it is sufficient to show the following properties hold.

(C1) For any $k < \alpha$ and for any $On \in S_k$, On causes the system overload.
(C2) For any $On \notin S$, On causes the system overload or $|On| \geq |On^{opt}|$.

41:**input:** U, v_0, E^f, cap, ld
42:$C \leftarrow \phi$;
43:$On^{opt} \leftarrow U$;
44:$S \leftarrow \{\langle K_{0W}, K_{WR}\rangle \mid K_{0W}, K_{WR} \subseteq \{1, \ldots, c\}\}$;
45:**while** $S \neq \phi$
46: $s \leftarrow$ some element chosen from S ;
47: $S \leftarrow S - \{s\}$;
48: $v \leftarrow$ the allocation made by s from v_0 ;
49: $C' \leftarrow \mathrm{SA}(U, v, E^f, cap, ld)$;
50: $On \leftarrow$ the set of active nodes determined by C' ;
51: **if** $|On| < |On^{opt}|$ **then**
52: $C \leftarrow C'$;
53: $v^{opt} \leftarrow v$;
54: $On^{opt} \leftarrow On$;
55: **if** $|On^{opt}| < m^2$ **then**
56: $S \leftarrow S - R$;
57: # R is the set of allocations satisfying condition (P5).
58:Transform C into strategy P^{opt} by using Eqs. (18) and (19) ;
59:**return** v^{opt}, P^{opt} ;

Fig. 6 Algorithm OA

For (C1): We assume that there exists $On \in S_k$ for some $k < \alpha$ such that LP3(On, k) (i.e., the output of LP3 for the inputs On and k) is component load *c-load*. Then *c-load* clearly satisfies (17) with On^{all}. Thus, Step 1 of SA finishes with this k. This contradicts $k < \alpha$.

For (C2): It is sufficient to show that for any state function $On \notin S$, if LP3(On, n) has a solution (where n is the number of columns in the system) then there exists a state function $On' \in S$ such that LP3(On', n) has a solution with $|On'| \leq |On|$. For space limitations, we show only the case that On does not satisfy (P4). That is, we assume that there exist (i, j) and (i', j') satisfying (P4.1)–(P4.6) with $On(v_R(i, j)) = 0$. By (P4.1) and (P4.2), we obtain $\sum_{j=1}^{m} load_W(v_R(i, j)) = \sum_{j=1}^{m} load_W(v_R(i', j))$. In addition, by (P4.6), we obtain $\sum_{h=1}^{j-1} cap(v_R(i, h)) < \sum_{h=1}^{j'-1} cap(v_R(i', h))$. Thus, by the assumption $On(v_R(i, j)) = 0$,

$$\sum_{h=1}^{j'-1} cap(v_R(i', h)) \geq \sum_{h=1}^{m} c\text{-}load_W(v_R(i', h)) + ld_R. \qquad (*)$$

Let On' be a state-function such that $On'(v_R(i', j')) = 0$ and for any node u except for $v_R(i', j')$, $On'(u) = On(u)$. For this state-function, from (P4.4), (P4.5), and inequation ($*$), we can conclude that LP3(On', k) has a solution. Clearly, since $On'(v_R(i', j')) = 0$, (P4.3) is false, thus On' satisfies (P4). Therefore, $On' \in S$. Moreover, $|On'| < |On|$, because $On'(v_R(i', j')) = 0$ while $On(v_R(i', j')) = 1$ and $On'(u) = On(u)$ with $u \neq v_R(i', j')$. □

Next, we introduce algorithm OA, which is presented in Fig. 6. For any given initial allocation, any degree, any system load, and any capacity, this algorithm first finds an optimal pair of a component load and a reconfigured allocation (up to line 58) then returns a solution to Problem 1 by transforming this component load into an optimal strategy with Eqs. (18) and (19). Clearly, we can find an optimal pair of an allocation and a component load by considering all the possible reconfigured allocations and then checking them one-by-one using algorithm SA. However, as mentioned before, to reduce the computational complexity of this procedure, we use the pruning technique as follows.

For any initial allocation v_0 and for any allocation $\langle v_W, v_R\rangle$, let $K(v_0, v_W)$ and $K(v_W, v_R)$ be sets of indexes of columns, satisfying the following conditions.

– $v_0 \xrightarrow{E'} v_W$ s.t. $E' = \bigcup_{k \in K(v_0, v_W)} (Col(k) \cap E_{left})$.

– $v_W \xrightarrow{E''} v_R$ s.t. $E'' = \bigcup_{k \in K(v_W, v_R)} (Col(k) \cap E_{left})$.

Note that, by (R5) and (R6), it is guaranteed that there exist such $K(v_0, v_W)$ and $K(v_W, v_R)$ for any v_0 and $\langle v_W, v_R\rangle$.

In the procedure to find an optimal pair of a component load and a reconfigured allocation from v_0, at lines 55 and 56 in Fig. 6, if it is known that the number of active nodes of this solution is the same or smaller than m^2 (where m is the number of rows in the system), algorithm OA omits the allocations, whose set is denoted by R, such that each $v \in R$ satisfies the following condition.

(P5) There exists $(i, j) \in E_{left}$ such that (i, j) and $v = \langle v_W, v_R\rangle$ satisfy both (P5.1) and (P5.2) below.
  (P5.1) $cap(v_W(i, j)) < cap(v_W(j, i))$, $i \notin K(v_0, v_W)$, and $i \in K(v_W, v_R)$.
  (P5.2) At least one of the following conditions (P5.2-1) and (P5.2-2) is satisfied.
    (P5.2-1) There exists i' such that $i < i'$ and $i' \in K(v_0, v_W)$.
    (P5.2-2) The following inequation holds: $\sum_{i'=1}^{c} \max\{cap(v_W(i', j)), cap(v_W(j, i'))\} < ld_R$.

Note that for any system in which the number of columns is the same or smaller than the one of rows (i.e., $n \leq m$), the number of active nodes of an optimal solution is always the same or smaller than m^2.

Even though there is such a reduction of search space, algorithm OA finds an optimal solution. That is, the following lemma holds.

Lemma 4 For any initial allocation v_0, any degree f, any system load ld, and any capacity cap, we can obtain

a pair of a component load and a reconfigured allocation from v_0, if any, which minimizes power consumption.

Proof To show this lemma, it is sufficient to show that for any allocation $v \in R$, if there is a component load *c-load* such that the power consumption determined by *c-load* is the same or smaller than m^2 and the pair of v and *c-load* is optimal for the given f, ld, and cap, then there exists an allocation $v' \notin R$ such that the pair of v' and *c-load* is also optimal.

For space limitations, we show only the case that v satisfies (P5.2-1). That is, there exist $(i,j) \in E_{left}$ and i' such that (i,j), i', and v satisfy both (P5.1) and (P5.2-1). Here we consider the following two cases: (a) $c\text{-}load_{WC}(v_W, i'') = 0$ for any $i'' \geq i'$, and (b) otherwise.

For (a): We define $v' = \langle v'_W, v'_R \rangle$ as $v \xrightarrow{\langle E', \phi \rangle} v'$ where $E' = E_{left} \cup Col(i')$. Then $v' \notin R$. By the assumption, as the power consumption of *c-load* is the same or smaller than m^2, we obtain $c\text{-}load_{WT}(v_W, \mathbf{t}) = 0$ for any $\mathbf{t} \in T$. Since $c\text{-}load_{WC}(v_W, i'') = 0$ for any $i'' \geq i'$, $c\text{-}load_{WC}(v_W, i') = 0$. Hence, $c\text{-}load_W(v, v_W(i', j')) = c\text{-}load_W(v', v_W(i', j')) = 0$ for any j' with $(i', j') \in E_{left}$. Thus, for any $u \in U$, the load of u determined by *c-load* is the same for both v and v' for the given f, ld, cap. Therefore, the pair of v' and *c-load* is also optimal.

For (b): We define $v' = \langle v'_W, v'_R \rangle$ as $v \xrightarrow{\langle E', \phi \rangle} v'$ where $E' = E_{left} \cup Col(i)$. Then the following equation holds.

$$c\text{-}load(v', v_0(h,l)) = \begin{cases} c\text{-}load(v, v_0(h,l)) - c\text{-}load_W(v, v_0(h,l)) & \\ \qquad \text{for } h = i \wedge (h,l) \in E_{left} & \\ c\text{-}load_W(v, v_0(l,h)) & \\ \qquad \text{for } l = i \wedge (h,l) \in E_{low} & \\ c\text{-}load(v, v_0(h,l)) & \\ \qquad \text{otherwise.} & \end{cases}$$

Here, by (P5.1), (R1), and (R2) we obtain $cap(v_0(i,j')) \leq cap(v_0(j',i))$ for any j' with $(i,j') \in E_{left}$. Thus, as $c\text{-}load_W(v, v_0(i,j')) \leq cap(v_0(i,j'))$ for any j' with $(i,j') \in E_{left}$, $c\text{-}load_W(v', v_0(j',i)) \leq cap(v_0(i,j'))$. Therefore, the pair of *c-load* and v' does not cause the system overload for the given f, ld, cap, and the allocation. Now we let On be the state-function determined by *c-load* and v. Here the load of node $v_0(j',i)$ with $(j',i) \in E_{low}$ may increase for *c-load* and v', compared with the pair of *c-load* and v. However, $On(v_0(j',i)) = 1$ for any j' with $(j',i) \in E_{low}$, because there exists $i'' \geq i' \geq i$ with $c\text{-}load_{WC}(v_W, i'') \neq 0$ and $i' \in K(v_0, v_W)$. Thus, $|On'| = |On|$, where On' is the state-function determined by *c-load* and v'. Therefore, the pair of v' and *c-load* is also optimal. □

Finally, we prove the main theorem below.

Proof (Proof of Theorem1) Let $\langle c\text{-}load^{opt}, v^{opt} \rangle$ be the pair of the component load and the reconfigured allocation obtained by algorithm OA. Then, by Lemma 4, this pair is an optimal one in the set of all the pairs of component loads and allocations. Thus, by Lemma 1, we can obtain a strategy P^{opt} whose power consumption is the same as $c\text{-}load^{opt}$. Moreover, P^{opt} is an optimal strategy for v^{opt}, because if there exists another strategy whose power consumption is smaller than that of P^{opt}, then by Lemma 2, a better solution also exists in the set of component load for v^{opt}. This contradicts Lemma 3. □

In closing this section, we present some remarks on the restrictions considered in this section, the computational complexity of our algorithm, and implementation of our proposed system.

Remark 1 (Restrictions on the capacity and the state of nodes) For simplicity in our discussion, we introduce in this section some restrictions on the capacity and the state of the nodes. However, if we disregard one of the restrictions (R1)–(R6), there may be a better solution than that obtained by our algorithm.

Such an example is as follows. Let On^{opt} and *c-load* be the state-function and component load obtained by algorithm OA for a given system and its workload. Then let k be $\max\{i \mid c\text{-}load_{WC}(i) \neq 0\}$. Now we consider a state-function On satisfying the following conditions.

- If $On^{opt}(u) = 0$ then $On(u) = 0$ for any $u \in U$;
- For any $i \notin K(v_W, v_R)$. $\sum_{j \in \{j' \mid On(v_R(i,j'))=1\}} cap(v(i,j)) \geq ld_R + \mathrm{LP2}(i,k)$.
- For any $(i,j) \in E_{left}$, if $On(v_R(j,i)) = 0$ then $\sum_{\mathbf{t} \in T, t_i = j} c\text{-}load_{RT}(t_i) = 0$.

In this case, the nodes, each of which u satisfies $On^{opt}(u) = 1$ and $On(u) = 0$ can be reduced if we disregard assumption (R3).

More concretely, (see Fig. 7 for the graphical presentation) let $n = 4$, $m = 4$, $f = \langle 0, 4 \rangle$, $ld = \langle 0, 3 \rangle$, and $cap(v_0(1,j)) = 2$ for any $1 \leq j \leq 4$, and $cap(v_0(i,j)) = 4$, otherwise. By (R3), the minimum set of active nodes is $\{v_0(i,j) \mid 1 \leq i \leq 2 \wedge 1 \leq j \leq 4\}$, where the total number of nodes is 8. Without (R3), however, nodes $\{v_0(i,j) \mid i = 2 \wedge 1 \leq j \leq 4\}$ are sufficient, giving a total number of 4 nodes.

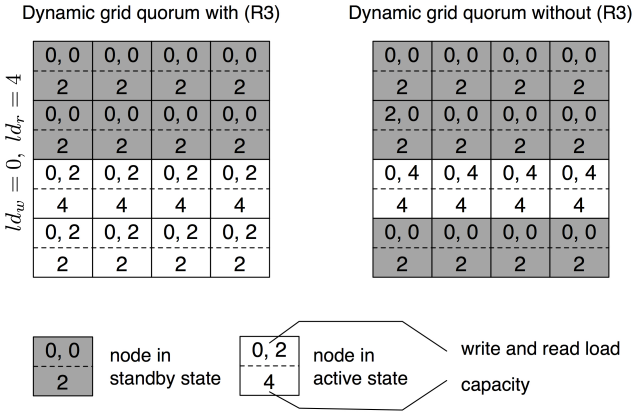

Fig. 7 Active nodes of dynamic grid quorum with/without (R3)

Remark 2 (Computational complexity) As explained before, to solve Problem 1 there are three factors causing an exponential increase in the computational complexity of the calculation, namely, the number of constraints used by the linear programming in SA, the number of state-functions treated by SA, and the number of reconfigured allocations treated by OA. In terms of pruning techniques as well as the notional component load, the computational complexity of our proposed algorithm is decreased. Indeed, as far as our experiments are concerned, we can find optimal solutions in a feasible time for all cases of the simulations in the next section. However, these three factors may still cause a search space explosion. The maximum values of these factors, say, ν_1, ν_2, and ν_3 can be represented with the O-notation as follows:

- $\nu_1 = O((m-r)^c)$,
- $\nu_2 = O((c-\pi)^r + c \cdot n \cdot m)$,
- $\nu_3 = O(2^c)$.

Here, n, m, c, and r are the numbers of columns and rows, and the degrees for columns and rows, respectively; $\pi = \min\{j \mid \sum_{0 \le j' \le j} cap(v_R(i,j')) \ge ld_R \wedge 1 \le i \le n\}$. In terms of these values, for the worst case, the required calculation to solve Problem 1 is the $c \cdot \nu_2 \cdot \nu_3$ iterations of linear programming using ν_1 constraints. A possible way to avoid such exponential growth is to restrict c to a small number. On the other hand, $(c-\pi)^r$ in the second equation is feasible in most cases, because large ld keeps $c-\pi$ small, and small ld makes the probability of breaking into short loops high. Moreover, $c \cdot n \cdot m$ in the second equation is also feasible, because it is shortened by pruning by (P4) of SA in a practical sense. This is a simple way to avoid the search space explosion, but an improvement to our algorithm is still needed. This will be investigated in one of our future studies.

Remark 3 (Implementation) Finally, we make some remarks related to the implementation of our proposed system. For implementation, first, the information of the current system load is needed to calculate optimal storage. It can be obtained by observing nodes in at most one column in read allocation and at most one row in write allocation. That is, it is sufficient to observe $v_0\,(Col(i))$ and $v_0\,(Row(i))$ for any i, where v_0 is the initial allocation.

Next, all the clients must share the allocation and strategy, however, from Proposition 1, it is derived that the consistency of the data are guaranteed even if some clients are out of synchronization. Thus, although it may cause temporarily high power consumption, it is not necessary to enforce strict synchronization among clients. Therefore, the overhead of updating the allocation and strategy will be small.

Finally, we point out that some methods are usable for managing the power of each node. For example, by restriction (R3), when node $v_0(i,j)$ must change state to active, $v_0(i,j-1)$ also changes state to active or is already active. Thus, it is sufficient for each node to manage the power of the above node. Therefore, clients need to manage the power of no more than m nodes.

8 Simulation Results

In the evaluation of this section, we consider the following four systems for a 10×10 grid: dynamic grid quorum with dual allocation (DDG), dynamic grid quorum with no dual allocation (i.e., single allocation) (DG), and fixed allocation of nodes optimized for write requests (WG) and read requests (RG). First we evaluate the impact of the write/read ratio for each case where the system load is low, medium or high. In contrast to the above settings, we also evaluate the impact of the system load, where the write/read ratio is fixed but the total number of requests varies.

Parameters and Settings. Each system in the evaluation consists of 70 nodes with capacity a (for some constant a) and 30 nodes with capacity $2a$. In the WG and RG, the nodes with $2a$ are allocated in Col_i and Row_i for $1 \le i \le 3$, respectively. The DG is a dynamic grid quorum with $f = \langle 3,1 \rangle$, while the DDG is a dynamic grid quorum with dual allocation with $f = \langle 3,3 \rangle$. The reason we consider DG and DDG with different degrees is that DG with $f = \langle 3,1 \rangle$ is better than the case of $f = \langle 3,3 \rangle$ with respect to the power consumption. To reduce the computational complexity, we consider the following restrictions (introduced in Section 7). The static systems WG and RG satisfy (R3) and (R4) while the dynamic systems DG and DDG satisfy (R1)–(R6). However, for DG, instead of (R3) we consider the restriction such that for any $i < i'$ and for

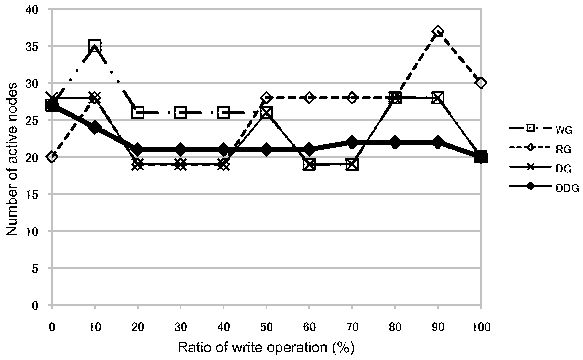

Fig. 8 Number of active nodes ($ld_W + ld_R = 2.5a$)

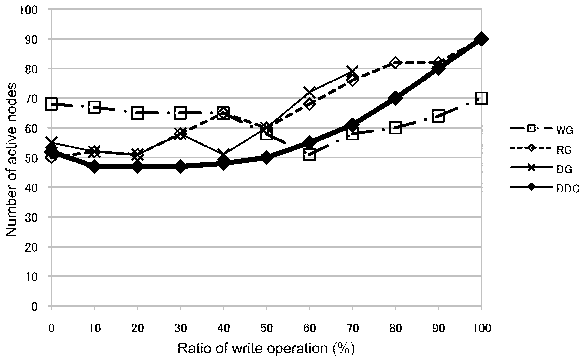

Fig. 10 Number of active nodes ($ld_W + ld_R = 7.5a$)

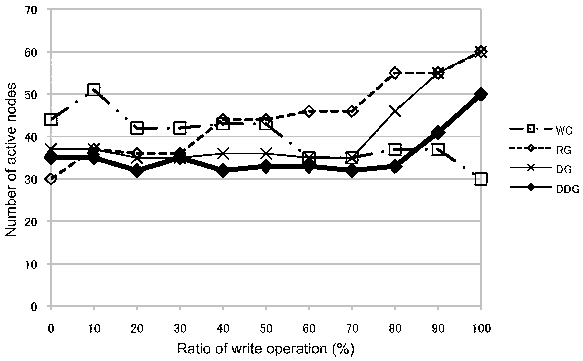

Fig. 9 Number of active nodes ($ld_W + ld_R = 5a$)

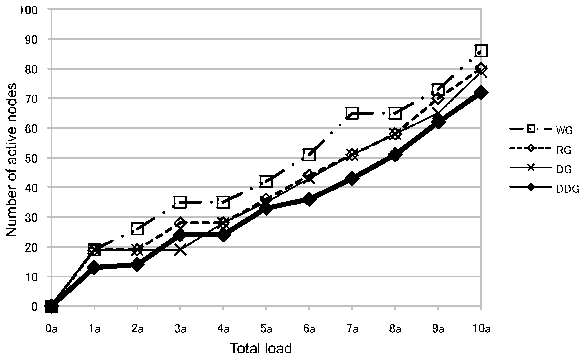

Fig. 11 Number of active nodes ($ld_W : ld_R = 1:3$)

any j if $On(v_R(i', j)) = 1$ then $On(v_R(i', j)) = 1$. This restriction reduces more power than in the case of (R3). The initial allocation of DG and DDG is the same as RG. For the impact analysis of the write/read ratio, we use $2.5a$, $5a$, and $7.5a$ as the values for low, medium, and high loads in the system. This setting arises from the fact that the write and read capacity for a grid of the same size, comprising homogeneous nodes with capacity a, is $10a$ and $6.5a$, respectively. For the impact analysis of the system load, we consider the write/read requests to be 1/3, which is also used as the default parameter in [16].

Impact of write/read ratio in the case of low load. Fig. 8 shows that the power consumption of the DDG is more stable over the range of the ratio of write operations compared with the others. In addition, compared with the WG, RG, and DG, DDG saves on average 13.6%, 14.8%, and 4.3% respectively, for all ratios. The results of this simulation show that dynamic grid quorum with dual allocation reduces power consumption when compared with all the other systems, WG, RG and DG.

Impact of write/read ratio in the case of medium load. Fig. 9 shows that the DDG consumes more power than the RG for the case of 0% of the write ratio and more power than the WG above 90% of the write ratio. However, except for these cases, the power consumption of the DDG is the least for all ratios. Considering more detailed results, the DDG saves on average 10.9%, 20.0%, and 12.5% compared with WG, RG, and DG. The results of this simulation show that our system is also efficient for a wide range of write ratios. In addition, in this simulation, the DDG shows better behaviour with a high write ratio than DG.

Impact of write/read ratio in the case of high load. Fig. 10 shows that the power consumption of the DDG is the least for the range 10% to 50% of the write ratio. The DDG consumes more power than the RG for the case of 0% of the write ratio and more power than the WG above 60% of the write ratio. However, a notable fact is that above 60% of the write ratio, DDG is still effective compared with RG and DG, while DG cannot handle a write ratio above 70%. Compared with the WG and RG, DDG saves on average 6.4% and 11.9%. Moreover, even if the write ratio is restricted in the range 0% to 70%, DDG saves 14.9% compared with DG.

Impact of load growth. Fig. 11 shows that the power consumption of the DDG is the least for almost all of the range for other system loads. Compared with the WG, RG, and DG our system saves on average 25.2%,

14.1%, and 10.6%. The results of this simulation show that our system is effective for any system load.

9 Conclusions and Future Work

In this paper, we presented a power-aware quorum system called the dynamic grid quorum. To reduce power consumption, the basic principle behind our system was to skew the workload towards a small number of quorums, which was realized by the following techniques: reconfiguration by exchanging nodes without data migration, dual allocation for write and read quorums, and an algorithm to find an optimal pair for reconfiguration and its strategy for a given system setting and workload. We also evaluated our techniques by comparing the proposed system using three alternatives, namely, static write- and read-optimized configurations, and dynamic reconfiguration with a single allocation. The result showed that our proposed system's power savings were, on average, between 10.6% and 25.2% with 1/3 write operation ratio.

There are several possible directions in which this work can be developed. From a theoretical viewpoint, as mentioned in Section 7, we are interested in improving our algorithm to avoid the situation that results in a search space explosion. In addition, we are interested in some extensions of our techniques, especially in modeling the heterogeneity of power consumption and the time required for the state transition of nodes, that were not considered in this paper. Furthermore, on a practical level, we plan to evaluate the effectiveness of our proposed techniques, which could be investigated through a prototype implementation.

Acknowledgements This study was supported in part by the Global COE Program on "Cybernics: fusion of human, machine, and information systems." This study was also partially supported through the SCOPE program by the Ministry of Internal Affairs and Communications of Japan.

References

1. Agrawal D, Abbadi AE (1991) An efficient and fault-tolerant solution for distributed mutual exclusion. ACM Trans on Computer Systems, 9(1):1–20
2. Bernstein PA, Hadzilacos V, Goodman N (1987) Concurrency Control and Recovery in Database Systems. Addison Wesley
3. Cheung S, Ammar M, Ahamad M (1992) The grid protocol: A high performance scheme for maintaining replicated data. IEEE Trans on Knowledge and Data Engineering, 4(6):582–592
4. Colarelli D, Grunwald D (2002) Massive arrays of idle disks for storage archives. Proc of the ACM/IEEE Conf on Supercomputing, pp 1–11
5. Frain I, Basmadjian R, Bahsoun JP, M'zoughi A (2006) How to improve the scalability of read/write operations with dynamic reconfiguration of a tree-structured coterie. Proc of the Int Conf on Parallel Processing Workshops, pp 123–134
6. Fu A (1997) Delay-optimal quorum consensus for distributed systems. IEEE Trans on Parallel and Distributed Systems, 8(1):59–69
7. Gifford D (1979) Weighted voting for replicated data. Proc of the ACM Symposium on Operating Systems Principles, pp 150–162
8. Harnik D, Naor D, Segall I (2009) Low power mode in cloud storage systems. Proc of IEEE Int Parallel and Distributed Processing Symposium, pp 1–8
9. Ibaraki T, Kameda T (1993) A theory of coteries: Mutual exclusion in distributed systems. IEEE Trans on Parallel and Distributed Systems, 4(7):779–794
10. Ishikawa M, Hasebe K, Sugiki A, Kato K (2009) Dynamic Grid Quorum: A Novel Approach for Minimizing Power Consumption without Data Migration in Grid Quorums. IEEE Int Conf on Service-Oriented Computing and Applications (SOCA'09), pp 142–149
11. Li D, Wang J (2004) EERAID: energy efficient redundant and inexpensive disk array. Proc of the ACM SIGOPS European Workshop, 6 pages
12. Lin X (2004) Delay optimization in quorum consensus. Algorithmica, 38(2):397–413
13. Maekawa M (1985) A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. ACM Trans on Computer Systems, 3(2):145–159
14. Peleg D, Wool A (1997) Crumbling walls: A class of practical and efficient quorum systems. Distributed Computing, 10(2):87–97
15. Pinheiro E, Bianchini R (2004) Energy conservation techniques for disk array-based servers. Proc of the Int Conf on Supercomputing, pp 68–78
16. Pinheiro E, Bianchini R, Dubnicki C (2006) Exploiting redundancy to conserve energy in storage systems. Proc of the joint Int Conf on Measurement and Modeling of Computer Systems, pp 15–26
17. Tsuchiya T, Yamaguchi M, Kikuno T (1999) Minimizing the maximum delay for reaching consensus in quorum-basedmutual exclusion schemes. IEEE Trans on Parallel and Distributed Systems, 10(4):337–345
18. Wang J, Zhu H, Li D (2008) eRAID: Conserving energy in conventional disk-based raid system. IEEE Trans on Computers, 57(3):359–374
19. Weddle C, Oldham M, Qian J, Wang AI, Reiher P, Kuenning G (2007) PARAID: A gear-shifting power-aware RAID. Proc of the USENIX Conf on File and Storage Technologies, pp 245–260
20. Yao X, Wang J (2006) RIMAC: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. Proc of the ACM SIGOPS/EuroSys European Conf on Computer Systems, pp 249–262
21. Zhu Q, Chen Z, Tan L, Zhou Y, Keeton K, Wilkes J (2005) Hibernator: helping disk arrays sleep through the winter. ACM SIGOPS Operating Systems Review, 39(5):177–190