Selective Replicated Declustering for Arbitrary Queries

K. Yasin Oktay, Ata Turk, and Cevdet Aykanat

Bilkent University, Department of Computer Engineering,
06800, Ankara, Turkey
koktay@ug.bilkent.edu.tr, {atat,aykanat}@cs.bilkent.edu.tr

Abstract. Data declustering is used to minimize query response times in data intensive applications. In this technique, query retrieval process is parallelized by distributing the data among several disks and it is useful in applications such as geographic information systems that access huge amounts of data. Declustering with replication is an extension of declustering with possible data replicas in the system. Many replicated declustering schemes have been proposed. Most of these schemes generate two or more copies of all data items. However, some applications have very large data sizes and even having two copies of all data items may not be feasible. In such systems selective replication is a necessity. Furthermore, existing replication schemes are not designed to utilize query distribution information if such information is available. In this study we propose a replicated declustering scheme that decides both on the data items to be replicated and the assignment of all data items to disks when there is limited replication capacity. We make use of available query information in order to decide replication and partitioning of the data and try to optimize aggregate parallel response time. We propose and implement a Fiduccia-Mattheyses-like iterative improvement algorithm to obtain a two-way replicated declustering and use this algorithm in a recursive framework to generate a multi-way replicated declustering. Experiments conducted with arbitrary queries on real datasets show that, especially for low replication constraints, the proposed scheme yields better performance results compared to existing replicated declustering schemes.

1 Introduction

Data declustering is one of the key techniques used in management of applications with humongous-scale data processing requirements. In this technique, query retrieval process is parallelized by distributing the data among several disks. The most crucial part of exploiting I/O parallelism is to develop distribution techniques that enable parallel access of the data. The distribution has to respect disk capacity constraints while trying to locate data items that are more likely to be retrieved together into separate disks.

There are many declustering schemes proposed for I/O optimization (See [1] and the citations contained within), especially for range queries. These schemes generally try to scatter neighboring data items into separate disks.

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 375–386, 2009.

There are also a few studies that propose to exploit query distribution information [2], [3], [4], if such information is available. For equal-sized data items, the total response time for a given query set can be minimized by evenly distributing the data items requested by each query across the disks as much as possible, while taking query frequencies into consideration. In [3], the declustering problem with a given query distribution is modeled as a max-cut partitioning of a weighted similarity graph. Here, data items are represented as vertices and an edge between two vertices indicate that corresponding data items appear in at least one query. The edge weights represent the likelihood that the two data items represented by the vertices of the edge will be requested together by queries. Hence, maximizing the edge cut in a partitioning of this similarity graph relates to maximizing the chance of assigning data items that will probably appear in the same query to separate disks. In [2] and [4], the deficiencies of the weighted similarity graph model are addressed and a correct hypergraph model which encodes the total I/O cost correctly is proposed. In this representation, vertices represent data items and hyperedges/nets represent queries, where each net representing a query connects the subset of vertices that corresponds to the data items requested by that query. The vertex weights represent the data item sizes and net weights represent the query frequencies. Recently, hypergraph models have also been applied for clustering purposes in data mining ([5]) and road network systems ([6], [7]).

In addition to declustering, replication of data items to achieve higher I/O parallelism has started to gain attention. There are many replicated declustering schemes proposed for optimizing range queries (See [8] and the citations contained within). Recently, there are a few studies that address this problem for arbitrary queries as well [9], [10], [11]. In [9], a Random Duplicate Assignment (RDA) scheme is proposed. RDA stores a data item on two disks chosen randomly from the set of disks and it is shown that the retrieval cost of random allocation is at most one more than the optimal with high probability. In [10], Orthogonal Assignment (OA) is proposed. OA is a two-copy replication scheme for arbitrary queries and if the two disks that a data item is stored at are considered as a pair, each pair appears only once in the disk allocation of OA. In [11], Design Theoretic Assignment (DTA) is proposed. DTA uses the blocks of an $(K, c, 1)$ design for c-copy replicated declustering using K disks. A block and its rotations can be used to determine the disks on which the data items are stored.

Unfortunately, none of the above replication schemes can utilize query distribution information if such information is available. However, with the increasing and extensive usage in GIS and spatial database systems, such information is becoming more and more available, and it is desirable for a replication scheme to be able to utilize this kind of information. Furthermore, replication has its own difficulties, mainly in the form of consistency considerations, that arise in update and deletion operations. Response times for write operations slow down when there is replication. Finally, with replication comes higher storage costs and there are applications with very large data sizes where even two-copy replication

is not feasible. Thus, if possible, unnecessary replication has to be avoided and techniques that enable replication under given size constraints must be studied.

In this study, we present a selective replicated declustering scheme that makes use of available query information and optimizes aggregate parallel response time within a given constraint on the replication amount due to disk size limitations. In the proposed scheme, there is no restriction on the replication counts of individual data items. That is, some data items may be replicated more than once while some other data items may not even be replicated at all. We propose an iterative-improvement-based replication algorithm that uses similar data structures with the Fiduccia-Mattheyses (FM) Heuristic [12] and recursively bipartition and replicate the data. FM is a linear time iterative improvement heuristic which was initially proposed and used for clustering purposes in bipartitioning hypergraphs that represent VLSI circuits. The neighborhood definition is based on single-vertex moves considered from one part to the other part of a partition. FM starts from a random feasible bipartition and updates the bipartition by a sequence of moves, which are organized as passes over the vertices. In [2], the authors propose an FM-like declustering heuristic without replication.

The rest of the paper is organized as follows. In Section 2, necessary notations and formal definition of the replication problem is given. The proposed scheme is presented in Section 3. In Section 4, we experiment and compare our proposed approach with two replications schemes that are known to perform good on arbitrary queries.

2 Notation and Definitions

Basic notations, concepts and definitions used throughout the paper are presented in this section. We are given a dataset D with $|D|$ indivisible data items and a query set Q with $|Q|$ queries, where a query $q \in Q$ requests a subset of data items, i.e., $q \subseteq D$. Each query q is associated with a relative frequency $f(q)$, where $f(q)$ is the probability that query q will be requested. We assume that query frequencies are extracted from the query log and future queries will be similar to the ones in the query log. We also assume that all data items and all disks are homogeneous and thus, the storage requirement and the retrieval time of all data items on all disks are equal and can be accepted as one for practical purposes.

Definition 1. *Replicated Declustering Problem: Given a set D of data items, a set Q of queries, K homogeneous disks with storage capacity C_{max}, and a maximum allowable replication amount c, find K subsets of D (or a K-way replicated declustering of D), say $R_K = \{D_1, D_2, \ldots, D_K\}$, which, if assigned to separate disks, minimizes the aggregate response time $T(Q)$ for Q and satisfies the following feasibility conditions:*

i. $\cup_{k=1}^{K} D_k = D$
ii. $\sum_{k=1}^{K} |D_k| \leq (1 + c) \times |D|$ *and*
iii. $|D_k| \leq C_{max}$.

Given a multi-disk system with replicated data, the problem of finding an optimal schedule for retrieving the data items of a query arises. The optimal schedule for a query minimizes the maximum number of data items requested from a disk. This problem can be solved optimally by a network-flow based algorithm [13]. Hence, given a replicated declustering R_K and a query q, optimal scheduling $S(q)$ for q can be calculated. $S(q)$ indicates which copies of the data items will be accessed during processing q. So $S(q)$ can be considered as partitioning q into K disjoint subqueries $S(q) = \{q_1, q_2 \dots q_K\}$ where q_k indicates the data items requested by q from disk D_k.

Definition 2. *Given a replicated declustering R_K, a query q and an optimal schedule $S(q) = \{q_1, q_2 \dots q_K\}$ for q, response time $r(q)$ for q is: $r(q) = \max_{1 \le k \le K}\{t_k(q)\}$, where $t_k(q) = |q_k|$ denotes the total retrieval time of data items on disk D_k that are requested by q.*

Definition 3. *In a replicated declustering R_K, the aggregate parallel response time for a query set Q is $T(Q) = \sum_{q \in Q} f(q)r(q)$.*

3 Proposed Approach

Here, we first describe a two-way replicated declustering algorithm and then show how recursion can be applied on top of this two-way replicated declustering to obtain a multi-way replicated declustering. Our algorithm starts with a randomly generated initial feasible two-way declustering of D into D_A and D_B, and iteratively improves this two-way declustering by move and replication operations. Since there are replications, there are three states that a data item can be in: A, B, and AB, where A means that the data item is only in part D_A, B means that the data item is only in part D_B, and AB means that the data item is replicated. Furthermore, for the data items requested by each query, we keep track of the number of data items in each part. That is, $t_A(q)$ indicates the number of data items requested by query q that exists only in part D_A, $t_B(q)$ indicates the number of data items requested by query q that exists only in part D_B, and $t_{AB}(q)$ indicates the number of data items requested by query q that are replicated. Note that the total number of data items requested by query q is equal to $|q| = t_A(q) + t_B(q) + t_{AB}(q)$.

In Algorithm 1, we initialize move gains and replication gains for each data item. First, for each query q, we count the number of data items that are in A, B and AB state among the data items requested by q (lines 1–5). Here, *State* is a vector which holds the states of the data items, i.e., $State(d)$ stores the current state of data item d. Then, we calculate the move gain $g_m(d)$ and the replication gain $g_r(d)$ of each data item (6–17). Note that only non-replicated data items are amenable to move and replication. So, for a non-replicated data item d, $State(d)$ denotes the source part for a possible move or replication associated with data item d. For a non-replicated data item d in state A, the associated move operation changes its state to B, whereas the associated replication operation changes its state to AB. A dual discussion holds for a data item d that is in state B. The for loop in lines 10–18

Algorithm 1. InitializeGains($(\mathcal{D}, \mathcal{Q})$, $\Pi_2 = \{\mathcal{D}_A, \mathcal{D}_B\}$)

Require: $(\mathcal{D}, \mathcal{Q})$, $\Pi_2 = \{\mathcal{D}_A, \mathcal{D}_B\}$
1: **for each** query $q \in \mathcal{Q}$ **do**
2: $t_A(q) \leftarrow t_B(q) \leftarrow t_{AB}(q) \leftarrow 0$
3: **for each** data item $d \in q$ **do**
4: $s \leftarrow State(d)$
5: $t_k(q) \leftarrow t_k(q) + 1$
6: **for each** non-replicated data item $d \in \mathcal{D}$ **do**
7: $g_m(d) \leftarrow g_r(d) \leftarrow 0$
8: $s \leftarrow State(d)$
9: **for each** query q that contains d **do**
10: $\Delta \leftarrow DeltaCalculation(q, State(d))$
11: **if** $\Delta \geq 2$ **then**
12: $g_m(d) \leftarrow g_m(d) + f(q)$
13: $g_r(d) \leftarrow g_r(d) + f(q)$
14: **else if** $(\Delta = 0) \wedge (2(t_k(q) + t_{AB}(q)) = |q|)$ **then**
15: $g_m(d) \leftarrow g_m(d) - f(q)$
16: **else if** $\Delta \leq -1$ **then**
17: $g_m(d) \leftarrow g_m(d) - f(q)$

computes and uses a Δ value for each query that requests d, where Δ represents the difference between the number of data items of q in the source and destination parts under an optimal schedule of query q across these two parts. Algorithm 2 shows the pseudocode for Δ calculation. In lines 1–5 of Algorithm 2, we calculate Δ when some of the replications are unnecessary for query q. This means that some of the replicated data items will be retrieved from the source part s, while others will be retrieved from the other part for q. In this case, if the required number of data items for that query is odd, Δ will be 1, and it will be 0 otherwise. Lines 6–10 indicate that all replications will be retrieved from only one part. We first check the case that all replications are retrieved from the given part s (lines 7–8) and in lines 9–10 we handle the other case.

Algorithm 2. DeltaCalculation(q, s).

Require: $(q \in \mathcal{Q})$, s
1: **if** $|t_A(q) - t_B(q)| < t_{AB}(q)$ **then**
2: **if** $|q|\%2 = 0$ **then**
3: $\Delta \leftarrow 0$
4: **else**
5: $\Delta \leftarrow 1$
6: **else**
7: **if** $t_s(q) < |q| - t_{AB}(q) - t_s(q)$ **then**
8: $\Delta \leftarrow 2 \times t_s(q) + 2 \times t_{AB}(q) - |q|$
9: **else**
10: $\Delta \leftarrow 2 \times t_s(q) - |q|$

Algorithm 3. Update gains after a move from A to B.

Require: $(\mathcal{D}, \mathcal{Q})$, $\Pi_2 = \{\mathcal{D}_A, \mathcal{D}_B\}$, $d^* \in \mathcal{D}_A$
for each query $q \in \mathcal{Q}$ that contains d^* **do**
  $\Delta \leftarrow$ DeltaCalculation(q, A)
  for each non-replicated data item $d \in q$ **do**
    if $d \in \mathcal{D}_A$ **then**
      if $\Delta = 3$ **then**
        $g_m(d) \leftarrow g_m(d) - f(q)$
        $g_r(d) \leftarrow g_r(d) - f(q)$
      else if $\Delta = 2$ **then**
        $g_r(d) \leftarrow g_r(d) - f(q)$
        if $t_{AB}(q) \geq 1$ **then**
          $g_m(d) \leftarrow g_m(d) - f(q)$
        else
          $g_m(d) \leftarrow g_m(d) - 2f(q)$
      else if $\Delta = 1 \wedge t_A + t_{AB}(q) = t_B + 1$ **then**
        $g_m(d) \leftarrow g_m(d) - f(q)$
      else if $\Delta = 0 \wedge |q| = 2(t_B(q) + 1)$ **then**
        $g_m(d) \leftarrow g_m(d) - f(q)$
    else if $d \in \mathcal{D}_B$ **then**
      if $\Delta = 1 \wedge (t_A(q) - t_B(q) = t_{AB}(q) + 1)$ **then**
        $g_m(d) \leftarrow g_m(d) + f(q)$
      else if $\Delta = 0$ **then**
        if $t_{AB}(q) = 0$ **then**
          $g_m(d) \leftarrow g_m(d) + 2f(q)$
          $g_r(d) \leftarrow g_r(d) + f(q)$
        else if $|t_A(q) - t_B(q)| = t_{AB}(q)$ **then**
          $g_m(d) \leftarrow g_m(d) + f(q)$
          if $t_B(q) - t_A(q) = t_{AB}(q)$ **then**
            $g_r(d) \leftarrow g_r(d) + f(q)$
      else if $\Delta = -1$ **then**
        $g_m(d) \leftarrow g_m(d) + f(q)$
        $g_r(d) \leftarrow g_r(d) + f(q)$
  $t_A(q) \leftarrow t_A(q) - 1$
  $t_B(q) \leftarrow t_B(q) + 1$
$State(d^*) \leftarrow B$
$Locked(d^*) \leftarrow 1$

In Algorithms 3 and 4, we update the move and replication gains of the unlocked data items after the tentative move or replication of data item d^* from the source part A to the destination part B, respectively. The dual of these algorithms which performs moves or replications from B to A are easy to deduce from Algorithms 3 and 4. We just update the gains of the data items that share at least one query with the moved or replicated data item d^*. Selection of the operation with maximum gain necessitates maintaining two priority queues, one for moves, one for replications, implemented as binary max-heaps in this work. The priority queue should support extract-max, delete, increase-key and decrease-key operations.

The overall algorithm can be summarized as follows. The algorithm starts from a randomly constructed initial feasible two-way declustering. The

Algorithm 4. Update gains after a replication from A to B.

Require: $(\mathcal{D}, \mathcal{Q})$, $\Pi_2 = \{\mathcal{D}_A, \mathcal{D}_B\}$, $d^* \in \mathcal{D}_A$
for each query $q \in \mathcal{Q}$ that contains d^* **do**
  $\Delta \leftarrow$ DeltaCalculation(q, A)
  for each non-replicated data item $d \in q$ **do**
    if $d \in \mathcal{D}_A$ **then**
      if $\Delta = 3 \vee \Delta = 2$ **then**
        $g_m(d) \leftarrow g_m(d) - f(q)$
        $g_r(d) \leftarrow g_r(d) - f(q)$
    else if $d \in \mathcal{D}_B$ **then**
      if $\Delta = 1 \wedge (t_A(q) - t_B(q) = t_{AB}(q) + 1)$ **then**
        $g_m(d) \leftarrow g_m(d) + f(q)$
      else if $\Delta = 0 \wedge (t_A(q) - t_B(q) = t_{AB}(q))$ **then**
        $g_m(d) \leftarrow g_m(d) + f(q)$
  $t_A(q) \leftarrow t_A(q) - 1$
  $t_{AB}(q) \leftarrow t_{AB}(q) + 1$
$State(d^*) \leftarrow AB$
$Locked(d^*) \leftarrow 1$

initial move and replication gains are computed using the algorithm shown in Algorithm 1. At the beginning of each pass, all data items are unlocked. At each step in a pass, an unlocked data item with maximum move or replication gain (even if it is negative), which does not violate the feasibility conditions, is selected to be moved or replicated to the other part and then it is locked and removed from the appropriate heaps. If maximum move and replication gains are equal, move operation is preferred. If the maximum gain providing operation is an infeasible replication, we trace all replicated data items to see if there are unnecessary replications. If this is the case, we delete those data items to see whether the subject replication operation becomes feasible. We adopt the conventional locking mechanism, which enforces each data item to be moved or replicated at most once during a pass, to avoid thrashing. After the decision of the move or replication operation, the move and replication gains of the affected data items are updated using Algorithms 3 and 4. The change in total cost is recorded along with the performed operation. The pass terminates when no feasible operation remains. Then, the initial state before the pass is recovered and a prefix subsequence of operations, which incurs the maximum decrease in the cost, is performed.

We applied the proposed replicated two-way declustering algorithm in a recursive framework to obtain a multi-way replicated declustering. In this framework, every two-way replicated declustering step for a database system (D, Q) generates two database sub-systems (D_A, Q_A) and (D_B, Q_B). We should note here that, since we can perform replication, $|D_A| + |D_B| \geq |D|$. Since we delete all unnecessary replications at the end of each pass, all replicated data items are necessary in both parts. Thus, all data items in state A and AB are added into D_A, whereas all data items in state B and AB re added into D_B. In order to

perform recursive replicated declustering, splitting of queries is a necessity as well. Each query q is split into two sub-queries depending on the two-way declustering and the optimal schedule for that query. So, the objective at each recursive two-way declustering step models the objective of even distribution of queries into K disks. We only discuss recursive declustering for the case when K is a power of two. However, the proposed scheme can be extended for arbitrary K values by enforcing properly imbalanced two-way declusterings. For $K = 2^\ell$, the storage capacity at the ith recursion level is set to be $C_{max} \times (K/2^i)$ for $i = 1, \ldots, \ell$. In our current implementation, the global maximum allowable replication amount c is applied at each recursive step where each replication operation reduces it by one and each deletion increases it by one.

4 Experimental Results

In our experiments, we used three of the datasets used in [2] along with the synthetically created arbitrary query sets. We used homogeneous data sizes, equal query frequencies, and homogeneous servers. We tested the proposed Selective Replicated Declustering (SRD) algorithm on these datasets and compared SRD with the RDA and OA algorithms. These algorithms are known to perform good for arbitrary queries and they can support partial replication. All algorithms are implemented in C language on a Linux platform.

Table 1 shows the properties of the three database systems used in the experiments. Further details about these datasets can be found in [2]. We have tested all of our datasets under varying replication constraints.

Figs. 1 and 2 display the variation in the relative performances of the three replicated declustering algorithms with increasing replication amounts for $K = 16$ and $K = 32$ disks, respectively. For the RDA and OA schemes, the data items to be replicated are selected randomly. In Figs. 1 and 2, the ideal response time refers to the average parallel response time of a strictly optimal declustering if it exists. So, it is effectively a lower bound for the optimal response time. Note that a declustering is said to be strictly optimal with respect to a query set if it is optimal for every query in the query set. A declustering is optimal for a query $q \in Q$, if the response time $r(q)$ is equal to $\lceil |q|/K \rceil$, where K is the number of disks in the system.

The relative performance difference between the replicated declustering algorithms decreases with increasing amount of replication as expected. The

Table 1. Properties of database systems used in experiments (taken from [2])

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{Q}\rvert$	Average query size
HH	1638	1000	43.3
FR	3338	5000	10.0
Park	1022	2000	20.1

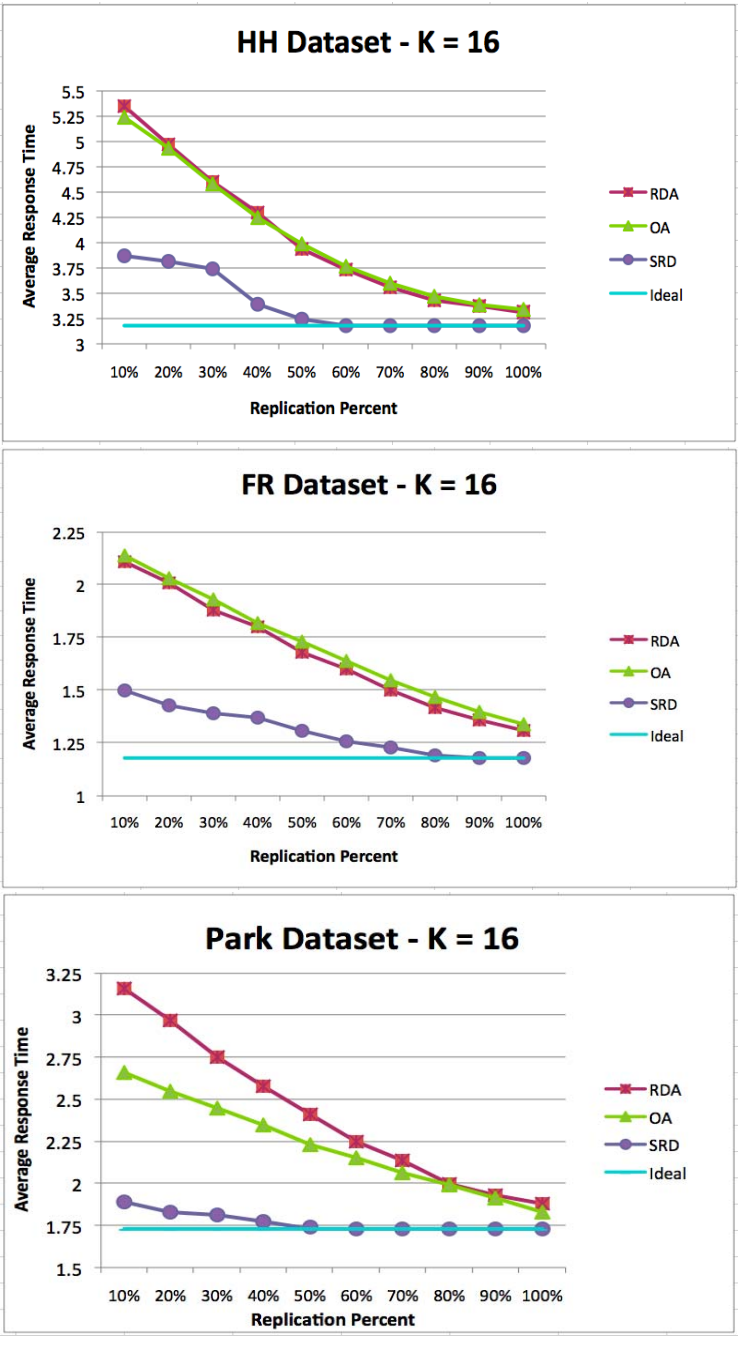

Fig. 1. Comparison of average response time qualities of the Random Duplicate Assignment (RDA), Orthogonal Assignment (OA), and Selective Replicated Declustering (SRD) schemes with increasing replication for $K = 16$ disks

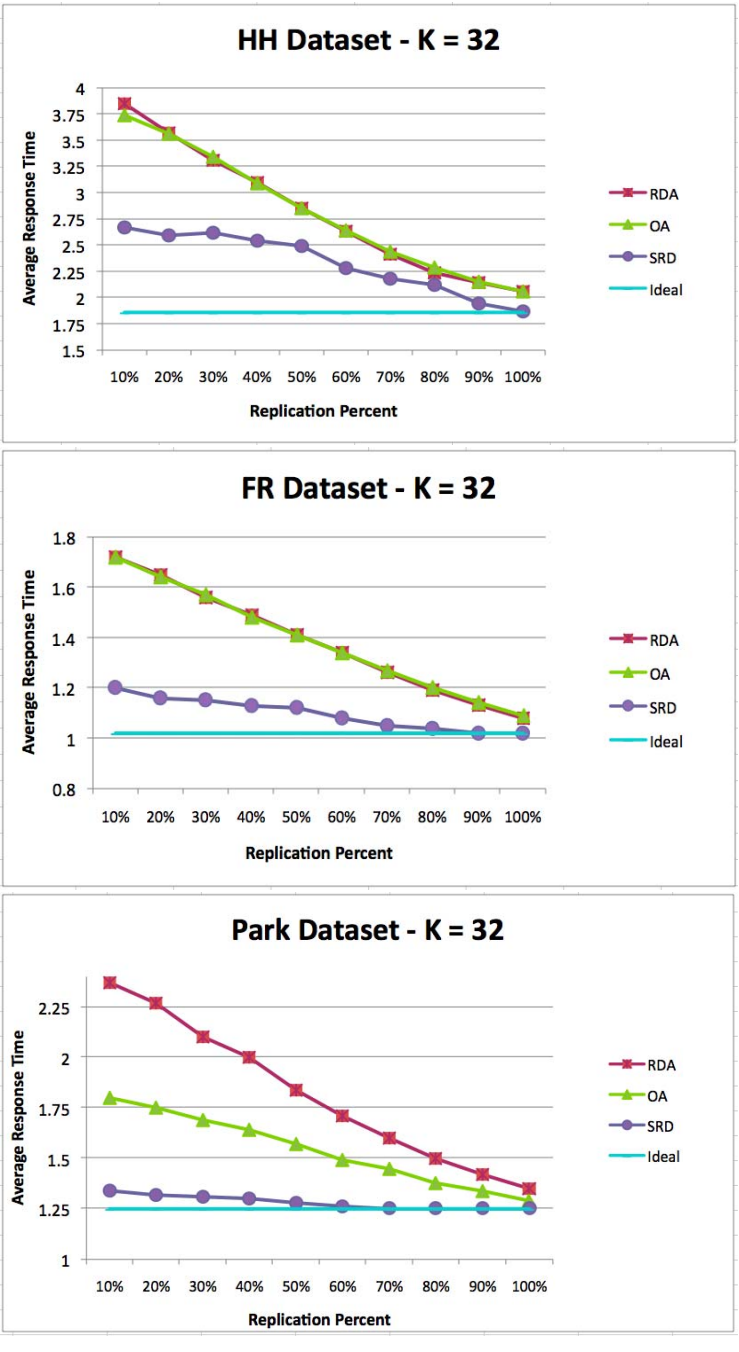

Fig. 2. Comparison of average response time qualities of the Random Duplicate Assignment (RDA), Orthogonal Assignment (OA), and Selective Replicated Declustering (SRD) schemes with increasing replication for $K = 32$ disks

existing algorithms RDA and OA show very close performance for the *HH* and *FR* datasets, whereas OA performs better than RDA for the *Park* dataset.

As seen in Figs. 1 and 2, the proposed SRD algorithm performs better than the RDA and OA algorithms for all declustering instances. We observe that the proposed SRD algorithm achieves very close to the ideal values and in fact achieves ideal results even for replication amounts less than 100% in all datasets apart from 32-way declustering of the *HH* dataset (where it achieves the ideal result at 100% replication). Furthermore, SRD provides very good response time results even for very low replication amounts such as 10% or 20%.

5 Conclusions

We proposed and implemented an efficient and effective iterative improvement heuristic for selective replicated two-way declustering that utilizes a given query distribution and a recursive framework to obtain a multi-way replicated declustering. We tested the performance of our algorithm on three real datasets with synthetically generated arbitrary queries. Our initial implementation indicates that the proposed approach is promising since we obtained favorable performance results compared to two state-of-the-art replicated declustering schemes that performs well in arbitrary queries.

As a future work, we will investigate development and implementation of efficient replica deletion schemes, intelligent schemes that set adaptive maximum allowable replication amounts across the levels of the recursion tree, and a multi-way refinement scheme for iterative improvement of the multi-way replicated declustering obtained through recursive two-way declusterings. This work assumes homogeneous data item sizes and homogeneous disks. Heterogeneity in both aspects can also be considered as a future research area.

References

1. Tosun, A.S.: Threshold-based declustering. Information Sciences 177(5), 1309–1331 (2007)
2. Koyuturk, M., Aykanat, C.: Iterative-improvement-based declustering heuristics for multi-disk databases. Information Systems 30, 47–70 (2005)
3. Liu, D.R., Shekhar, S.: Partitioning similarity graphs: a framework for declustering problems. Information Systems 21, 475–496 (1996)
4. Liu, D.R., Wu, M.Y.: A hypergraph based approach to declustering problems. Distributed and Parallel Databases 10(3), 269–288 (2001)
5. Ozdal, M.M., Aykanat, C.: Hypergraph models and algorithms for data-pattern-based clustering. Data Mining and Knowledge Discovery 9, 29–57 (2004)
6. Demir, E., Aykanat, C., Cambazoglu, B.B.: A link-based storage scheme for efficient aggregate query processing on clustered road networks. Information Systems (2009), doi:10.1016/j.is.2009.03.005
7. Demir, E., Aykanat, C., Cambazoglu, B.B.: Clustering spatial networks for aggregate query processing: A hypergraph approach. Information Systems 33(1), 1–17 (2008)

8. Tosun, A.S.: Analysis and comparison of replicated declustering schemes. IEEE Trans. Parallel Distributed Systems 18(11), 1587–1591 (2007)
9. Sanders, P., Egner, S., Korst, K.: Fast concurrent access to parallel disks. In: Proc. 11th ACM-SIAM Symp. Discrete Algorithms, pp. 849–858 (2000)
10. Tosun, A.S.: Replicated declustering for arbitrary queries. In: Proc. 19th ACM Symp. Applied Computing, pp. 748–753 (2004)
11. Tosun, A.S.: Design theoretic approach to replicated declustering. In: Proc. Int'l Conf. Information Technology Coding and Computing, pp. 226–231 (2005)
12. Fiduccia, C.M., Mattheyses, R.M.: A linear-time heuristic for improving network partitions. In: Proc. of the 19th ACM/IEEE Design Automation Conference, pp. 175–181 (1982)
13. Chen, L.T., Rotem, D.: Optimal response time retrieval of replicated data. In: Proc. 13th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, pp. 36–44 (1994)