Query-Log Aware Data Replicated Declustering

Anisaara Nadaph, Vikas Maral

Abstract— Query-Log is general record of what mysqld is doing, the server writes information to this log when client connect or disconnect. Declustering parallelizes the query retrieval process by distributing the data items requested by queries among several disks. Replication enables alternative disk choices for individual disk items and thus provides better query parallelism options. existing replicated declustering schemes do not consider query log information and try to optimize all possible queries for a specificquery type, such as range or spatial queries. In such schemes, it is assumed that two or more copies of all data items are to be generated and these data items are copied to disks by different algorithm. However, It is not feasible in some applications for generation of even two copies of all of the data items, since data items tend to have very large sizes. In this work we assume that there is a given limit on disk capacities and thus on replication amounts. We utilize existing query-log information to propose a selective replicated declustering scheme, in which we select the data items to be replicated and decide on their scheduling onto disks. We suggest an iterative algorithm to get a two-way replicated decluster and by making use of this algorithm recursively to generate a multiway replicated declustering. Later by makiing use of efficient heuristics we improve the multi-way replicated declustering. The survey show that the suggested work gives bettter performance result over the existing replicated declustering schemes.

Index Terms— Declustering, replication, parallel disk architectures, iterative improvement heuristics.

I. INTRODUCTION

The declustering problem is to partition data across multiple disks which can be accessed in parallel to reduce the query response time Minimizing query-response times is a crucial issue in designing high-performance database systems for application domains such as scientific, spatial and multimedia. These systems are often used interactively and amounts of data to be retrieved for individual queries are quite large. In such database systems, the I/O bottleneck is overcome through parallel I/O across multiple disks. Disks are accessed in parallel while processing a query, response time for a query can be minimized by balancing the amount of data to be retrieved on each disk. Therefore, data is distributed across multiple disks, respecting disk capacity constraints, in such a way that data items that are more likely to be retrieved together are located into separate disks. This operation is known as declustering.

There are some applications that also query very large data items in a random fashion and in such applications utilization of query log information is of essence for efficient declustering [1] [3][5].

In the declustering problem with a given query distribution is modeled as a max-cut partitioning of a weighted similarity graph, where data items are represented by vertices and an edge between two vertices implies that corresponding data items appear in at least one common query. In [3] and [5], the deficiencies of the weighted similarity graph modelare addressed and hypergraph models which encode the total I/O cost correctly are proposed.

Data replication is a widely applied technique in various application areas such as distributed data management[6] and information retrieval [7], to achieve fault tolerance and fault recovery. Data replication can also be exploited to achieve higher I/O parallelism in a declustering system. However, while performing replication, one has to be careful about consistency considerations, which arise in update and delete operations.Furthermore, write operations tend to slow down when there is replication. Finally, replication means extra storage requirement 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.

When there is data replication, the problem of query scheduling has to be addressed as well. That is, when a query arrives, we have to decide which replicas will be used to answer the query. A maximum-flow formulation is proposed to solve this scheduling problem optimally. There are replicated declustering schemes that aim to minimize this scheduling overhead while minimizing I/O costs. A variation of this problem arises when replicas are assumed to be distributed over different sites, where each site hosts aparallel-disk architecture [8].

This variation can be modeled as a maximum flow problem as well. N number of Existing replicated declustering schemes proposed for range queries. There are some replicated declustering schemes proposed for arbitrary queries as well. All of these schemes assume items with equal sizes and they also assume that all data items will be requested equally likely and thus generate equal number of replicas for all data items. Furthermore, they replicate all data items two or more times.

Table 1: Notation used

Symbol	Description
D	Dataset
Q	Query set
K	Total number of disks
Dk	Set of data items assigned to disk k
Cmax	Maximum storage capacity of a disk
di	A data item in the dataset
s(di)	Storage requirement for data item di
q	A query in the query set
|q|	Number of data items requested by q
f(q)	Frequency of q in Q
r(q)	Response time for q
tk(q)	Response time of disk k for q
Sopt(q)	Optimal scheduling for q
ropt(q)	Optimal response time for q

Manuscript received on April, 2013.

Anisaara Nadaph, Computer Engg., K.J College Of Engg. And Management Research, Pune, Pune, India.

Prof. Vikas Maral, Computer Engg., K.J College Of Engg. And Management Research, Pune, Pune, India.

Rk	A K-way replicated declustering
Tr(Rk,Q)	Parallel response time of RK for Q
Tropt(Q)	Optimal parallel response time for Q
TrO(Rk,Q)	Parallel response time overhead of RK for Q
gm(d)	n a two-way replicated declustering phase, reduction to be observed in the overall query processing cost, if d is moved to the other disk.
gr (d)	In a two-way replicated declustering phase, reduction to be observed in the overall query processing cost, if d is replicated in both disks.
guX (d)	In a two-way replicated declustering phase, reduction to be observed in the overall query processing cost, if a replica of d is deleted from disk DX .
vg(d)	Number of queries requesting d such that the disk(s) that d resides in serve(s) more than optimal number of data items for these queries.
gm (d, k)	In a K-way replicated declustering phase, reduction to be observed in the overall query processing cost, if d is moved to disk k.
gr (d, k)	In a K-way replicated declustering phase, reduction to be observed in the overall query processing cost, if d is replicated in disk k.

II. EXISITNG WORK AND ITS CONTRIBUTION

We present a selective and query-log aware replication scheme which works in conjunction with declustering. The proposed scheme utilizes the query log information to minimize the aggregate parallel query response time while obeying given replication constraints due to disk sizes. There are no restrictions 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 first propose an iterative-improvement-based replicated two-way declustering algorithm. In this algorithm, in addition to the replication operation that we proposed, we successfully incorporate unreplication operation to the replicated two-way declustering algorithm to prevent unnecessary replications. We also provide simple closed-form expressions for computing the cost of a query in a two-way replicated declustering. Utilizing these expressions, we avoid usage of expensive network-flow based algorithms for the construction of optimal query schedules. By recursively applying our two-way replicated declustering algorithm we obtain a K-way replicated declustering. Our unreplication algorithm prevents unnecessary replications to advance to the next levels in the recursive framework.

We then propose an efficient multi-way replicated refinement heuristic that considerably improves the obtained K-way replicated declustering via multi-way move and multi-way replication operations. In this iterative algorithm, we adapt a novel idea about multiway move operations and obtain an efficient greedy multi-way move/replication scheme. We also present an efficient scheme to avoid the necessity of computing the optimal schedules of all queries at each iteration of our multi-way refinement algorithm. The proposed scheme enables us to compute the optimal schedules of all queries just once, at the beginning of the multi-way refinement, and then update the schedules incrementally according to the performed operations.

III. PROPOSED A PPROACH

We suggest a two-phase approach for solving the K-way replicated declustering problem.

Phase 1: In this phase, we use a recursive replicated declustering heuristic to obtain a K-way replicated declustering. We should note that, by allowing extreme two-way declusters in this phase, we are able to obtain K-way declusterings for arbitrary K values.

Phase 2: In this, we use a refinement heuristic to improve the K-way replicated declustering obtained in the first phase.

(A) Phase study

Phase 1(Recursive replicated declustering phase):

The objective in the recursive replicated declustering phase is to evenly distribute the data items of queries at each two-way replicated declustering step of the recursive framework. That is, at each two-way step, we try to attain optimal response time ropt(q) = [|q|/2] for each query q as much as possible

Two-way replicated declustering :

In the two-way replicated algorithm, we start with a given (and possibly randomly generated) initial feasible two-way declustering of the dataset D

R2 = {DA,DB} and iteratively improve R2 by three refinement operations defined over the data items Namely

i. Move
ii. Replication
iii. Unreplication operations

i. **move gain (gm (d)) :** reduction in the overall query processing cost, if d is moved to the other disk,

ii. r**eplication gain (gr (d))** : reduction in the overall query processing cost, if d is replicated to the other disk,

iii. **unreplication-from-A gain (guA (d))** : reduction in the overall query processing cost, if a replica of d is deleted from DA ,

iv. **unreplication-from-B gain (guB (d)) :** reduction in the overall query processing cost, if a replica of d is deleted from DB.

A two-way replicated declustering R2 = {DA , DB } can be considered as partitioning the dataset D into three mutually disjoint parts: A, B, and AB, where part A is composed of the data items that are only stored in disk DA , part B is composed of the data items that are only stored in disk DB , and part AB is composed of the data items that are replicated. In this view, DA = A U AB and DB = B U AB.

A variable State(d) is kept to store the part information of each data item d.

For each query q, we maintain a 3-tuple

dist(q) = (|qA | : |qB | : |qAB |), -------(7)

where |qA |, |qB |, and |qAB | indicate the number of data items of q in parts A, B, and AB, respectively. That is,

qA = q ∩ A , qB = q ∩ B and qAB = q ∩ AB. --------(8)

The total number of data items requested

by query q is
equal to: |q| = |qA | + |qB | + |qAB |
query q from disks DA and DB can be written as follows, without loss of generality assuming that |qA | ≥ |qB |:

tA (q) = |q|/2 if |qAB | ≥ (|qA | − |qB |) − 1
|qA | otherwise
tB (q) = |q|/2 if |qAB | ≥ (|qA | − |qB |) − 1
|qB | + |qAB | otherwise

---------- (9)

Table A : Condition for Equation (9)

Condition	
qAB	≥ (|qA | − |qB |) − 1	implies the case in which there are enough number of replicated data items requested by q that can be utilized to achieve even distribution of q among DA and DB .
otherwise	condition corresponds to the case for which even distribution of q among the disks is not possible.

The cost r(q) of q can be computed with the following closed-form expression:
|q|/2 if |qAB | ≥ (||qA | − |qB ||) − 1
max(tA (q), tB (q)) other --------(10)
That is, r(q) in Equation 10 gives the cost of query q that can be attained by an optimal schedule for q, without constructing Sopt (q) through costly network-flow based algorithms.Our overall two-way replicated declustering algorithm works as a sequence of two-way refinement passes performed over all data items. In each pass, we start with computing the initial operation gains of all data items. Then, we iteratively perform the following computations: find the data item and the operation that produces the highest reduction in the cost; perform that operation; update gain values of neighboring data items; lock the selected data item to further processing to prevent thrashing.
We perform these computations until there are no remaining data items to process. We restore the declustering to the state where the best reduction is obtained during the pass and we start a new pass over the data items if the obtained improvement in the current pass is above a threshold or if the number of passes performed is below some predetermined number. Once we obtain a two-way declustering, we can recursively apply our two-way declustering algorithm on each of these declusters to obtain any number of declusters. All operations are kept in priority queues keyed according to their gain values. The priority queues are implemented as binary heaps. For a two-way declustering, we maintain six heaps:
2 Heaps : storing the move operations of data items from part A to B and from part B to A
2 Heaps : storing the replication operations of data items from part A to B and from part B to A
2 Heaps : storing the unreplication operations of replicated data items from part A and from part B.
In our two-way replicated declustering algorithm, we start with calculating the initial move, replication and unreplication gains of all data items (Appendix, Algorithm 2). After initializing the gains, we retrieve the highest gains and the associated data items for each operation type and by comparing these gains we select the best operation to perform. If there are any possible unreplication operations which do not increase the total cost of the system (i.e., with zero unreplication gain), those unreplication operations are performed first. After we finish possible unreplications, we compare the gains to be obtained by move and replication operations. If the gains are the same, we prefer to perform move operations. Recall that each data item is eligible for two types of operations and thus has two related gain values. So, after deciding on the best operation to perform, we remove the data item from the two related heaps by extractMax and delete operations. After performing an operation (move, replication or unreplication) on a data item d∗ , we may need to update the gains of operations related with the data items that are neighbor to d' (Appendix, Algorithms 3, 4, and 5). For any data item d, we have gr (d) ≥ gm (d), hence, in a pass, the number of replication operations tend to outweigh the number of move operations.

(B) Query Splitting

At the last phase of a two-way replicated declustering R2 ={DA,DB} of a dataset and query set pair {D,Q}, we split the queries of Q among the obtained two sub-datasets as evenly as possible so that split queries correctly represent the optimizations performed during that two-way replicated declustering step. That is, an R2 is decoded as splitting each query q ∈ Q into two sub-queries

q' ⊆ q ∩ DA and q'' ⊆ q ∩ DB, --------(12)

such that the difference ||q'|−|q''|| is minimized. The split queries q' and q'' are added to sub-query sets QA and QB, respectively so that further two-way declustering operations can be recursively performed on {DA,QA} and {DB,QB} pairs. Constructing the optimalschedule of a query q in a replicated declustering system requires network-flow-based algorithms. However, for two-way replicated declustering this feat can be achieved by utilizing the item distribution dist(q) of q and the value of r(q), which can be computed via the closed form definitions given in Equations 7–10. We know that in an optimal splitting according to the optimal schedule, the size of q should be |q | = tA (q) and the size of q should be |q | = tB (q).

(C) Three-way partition of query:

Three-way partition of query q into qA ,qB , and qAB (according to Equation 8) induced by the two-way replicated declustering.

Notations Used	Description
q'	Items from qA
q"	Items from qB
q'AB	Replicated Items that goes in q'
q"AB	Replicated Items that goes in q"

q '= qA Ú q'AB and q" = qB Ú q"AB ,
Since we want to enforce a splitting such that
|q '| = tA (q)

and |q" | = tB (q),
we can say that

|qAB | = tA (q)−|qA | and
|qAB | = tB (q)−|qB | = |qAB |−|qAB |. (14)
Any splitting of the data items in qAB that respects the size constraints given in Equation 14 satisfies

the optimality condition. In our studies we assign the first tA (q) − |qA | items of qAB to q and the remaining items of qAB to q . Other assignment schemes can be explored for better performance results.

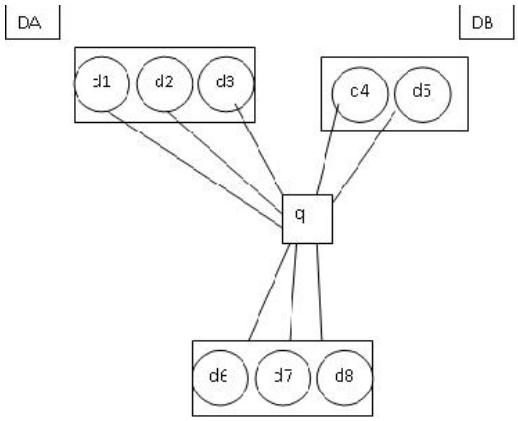

Fig. 1 Query(q) splitting according to a two-way replicated declustering R2={DA,DB}.

here
qA={d1,d2,d3}
qB={d4,d5}
qAB={d6,d7,d8}
q'AB={d6}
q''AB={d7,d8}
q'={d1,d2,d3,d6}
q''={d4,d5,d7,d8}
Splitting of a query q with eight data items is given in Fig.1.
According to Equation 9,
q, tA (q) = 4
and tB (q) = 4, hence |q | = 4 and |q | = 4
From diagram
qA | = 3, |qB | = 2
By Equations 13 and 14,
we can say that |qAB | = tA (q) − |qA | = 1 and |qAB | = tB (q) − |qB | = 2.
Any splitting of qAB according to thesize constraints satisfies the optimality condition, and according to our assignment scheme qAB = {d6 } and qAB = {d7 , d8 }. Hence, q = qA U qAB = {d1 , d2 , d3 , d6 }
and q = qB U qAB = {d4 , d5 , d7 , d8 }.

Phase 2(Multi-way replicated refinement)

The survey of multi-way replicated refinement scheme
Using the above notation, the retrieval times of a given starts with the K-way replicated declustering of the dataset D, say RK = {D1 , . . . , DK }, generated by the recursive replicated declustering scheme d, RK is Iteratively improve by multi-way refinement operations K-way move and K-way eplication. In order to perform these operations we maintain the following gain values for each data item d

Table B : Reduction Calculation

Move gain(gm(d,k))	reduction to be observed in the overall query processing cost, if d is moved to disk k,
replication gain (gr (d, k)	reduction to be observed in the overall query processing cost, if d is replicated in disk k.

If we were to maintain the above gain values for all dataitems, we would need approximately 2 × (K − 1) gain values for each data item, because a data item can be moved or replicated from its current source disk(s) to any of the disks that does not already store it. Instead of this expensive schema, we adapt an efficient greedy approach that was proposed for unreplicated declustering in [4] to support multi-way refinement and we develop a multi-way refinement heuristic suitable for replicated declustering. Our heuristic can perform multi-way move and replication operations.

Early Approach

early approach was based on the observation that a move operation can be viewed as a two-stage process, where in the first stage the data item d' to be moved is assumed to leave the source disk and in the second stage d∗ arrives at the destination disk. The first stage represents the decrease in the load of the source disk due to the relief in processing of the queries related with d' , resulting with a decrease in the cost. The second stage represents the increase in the load of the destination disk due to the excess in processing of the queries related with d', resulting with an increase in the cost.

Surveyed Approach

Here we extend this efficient greedy approach to support both multi-way move and replication selection operations. Our adapted schema requires maintenance of only a single gain value (virtual leave gain) for each data item d.Virtual leave gain vg(d) indicates the number of queries requesting d such that the disk(s) that d resides in serve(s) more than optimal number of data items for these queries. That is, the virtual leave gain of a data item d that resides on disk Ds is:

$$vg(d) = \sum_{q'Q+(d,s)} f(q), \text{ where} \qquad \text{--------(15)}$$

$$Q+(d,s) = \{q \text{ ' } Q : d \text{ ' } q' \text{ ts } (q) > \text{ropt } (q)\} \qquad \text{--------(16)}$$

That is, each query q that requests data item d contributes f (q) to vg(d), if the number of data items in q that are retrieved from disk Ds is greater than the optimal response time ropt (q) of q. This means that it is possible to improve the distribution of query q through moving or replicating data item d to an appropriate destination disk Dz . Thus, virtual leave gain is an upper bound on the actual move or replication gain. We should note here that our definition of virtual leave gain is different from the early algorithm
Our overall K-way replicated declustering refinement algorithm works as a sequence of multi-way passes per-formed over all data items. Before starting the multi-way refinement passes, as a preprocessing step, we compute the optimal schedules for all queries once and maintain these schedules in a data structure called OptSched. The process of initial optimal schedule calculation is performed using network-flow based algorithms [9]. OptSched is composed of |Q| arrays, where the ith array is of size |qi | and stores from which disks the data items of qi are answered in the optimal scheduling. OptSched is kept both to identify bottleneck disks for queries and also to report the actual aggregate parallel response time of the replicated declustering produced by the recursive declustering phase. A bottleneck disk for a query q is the disk from which q requests the maximum number of data items (and hence determines response time r(q)). In a multi-way refinement pass, we start with computing the virtual leave gains of all data items (Appendix, Algorithm 6). At each iteration of a pass, a data item d' with the highest virtual leave gain is selected. The K−1 move and K−1 replication gains associated with d' are computed (Appendix, Algorithm 7), the best operation associated with d∗ is selected and performed if it has a

positive actual gain and if it obeys the given capacity constraints, and then the virtual leave gain values of the neighboring data items of d' are updated (Appendix, Algorithm 8). Also the optimal schedules of each query that requests d' is considered for update in constant time by investigating possible changes in the bottleneck disk of that query. We perform these passes until the obtained improvement is below a certain threshold or we reach a predetermined number of passes.

Survey Experimental Result

T\he survey results conducted to compare the performance of the proposed Selective Replicated Assignment (SRA) scheme against the state-of-the-art Random Duplicate Assignment (RDA) and Orthogonal Assignment (OA) schemes. RDA and OA are selected since they are known to perform good for arbitrary queries. Also it is possible to modify these approaches for selective replication. We modified both RDA and OA to support partial replication, and improved RDA such that it utilizes query logs and selects the most frequently requested data items and replicates them at random disks. We call this modified version the Most Frequent Assignment (MFA) scheme.

In the comparisons we used 3 datasets: Airport, Park, State. The properties of these datasets are presented in Table 2.

Table 2 : Properties of dataset

Class	D-Data set	|D|	|Q|	Avearge query size
GIS(Point)	Airport	1176	5000	22.8
GIS(Polygon)	Ntar	8952	10000	29.2
GIS(Polygon)	Park	1022	4000	20.1

While testing the performance of MFA and SRA, the query sets for all datasets are divided into two equal parts. The first half is used for replication and declustering and the second half is used for testing the performance. All of the algorithms used in the experiments are implemented in C programming language, and experiments are conducted on a 2GHz Intel Core Duo machine with 2MB L2 cache and 2GB DDR2 667 MHz memory. Query processing performances of the compared algorithms are tested on K=16, 24, 32 disks and the allowed overall replication ratio is varied from 10% to 100%. With 3 different datasets, 3 different disk counts, and 10 different replication ratio values, For each SRA experiment instance, we report the average of 10 runs, since we use randomly generated initial feasible two-way declusterings in our replicated declustering phase.

The query processing performance of a given algorithm is evaluated in terms of the average retrieval overhead per query induced by the resulting replicated declustering. Here, average retrieval overhead per query . (arO) for a given replicated declustering of a dataset and a query set is defined as total response time overhead (Equation 5) divided by the number of queries. That is,

$$arO(Q) = T\ rO(RK , Q)/|Q|. \quad \text{----------(17)}$$

average retrieval overhead of SRA over the two datasets with increasing replication ratio, where the allowed replication ratio is distributed between the recursive replicated declustering and multi-way refinement phases according to the percentage values displayed over the columns. In Table 3, we present the arithmetic averages of the average retrieval overhead of SRA over the two datasets with increasing replication ratio, where the allowed replication ratio is distributed between the recursive replicated declustering and multi-way refinement phases according to the percentage values displayed over the columns.

Table 3 : Arithematic average of the arO values for K=32 disks over 3 datasets.

	percent distribution of replication ratioamong recursive replicated declustering and multi-way refinement phases						
withou t unrep.		with unreplication					
100–0	100–0	80–20	60–40	40–60	20–80	0–100	% rep
0.40	0.31	0.29	0.27	0.24	0.23	0.21	10%
0.36	0.28	0.25	0.22	0.19	0.16	0.15	20%
0.32	0.22	0.23	0.19	0.15	0.12	0.11	30%
0.25	0.19	0.20	0.18	0.12	0.09	0.09	40%
0.19	0.15	0.15	0.14	0.10	0.06	0.07	50%
0.15	0.12	0.13	0.13	0.09	0.05	0.06	60%
0.12	0.09	0.11	0.11	0.09	0.04	0.05	70%
0.10	0.07	0.09	0.09	0.08	0.03	0.04	80%
0.07	0.05	0.07	0.07	0.08	0.03	0.03	90%
0.06	0.04	0.06	0.06	0.07	0.02	0.02	100 %

Table C : Column Header Information of Table 2

80%	the recursive replicated declustering phase is allowed to utilize of the 80% replications
20%	the multiway refinement phase is allowed to utilize 20% of the replications.

The values in the table indicate the retrieval overhead of the replicated declusterings obtained by SRA under the given replication distribution.

2nd Column Indicate

When there is UnReplication and 100-0% where onlt replication is performed in recursive replicated declustering phase.

Analysis of Experiment

i. Low replication ratios (between 10% to 30%), the average results obtained by SRA are best when the given replication amount is fully utilized in the multi-way refinement phase (0%–100% replication distribution).

ii. For higher replication ratios (between 40% to 100%), best results are obtained in 20%–80% replication-distribution scheme. These results indicate that, for small allowed replication ratios, performing replications at a later phase whereas for higher allowed replication ratios, performing a small percent of the replications at an earlier phase, that is during the recursive bipartitioning phase, has a more positive effect on the overall SRA performance. The 20%-80% replication-distribution scheme, which is a combination of recursive replicated declustering and multi way replicated refinement schemes most of the time outperforms the 0%–100% scheme, which is an approach where replication and declustering is decoupled demonstrates the need for our recursive replicated declustering algorithm.

X-axis : - percent replication ratio
Y-axis :- average retrieval overhead (aro)

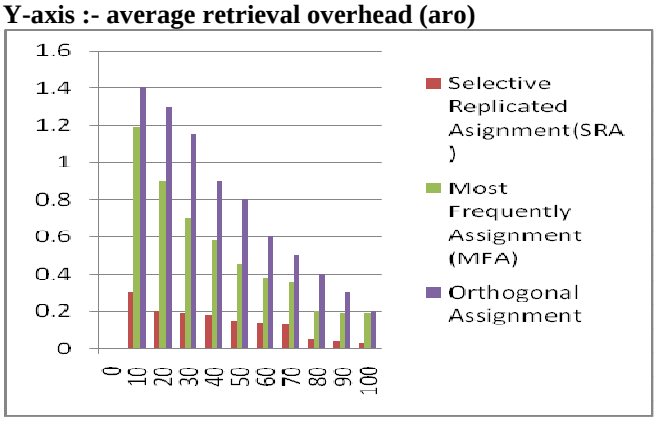

Fig. 2(a)Airport Dataset

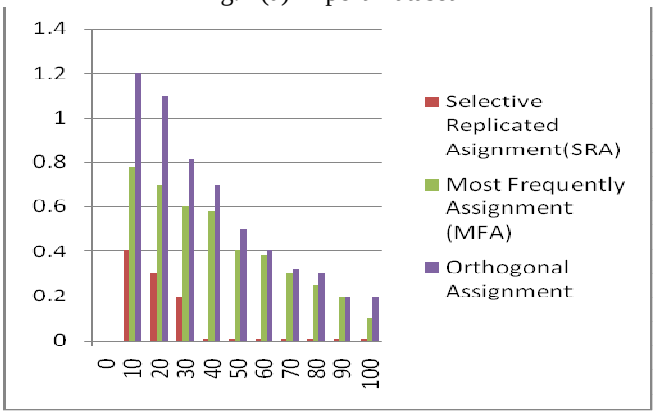

Fig. 2(b)Park Dataset

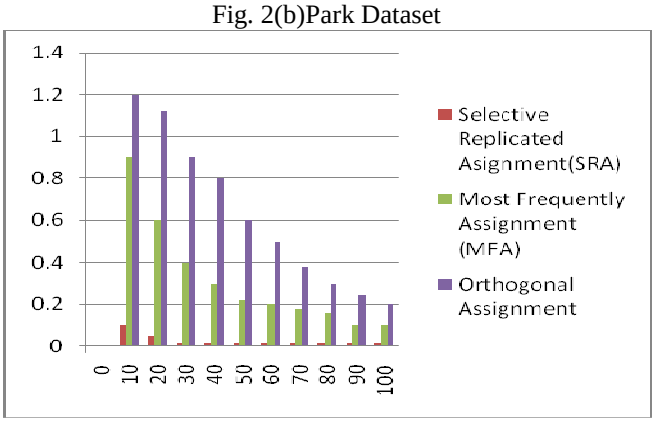

Fig. 2(c)Park Dataset

Fig. 2 display the individual performances of the algorithms for K=24 disks over the 9 datasets. Similar detailed analysis for K=16 and K=32 disks. In the figure, variation of the arO values of algorithms with increasing replication ratio is presented. Closer points to x-axis mean better average retrieval times. As seen in the figure, SRA has better (smaller) average retrieval time than MFA and OA for all experiment instances. While comparing MFA with OA, MFA performs much better than OA . Only in Park datasets OA performs slightly better than MFA. We observe that with increasing replication amount, the deviation of OA from the strictly optimal declustering decreases linearly, whereas in both MFA and SRA we observe a quadratic decrease. These results point to the importance of using query logs in improving performance, since MFA also makes use of query logs by replicating frequently requested items. An analysis of Fig. 2 reveals that the performance gap between the proposed SRA algorithm and the state-of-the-art MFA and OA algorithms decreases with increasing replication amount. However, as also seen in the figure, SRA still performs considerably better then MFA and OA even for high replication amounts.

IV. CONCLUSION

In the survey conducted, we propose a K-way replicated declustering scheme that utilizes a given query distribution. To improve quality of two-way replicated declustering we propose an Iterative Improvement based two-way replicated declustering scheme. We recusively apply two-way scheme to obtain a K-way replicated declustering. Then a effictive multi-way refinment scheme was proposed tha can perform multi-way move and replication of data items. With this scheme, We further improve the quality of the obtained K-way declustering and improve balance if possible. There are some experimental result on homogeneous disk with homogeneous data item retrieval time and the experiment indicates the merits of utilizing query logs in partial and selective replication. The proposed scheme achieves much better results compared to state-of-the-art replicated declustering schemes, many times achieving optimal overall response time with less than 100% replication ratio.

V. ACKNOWLEDGMENT

The Author is thankful to Prof. Vika Maral who has guided Author for the survey. Also to Prof. Das and Prof.Mehtre from ―K.J. College of Engg. And Management Research, Pune, for their help and suggestions.

REFERENCES

[1] Query-Log Aware Replicated Declustering Ata Turk, K. Yasin Oktay and Cevdet Aykanat.
[2] D. R. Liu and S. Shekhar, “Partitioning similarity graphs: a frame-work for declustering problems,” Information Systems, vol. 21, pp. 475–496, 1996.
[3] D. R. Liu and M. Y. Wu, “A hypergraph based approach to declus-ering problems,” Distributed and Parallel Databases, vol. 10(3),pp. 269–288, 2001..
[4] A. S. Tosun, “Threshold-based declustering,” Information Sciences, vol. 177(5), pp. 1309–1331, 2007
[5] M. Koyuturk and C. Aykanat, “Iterative-improvement-based declustering heuristics for multi-disk databases,” Information Systems, vol. 30, pp. 47–70, 2005.
[6] J. Gray, P. Helland, P. O’Neil, and D. Shasha, “The dangers of replication and a solution,” in Proc. ACM SIGMOD International Conference on Management of Data, pp. 173–182, 1996J.
[7] S. Ghemawat, H. Gobio, and S. T. Leung, “The google file system,” ACM SIGOPS Operating Systems Review, vol. 37(5), pp. 29–43, 2003.
[8] A. S. Tosun, “Multi-site retrieval of declustered data,” in Proc. 28th Int’l Conf. Distributed Computing Systems, pp. 486–493, 2008.
[9] “National transportation atlas databases.” CD-ROM, 1999. Bureau of Transportation Statistics.