On hierarchical clustering-based approach for RDDBS design

Hassan I. Abdalla
Zayed University, hassan.abdalla@zu.ac.ae

Ali A. Amer
Zayed University, amal.amer@zu.ac.ae

Sri Devi Ravana
Universiti Malaya

Journal of Big Data

METHODOLOGY Open Access

On hierarchical clustering-based approach for RDDBS design

Hassan I. Abdalla[1], Ali A. Amer[1,2*] and Sri Devi Ravana[3]

*Correspondence: aliaaa2004@yahoo.com; aliaaa2004@gmail.com

[1] College of Technological Innovation, Zayed University, Abu Dhabi, UAE
[2] Computer Science Department, Taiz University, Taiz, Yemen
[3] Department of Information Systems, Faculty of Computer Science and Information Technology, Universiti Malaya, Kuala Lumpur, Malaysia

Abstract

Distributed database system (DDBS) design is still an open challenge even after decades of research, especially in a dynamic network setting. Hence, to meet the demands of high-speed data gathering and for the management and preservation of huge systems, it is important to construct a distributed database for real-time data storage. Incidentally, some fragmentation schemes, such as horizontal, vertical, and hybrid, are widely used for DDBS design. At the same time, data allocation could not be done without first physically fragmenting the data because the fragmentation process is the foundation of the DDBS design. Extensive research have been conducted to develop effective solutions for DDBS design problems. But the great majority of them barely consider the RDDBS's initial design. Therefore, this work aims at proposing a clustering-based horizontal fragmentation and allocation technique to handle both the early and late stages of the DDBS design. To ensure that each operation flows into the next without any increase in complexity, fragmentation and allocation are done simultaneously. With this approach, the main goals are to minimize communication expenses, response time, and irrelevant data access. Most importantly, it has been observed that the proposed approach may effectively expand RDDBS performance by simultaneously fragmenting and assigning various relations. Through simulations and experiments on synthetic and real databases, we demonstrate the viability of our strategy and how it considerably lowers communication costs for typical access patterns at both the early and late stages of design.

Keywords: Database, Relational DDBS, Fragmentation, Clustering, Replication, Allocation

Introduction

Building an effective distributed database is more important than ever as computer networks and information technology improve. However, creating a distributed database is a highly challenging undertaking because there are so many geographically scattered sites and database relations. Reducing the communication costs and accelerating the query responses must also be considered [1, 2]. Particularly speaking, as database management technology develops, relational distributed database systems (RDDBSs) have become an essential resource for managing the ever-expanding volume of data. However, due to the massive expansion in data amounts, RDDBS performance has recently encountered clear demands on data management. Consequently, in an endeavor to produce the optimal design, the literature keeps presenting DDBS architectures. Overall, a distributed database can be created using three main techniques: fragmentation, data allocation, and replication. It is noteworthy that these techniques are routinely employed separately and are rarely combined. Various allocation options are employed, regardless of the fragmentation process or replication technique. In contrast, some fragmentation approaches do not consider allocation or replication procedures [3–5].

In fact, fragmentation is essential to databases because it enables a suitable design in distributed environments, which has benefits in terms of the execution costs for read and write operations. Technically speaking, fragmentation in a distributed database is typically quite advantageous in terms of utilization, dependability, and efficiency. Since the distributed database is divided into separate replica partitions, or "fragments," handling fragmented database replication presents administrators with a difficult problem. Replication, on the other hand, is one of the methods for managing data because it enhances its accessibility and dependability. Nevertheless, the amount of different data is growing quickly because technology is generally accessible and inexpensive. The problem arises when the transactions only function with a portion of the database and not the entire database [6].

Despite all of these issues, research on communication costs and response times is still a top priority, especially with the current exponential advancements in database and network technologies. There are a number of important factors that may justify this intense focus on DDBS design challenges. One of these factors—possibly the most important—is that response times and communication costs form the basis for DDBS performance, and their relationship to performance is completely modifiable. In the sense that DDBS performance has increased whenever communication costs and response times have decreased [7]. In other words, the performance of DDBS would benefit from the significant reduction of these expenditures. If data localization maximization and irrelevant remote data access minimization are successful, these costs can be greatly diminished. This premise is absolutely satisfied if the most highly correlated, "accessed together," tuples or attributes are intelligently aggregated into comparable groups and carefully placed in their most in-need sites. In fact, this is the main justification for using the hierarchical clustering method (HC), which performs better on tuples than attributes. The primary function of HC is to serve as the foundation for the efficient fragmentation of RDDBS.

One originality of the approach we propose is that, to best of our knowledge, no previous study utilizing HC has been recognized to fragment the “relational” RDDBS pattern that we have in our work. Our experimental findings show that HC facilitates the development of an efficient design by simplifying the allocation of tuples quickly and effectively. On both synthetized and real databases, the results demonstrate that HC supports the formation of an effective horizontal fragmentation, simplifying the equitable and effective distribution of tuples among network nodes, which is the main objective of this paper. To put it another way, HC aims to create data fragments through the clustering process, which are subsequently assemble together the strongly linked data in each fragment, as given in the drawn below example. The following is a list of the paper’s contributions in brief:

1. Developing a five-step horizontal fragmentation and allocation technique based on clustering that doesn't require dataset statistics, empirical findings, or query frequencies (at least in the early design stages). This characteristic makes the proposed approach useful for both the early and/or later stages of DDBS design, especially when statistics or even data access patterns are insufficient or unavailable.
2. The effective simultaneous execution of data fragmentation and allocation together ensures that processes flow into one another without introducing undue complexity. Contrarily, due to the complexity of the total problem and the simultaneous consideration of both difficulties, most previous research attempted to address either fragmentation or allocation separately.
3. Introducing a data distribution model that is based on the knapsack method. The allocation procedure is done in two phases: the early phase and the later phase. Three different scenarios—complete replication, partial replication, and no replication—are run for each phase. To discover the most suitable situation for constructing DDBS, these scenarios are drawn. The ideal scenario would therefore be included in the layout of the proposed approach.
4. Presenting a straightforward but efficient way to further fragment and allocate numerous relations at once, employing the notion of "row group," which is based on the established relationships between relations.
5. In closing, a thorough investigation (both internal and external) is made to show the practicality of the proposed approach. According to our evaluation of the previous studies, no horizontal fragmentation and allocation technique has ever been found (mathematically and practically at the same time) as a comprehensive, efficient, and dependable solution in RDDBS design. The proposed methodology is therefore expected to compete as a promising approach (and all-encompassing), with the expectation that it will significantly increase static and dynamic DDBS throughputs and productivity.

This paper is structured as follows: In "Related work" section, the earlier works are critically explored. "Proposed methodology" section presents the proposed methodology, including the technique's heuristics and architecture, fragmentation and allocation cost models, and site clustering. In "Multi-relations fragmentation and allocation" section, the proposed method to fragment and allocate multiple relations is introduced. "Results and discussion" and "Performance evaluation and discussion" sections summarize the results obtained and provide brief discussions and performance evaluations, respectively. Finally, in "Conclusions and prospective" section, the conclusions and prospects are briefly displayed.

Related work

Recent spikes in information demand make it imperative to adopt effective distributed database design strategies to boost its performance. In relational databases, strategies like data fragmentation, allocation, and replication are frequently employed to promote relational DDBS performance. A survey of the literature on these strategies is provided in this section. To address such problems of data distribution and partitioning, numerous techniques and algorithms have been developed. These techniques fall into heuristic

[6, 8, 9], meta-heuristic [10–16], affinity, or min-term-predicate based [8, 17–20], or clustering-based ones [20–25].

For example, to allocate and reproduce fragments optimally, either by optimizing access frequency or lowering transmission cost, an enhanced VAM mathematical model was created in [9]. By assigning fragments to locations with higher access frequencies, the technique aimed to limit fragment migration across the network. Another important constraint taken into account here is the storage capacity of the sites. The effectiveness and ideal behavior of the suggested method were demonstrated using the successful retrieval ratio of pieces. It utilized 3 fragments and 4 sites, and the SRR it acquired for allocation and replication—0.26 for each—was higher than that of comparable works. Replication boosted the retrieval ratio, according to experiments. However, the dataset used for evaluation was very small, making this technique unreliable. In the same page, database fragmentation was presented in [6], and the results demonstrated that managing databases with fragmentation would substantially aid a transaction to deal with the specified piece of the database rather than the complete database.

Meanwhile, in [17], 83 works on data fragmentation and replication approaches were analyzed and categorized. The authors presented an examination of various approaches [17] for database fragmentation, allocation, and replication in their paper [18]. The technique that was chosen in the analysis stage is then adopted by a Web application called FRAGMENT (based on a cost model which offered a fragmentation and replication method) was applied to a cloud environment. This work illustrated an issue with fragmentation methods known as overlapping fragments and offered an algorithm with a solution. The predicates produced by this technique define each fragment in a distributed context. In a follow-up, [19] came to present a new algorithm which was part of a strategy to additionally do allocation and replication. The enhancement entailed ranking the predicates according to cost and obtaining each predicate in this way while taking into account the possibility of fragment overlap. Investigation was made to show that the improved approach significantly decreased query response time.

In the same line, a method was presented in [9] for determining the number of copies of a fragment (benefit of degree of replication) that should be placed on various locations. The benefit of replication was continuously growing if there were two to four clones. This improved data accessibility, boosted system dependability, lightened the burden on the network, and facilitated parallelism. Nevertheless, adding more duplicates was proportionately less advantageous. Using evolutionary techniques, on the other hand, a new replicated data allocation strategy was proposed in [13]. The suggested method was created using simplified biogeography-based optimization (SBBO). The system's overall performance was improved while the total processing cost of a query was decreased using an SBBO-based method. The performance of the BBO and GA based approaches was compared to the results from the SBBO based approach. Furthermore, as a follow-up, Singh et al. developed a novel method for non-redundant data allocation in distributed database design [11]. The proposed approach assigned the data based on Simplified Biogeography Based Optimization (Simplified-BBO). Simplified-BBO based approach performance was compared with GA and BBO based approaches. This method assisted in lowering the cost of data communication during query execution, which improved the overall functionality of distributed database systems.

For the purpose of resolving the data allocation problem, Lotfi et al. suggested a novel hybrid solution based on the Simulated Annealing Algorithm (SA) and Variable Neighborhood Search (VNS) mechanism [12]. The suggested hybrid technique incorporates VNS mechanisms into the SA method to improve performance. Technically, this method (VNSA) explored the search space via SA employed neighborhood search mechanisms in order to uncover more promising sections of the search space. Furthermore, both approaches traversed the search space more quickly than population-based ones because they are single solution-based. The outcomes showed that VNSA outperformed its rivals in the majority of test difficulties. In order to tackle fragmented database replication, the study in [14] suggested a novel algorithm called Binary Vote Assignment on Grid Quorum with Association Rule (BVAGQ-AR). The BVAGQ-AR algorithm was able to divide the database into separate chunks.

In order to accomplish both horizontal and vertical fragmentation, Ahmed et al. introduced a hybrid data fragmentation model using an advanced optimization-based clustering methodology [15]. The Killer Whale Optimization (KWO) and Genetic Algorithms (GA) were combined with the common subspace clustering technique to create the proposed model, known as Genetic Killer Whale Optimization based Clustering (GKW, OC). The tuples and attributes were fragmented using an optimization-based hybrid fragmentation model. In the same page, using clustering algorithms, one of the best methods for horizontal fragmentation was constructed in [8]. Fragmentation, allocation, and clustering were all artfully merged into a single efficient technique, resulting in a significant answer for the improvement of DDBS production. The results of the experiments were almost exclusively noted as being in favor of DDBS performance. According to tuple and attribute patterns, this approach [24, 25] generated data fragments using subspace clustering algorithm, which was then used to group together strongly associated data. Comparing this clustering-based approach to the fragments selected at design time using pre-known statistics showed that it was beneficial in cutting down database access time.

In order to reduce the overall transmission costs of each site-fragment dependency and each inner-fragment dependency, the study in [26] provided a greedy method. Utilizing some DAP problems, an experimental research was conducted to gauge the effectiveness of the suggested strategy. According to author's findings, the proposed approach had improved quality in terms of execution time and overall cost. In the same vein, our work, therefore, comes to enrich the RDDBS literature with a novel strategy that explores both factors (communication costs and response time) in a single work. The proposed approach seeks to effectively address problems at both the early and late stages of design, as well as enhance the performance of both dynamic and static RDDBS, according to experimental findings and performance evaluation.

Proposed methodology

Motivations

Several methods have been developed to promote RDDBS performance through perfecting the RDDBS's design. Among the most popular and successful techniques are data fragmentation, data allocation and replication, and site clustering; otherwise, DDBS design and rendering would be prohibitively expensive. Therefore, incorporating all of these strategies into a single work will undoubtedly result in a considerable boost to the DDBS's influence. As a result, all of these aspects are combined in our work as a whole.

Furthermore, based on our investigation of the majority of previous works, a difficulty has been observed to analyze data patterns and features in large databases with these works as their fragmentation strategies are essentially dependent on query frequency and other data statistics, especially at the beginning of RDDBS building. Therefore, the goal of this study is to add to the body of DDBS literature with a fresh, adaptable design that would guard against data loss and inappropriate grouping of large datasets (at least during the first design phase). Without requiring dataset statistics, empirical findings, or even query frequency in the initial stage, this study's five-step horizontal fragmentation and allocation approach is developed based on clustering. Specifically, this aspect makes the method we propose viable for use at both the initial and/or later stages of DDBS design where statistics or even data access patterns were insufficient or unavailable. However, at later stages of design, query frequency has proven essential for carrying out the appropriate re-allocation operation. Finally, it is important to note that we reviewed a good number of the recently proposed methods in "Related work" section, and our work has been greatly influenced by and compared to [8, 9, 19].

Heuristics

In the DDBS, it is well-known that before executing data allocation and fragmentation, the designer must first gather all necessary information. The number of network sites, communication expenses, and database information essentially represented in relations' metadata. The quantity of attributes, attribute cardinalities, attribute sizes, tuple sizes, predicates used in queries for each relation, the frequencies of queries, attribute usage, and predicate usage at subsequent design stages are only a few examples of such information. DDBS horizontal fragmentation and allocation can then be achieved with sophistication using this information.

The fragmentation heuristics composes of five steps that make up the fragmentation process, and they are as follows:

Stage (1): Locating and removing the list of potential query predicates. If there are no duplicate predicate sets, the predicate set identification will be simple. However, this replicated set would be eliminated if any cluster discovered the replication for the predicate set. This step helps prevent fragment redundancy when fragments produced in the subsequent stage (disjoint-ness properly maintained).

Stage (2): Based on the currently filtered list of predicates, the query clustering procedure is initiated. Based on the idea of the least difference value (LDV) for each query pair inside the target cluster, a new centroid is chosen for each cluster at each loop. The complexity and execution time of the proposed technique has been significantly reduced by closely adhering to this step. It's important to note that query Qi will set up a new cluster as the cluster's centroid on its own if it has the same LDV with more than one centroid during the clustering process.

Stage (3): Producing all possible predicate set combinations individually for each cluster.

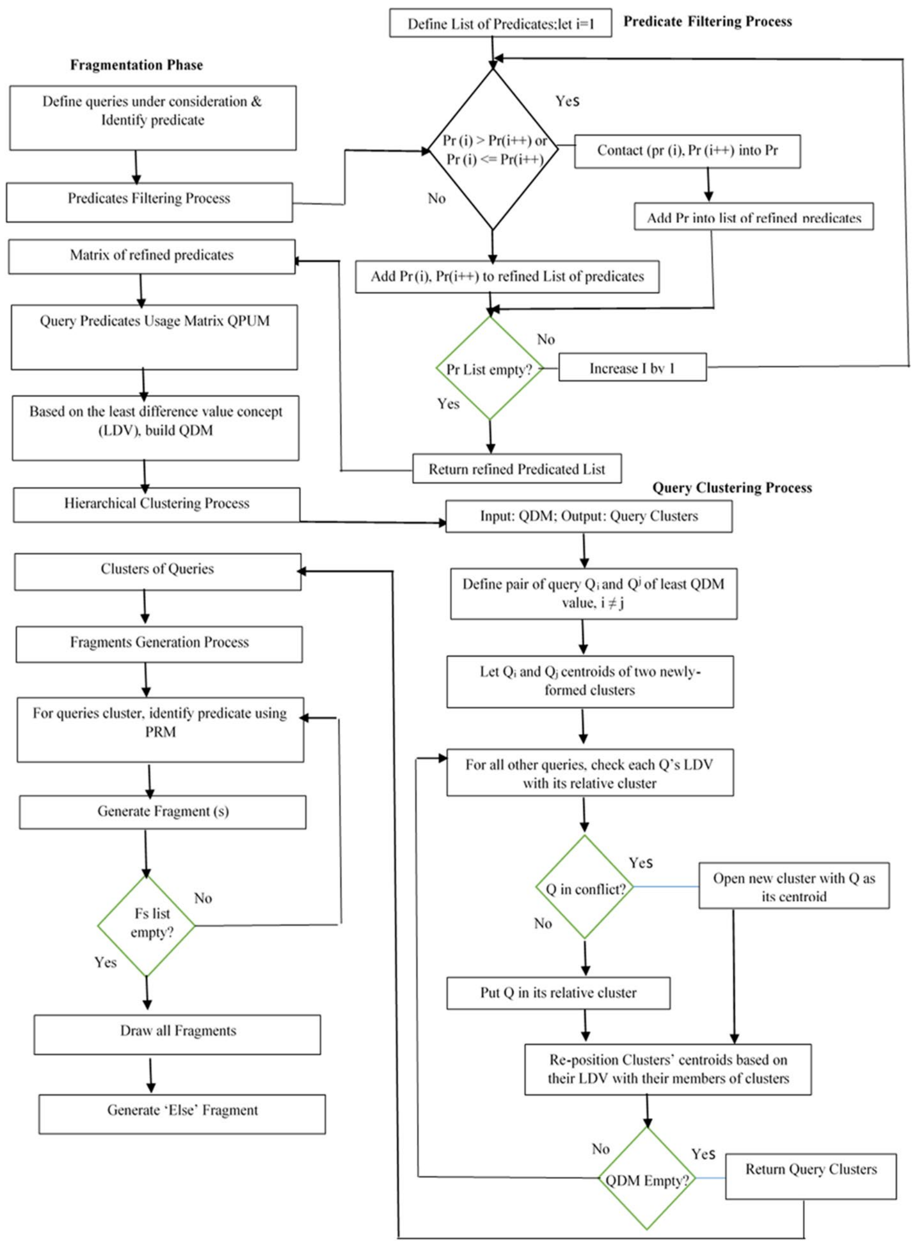

Fragmentation Phase
Define queries under consideration & Identify predicate
Predicates Filtering Process
Matrix of refined predicates
Query Predicates Usage Matrix QPUM
Based on the least difference value concept (LDV), build QDM
Hierarchical Clustering Process
Clusters of Queries
Fragments Generation Process
For queries cluster, identify predicate using PRM
Generate Fragment (s)
Fs list empty?
No
Yes
Draw all Fragments
Generate 'Else' Fragment
Define List of Predicates:let i=1
Predicate Filtering Process
Pr (i) > Pr(i++) or Pr (i) <= Pr(i++)
Yes
No
Contact (pr (i), Pr (i++) into Pr
Add Pr into list of refined predicates
Add Pr(i), Pr(i++) to refined List of predicates
Pr List empty?
No
Yes
Increase I by 1
Return refined Predicated List
Query Clustering Process
Input: QDM; Output: Query Clusters
Define pair of query Qi and Qj of least QDM value, i ≠ j
Let Qi and Qj centroids of two newly-formed clusters
For all other queries, check each Q's LDV with its relative cluster
Q in conflict?
Yes
No
Open new cluster with Q as its centroid
Put Q in its relative cluster
Re-position Clusters' centroids based on their LDV with their members of clusters
QDM Empty?
No
Yes
Return Query Clusters

Fig. 1 Five-step data fragmentation process

Stage (4): Generating the fragments (Fs) based on the predicate combinations created in stage 3. Generating the fragment (Fs) under the condition that neither the predicate set nor its opposite set is exist in the same cluster. This step eliminates the emptiness of the fragments (the lossless-ness property is maintained).

Stage (5): Produce the 'Else F' fragment, R/Fs., "the final fragment". All the excluded information in those currently formed fragments is set to be in the "else F" (Fs). In other words, R/Fs, where R represents the entire relation, could be used to mathematically express Else F. This action favors keeping the minimal predicate set property. Finally, Fig. 1 depicts the entire data fragmentation process.

Fragmentation procedure: problem formulation

Provided that we have a set of “A” attributes $A=\{A_1, A_2, \ldots, A_n\}$ required by set of queries $Q=\{Q_1, Q_2, \ldots, Q_q\}$. $\forall$ Q_i, $\exists$ predicates $P=\{P_1, P_2, \ldots, P_m\} \in$ attribute A_j. $\forall$ Predicate pair P_i and P_j, $\ni A_h$, where $h=\{1, \ldots, n\}$, and both predicates are identified in one of three states. $\forall P_i \supset P_J$ as $P_i \in P_j$, or $P_I \supset P_i$ as $P_j \in P_i$. The final state is, $\forall$ P, F_k is defined independently, where $k=\{1, \ldots, f\}$, and f is the number of fragments. Using the filtering procedure, all P_{hs} are drawn to be fed into HC. These predicates are fed into HC to be grouped into C_c clusters $\{C_1, C_2 \ldots, C_c\}$. Then, the HC algorithm would work on the predicate clusters. These clusters would represent the non-overlapping final data fragments, $Fs=\{F_1, F_2, \ldots, F_f\}$.

Hierarchical clustering (HC)

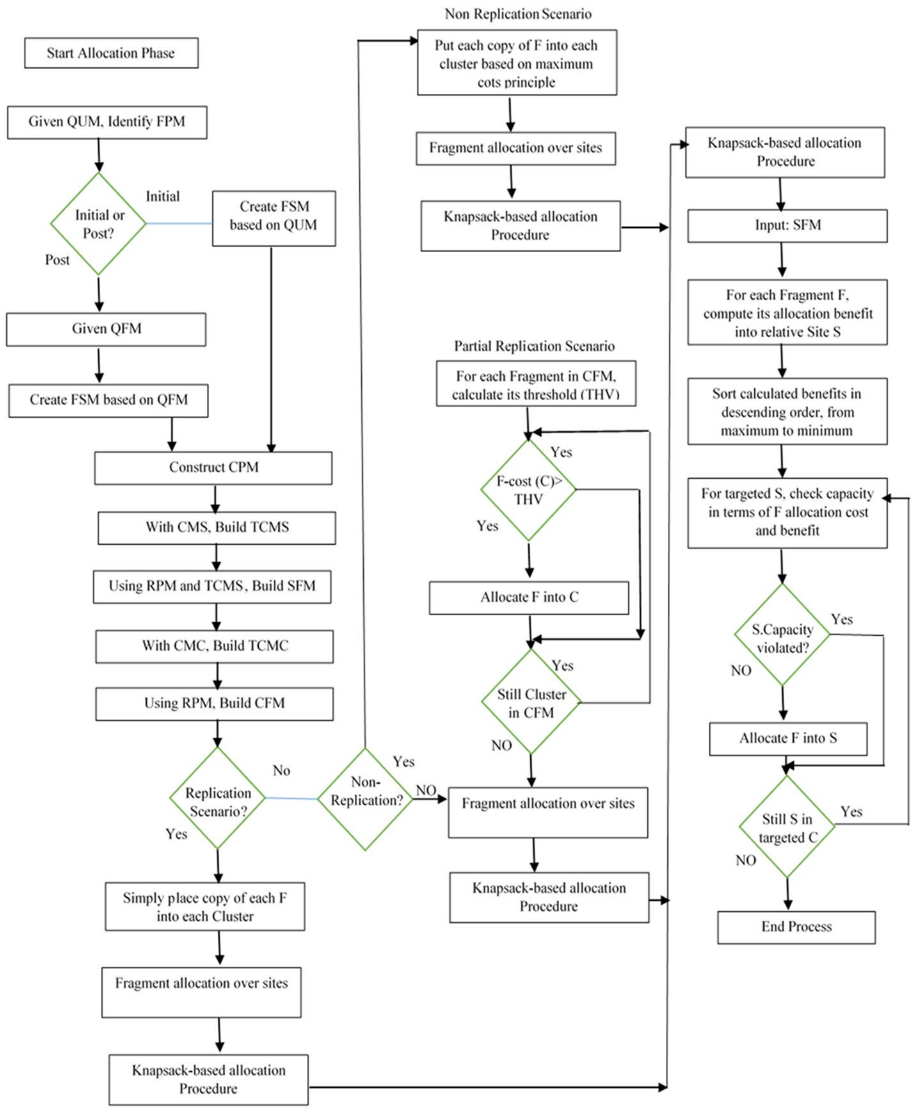
A single, exhaustive cluster is located at the top, and singleton clusters of distinct items are found at the bottom using the predicate sets $P=\{P_1, P_2, \ldots, P_m\}$. The next move is to merge each cluster pair in each middle level from the grade below or splitting the cluster from the grade above. The following steps are performed in this order: (1) calculate the similarity (using Euclidean distance) between all pairs of predicates, i.e., create a similarity matrix whose IJth leaf in the clustering tree represents the similarity between the Ith and Jth predicates; (2) incorporate those that are most similar (closest) predicates; (3) update the similarity matrix in each clustering incrementally to include the new pairwise similarity between the fresh predicates and the original ones; and (4) repeat steps (2) and (3) until the desired clusters are found (see Tables 14, 15). The final fragments are produced using these clusters (see Table 16). Figure 2 simply depicts the HC steps in high-level abstract.

```
HC Algorithm
Input: Predicate set P, to be clustered, the Euclidean distance to find the similarity: $Euc(P_i, P_j) \rightarrow [0,1]$,
        c=|PC|, where PC stands for the resulted predicate clusters, so c is the number of clusters.
Output: The set of hierarchical clusters $PC=\{pc_1, .., pc_c\}$
Begin
PC={}
c=0
While P list is not empty
$(p_{I*}, p_{j*})=\text{argmax}(P_i, P_j)\,[\text{sim}(P_i, P_j)]$;
$PCnew= =P_{I*} \cup P_{J*}$;
Update P and PC
$P=P- \{P_{I*} \cup P_{J*}\} \cup PC_{new}$;
$PC=PC+ PC_{new}$
C++
End while
End
```

Algorithm 1 The High-Level Abstract of Hierarchical Clustering Algorithm

Allocation and replication heuristics

The knapsack algorithm is used to maximize each site’s capacity (and thus each cluster). The allocation of data fragments must be done by rows rather than by attributes to ensure that each fragment is placed in the best possible location, which reduces the allocation process’s time.



Start Allocation Phase
Given QUM, Identify FPM
Initial or Post?
Initial
Post
Create FSM based on QUM
Given QFM
Create FSM based on QFM
Construct CPM
With CMS, Build TCMS
Using RPM and TCMS, Build SFM
With CMC, Build TCMC
Using RPM, Build CFM
Replication Scenario?
No
Yes
Non-Replication?
Yes
NO
Simply place copy of each F into each Cluster
Fragment allocation over sites
Knapsack-based allocation Procedure
Non Replication Scenario
Put each copy of F into each cluster based on maximum cots principle
Fragment allocation over sites
Knapsack-based allocation Procedure
Partial Replication Scenario
For each Fragment in CFM, calculate its threshold (THV)
F-cost (C)> THV
Yes
Yes
Allocate F into C
Still Cluster in CFM
Yes
NO
Fragment allocation over sites
Knapsack-based allocation Procedure
Knapsack-based allocation Procedure
Input: SFM
For each Fragment F, compute its allocation benefit into relative Site S
Sort calculated benefits in descending order, from maximum to minimum
For targeted S, check capacity in terms of F allocation cost and benefit
S.Capacity violated?
Yes
NO
Allocate F into S
Still S in targeted C
Yes
NO
End Process

Fig. 2 Data allocation process

Full replication-based allocation scenario

Phase 1: As fragment replication is implemented in the first phase, each fragment would be assigned to every cluster. The goal of this replication is to make data more local, which will significantly reduce response times and transmission costs when dealing with distributed queries.

Non-replication-based allocation scenario

Phase 1: Each fragment is assigned to the cluster with the highest access cost. Fragments are distributed among clusters according to the total access cost of each site's cluster (TCPM). As the distributed query is processed, such allocation process is certain to reduce the communication costs.

Phase (2): Both full replication and non-replication scenarios: The allocation criterion is the net cost of each site, S_j, to access all of the fragment F_i's attributes. Therefore, it is necessary to exactly compute the total access cost ($TSFM_{ij}$) for each site to access all of these attributes. The SPM and CSM matrices are used to create the TSFM matrix (see Eq. 13). The site with the highest access value for a particular fragment is chosen as the prime candidate storage site for this pertinent fragment (see TSFM matrix),

Partial replication-based allocation scenario

Each fragment is assigned to a site in accordance with the pre-determined threshold. In turn, the threshold (in Eq. 1) is determined by averaging the costs associated with allocating resources across all network sites.

$$\begin{aligned}\mathrm{Thv}(\mathrm{F_i}) =& (\text{Cots of allocation } \mathrm{F_i}(\mathrm{S_1}) + \text{Cots of allocation } \mathrm{F_i}(\mathrm{S_2}) \\ & + \cdots + \text{Cots of allocation } \mathrm{F_i}(\mathrm{S_m}))/\mathrm{M}\end{aligned} \tag{1}$$

Network clusters and sites will engage in fierce rivalry for the fragment allocation after each fragment has reached its threshold. Each Fi in this competition is assigned to the cluster or site whose access cost exceeds the set threshold. Furthermore, even if the intended site still has access costs above the threshold, the number of allocation times for each fragment shouldn't be greater than $|M/2|$, where M is the total number of sites.

Since priority is given to clusters/sites of most benefit (to store the relative fragment) based on the Knapsack algorithm mechanism, sites are configured to fiercely compete for the site with the maximum cost, which is the most likely place to store the targeted fragment. It is worth emphasizing that, for all scenarios, the allocation process is done while the sites' (clusters') constraints are maintained. If one or more constraints are violated under any circumstances, the underlying fragment is automatically assigned to the site whose cost value is the next. The whole process of data allocation is drawn in Fig. 2.

Fragmentation and allocation cost models

Table 1 presents all the notations used in this work.

Objective function

The objective function is configured to implicitly describe the number of disc accesses (i.e., transmission costs), which would be the key performance indicator for the proposed strategy.

$$TC = Min\left(\sum_{k=1}^{q}\sum_{j=1}^{m} QFM_{kf} * XF_{kj} * COM_{s_iS_j} * \left\{Sel(Q_k) * size(F_k)\right\}\right) \tag{2}$$

Fragmentation and allocation cost functions

Based on the considered queries and the results of the predicate filtering process (i.e., Table 17), a matrix of query predicate usage (QPUM) is drawn (i.e., Table 18). Each

Table 1 Notation used

nf	The total number of fragments
m	The number of sites
Q	The number of queries under consideration
pr	The number of predicates
CN	The number of clusters of sites
CQ	The number of clusters of queries
i	Fragment index, I = {1,…, n}
j	Site index, j = {1,…, m}
K	Query index, h = {1,…, k}
cni	Sites' cluster index Cn_i = {1, …, cn}
cnq	Query's cluster index Cn_q = {1, …, cq}
C_i	Storage capacity of site S_i
V	The attribute size
Size (F_k)	Size of the kth fragment
Sel (Q_k)	The percentage rate of rows returned by the kth query
QFM_{ij}	The frequency of the ith query over site S_j
CCM_{ij}	The communication cost unit between cluster C_i and cluster C_j
CMS_{ij}	The communication cost unit between site S_i and site S_j, measured in "ms/byte" units, and bandwidths fall in; 64 kbps, 128 kbps and 512 kbps
COM_{ij}	CMS_{ij} cost, if S_i and S_j are at the same cluster; otherwise, CCM_{ij} cost
LNF	The lowest number of fragments allowed at each site
UNF	Upper number of fragments allowed at each site
TC	The total cost of data transmission as distributed query processed
XF_{kj}	1, if fragment F_k is placed in S_j; 0, otherwise
ACvalue	ACvalue is a piled-up value that refers to the total access cost produced by each query to access its relative fragment
FPM	Matrix in which each FPM_{ij} is 1, F_i contains P_j; 0, else
FSM	Matrix in which each FSM_{ij} is ACvalue, S_j requires queries of F_i; 0, otherwise
SPM	Site Predicate Matrix
TSPM	The total cost of SPM which is grouped based on the container clusters of sites
TCCM	Matrix of Total costs of communication between clusters
TCSM	Matrix of Total costs of communication between sites
QUM	Query Usage Matrix; 1, Q_i required by S_j; 0, otherwise
QDM	Query Difference Matrix, so each QDM_{ij} is the difference value between Q_i and Q_j

$QPUM_{ij}$ indicates whether query Q_k is approaching the predicate P_i. Moreover, suppose that the QUM is given by the database administrator (DBA) so that each QUM_{ij} refers to whether each query is released from its relevant site. Thus, the horizontal fragmentation and (initial and post) allocation are made based on these requirements using the following cost functions:

$$P(Q_k) = \sum_{k=1}^{q} \sum_{i=1}^{pr} QPUM_{ki} \tag{3}$$

$$\text{Similarity}\,(\text{Q}_{\text{k1}},\text{Q}_{\text{k2}}) = \sum_{\text{k1}=1}^{\text{q}}\sum_{\text{k2}=1}^{\text{q}} (\text{1-dif(P((Q}_{\text{k1}}),\text{P(Q}_{\text{k2}})), \tag{4}$$

where P(Q) means the numerical pattern drawn as per the contained predicates in each query.

$$\text{QDM}_{\text{k1k2}} = \sum_{\text{k1}=1}^{\text{q}}\sum_{\text{k2}=1}^{\text{q}} \text{Hamming-distance}\,(\text{Q}_{\text{k1}},\text{Q}_{\text{k2}}) \tag{5}$$

The QDM matrix is fed into the AHC algorithm to generate the required fragments (i.e., Tables 27, 28, 29, 30, 31). Bearing in mind that the difference of (Q_i, Q_{i+1}) is kept at a minimum. Then, using FPM and QUM matrices (i.e., Tables 34, 35), the allocation process would begin as follows;

$$SPM = \sum_{i=1}^{p}\sum_{j=1}^{m} FPM_{ij} * FSM_{jk} \tag{6}$$

$$TSFM = \sum_{i=1}^{pr}\sum_{j=1}^{m}\sum_{k=1}^{m} SPM_{prk} * CSM_{kj} \tag{7}$$

$$CPM = \sum_{cni=1}^{cn}\sum_{i=1}^{pr}\sum_{j=1}^{m} accumulate\,(SPM_{ji}) \tag{8}$$

$$TCPM = \sum_{i=1}^{pr}\sum_{cni=1}^{cn}\sum_{k=1}^{cn} CPM_{prk} * CCM_{kj} \tag{9}$$

For fragmentation, Eq. (3) represents the numerical pattern of each query in terms of rows of (0, 1) based on the filtering process's results, allowing patterns to be drawn precisely in the QPUM matrix. Most importantly, Eq. (4) is used to calculate the similarity between each pair of patterns. Finally, Eq. (5) is used to find the intended similarity in order for the clustering process to begin. It is worth stressing that Eqs. (4) and (5) are performed simultaneously to produce a query difference matrix (QDM) matrix.

For initial allocation

The FSM matrix is built based on the QUM matrix with the help of the outcomes of the filtering process's predicates. Each FSM value points to the number of times each site S_j accesses F_i based on the constituent queries of each fragment (tacitly counted based on the refined predicates). While Eq. (6) produces the site-predicate matrix (SPM) in which each site provided its actual access for each predicate, Eq. (7) calculates the total communication costs that each site had to incur to approach the queries remotely. The matrix of the total costs of each cluster to access the relative predicate, on the other hand, is determined by adding the access costs of that cluster (in terms of its contained sites) using SPM [tacitly expressed by the query in Eq. (8)]. Finally, the total communication costs each cluster incurred to remotely access queries are calculated using Eq. (9).

Table 2 Site constraints

Site	S_1	S_2	S_3	S_4	S_5	S_6
Capacity (MB)	28,500	42,000	26,000	39,000	45,000	33,000
LNF	1	1	1	1	1	1
UNF	12	14	11	7	10	14

For post allocation

Given that the FSM is created using the Query Frequency Matrix (QFM) rather than the Query Utility Matrix (QUM), all Eqs. (5–8) are employed. DBA allegedly provided QFM at a later stage in the design process. Finally, it is necessary to maintain the following restrictions throughout the data allocation process:

$$\sum_{k=1}^{cq} Size(F_k) \leq Capacity\left(S_j\right), \quad \forall j = 1, \ldots, m. \tag{10}$$

$$LNF_i \leq \sum_{i=1}^{n} X_{ij} \leq UNF_i, \quad \forall j = 1, \ldots, m. \tag{11}$$

$$X_{ij} \in (0, 1), \tag{12}$$

Constraint (11) ensures that the overall amount of fragments assigned to one site would not be greater than that site's disc capacity, as shown in Table 2. By this restriction, it is stated that the number of fragments assigned was between LNF and UNF (11). The final restriction is the binary constraint on the decision variable (12). The limitations for each site are shown in Table 2, along with their capacity (in Mbytes) and the minimum and maximum number of fragments that can be kept there.

Site clustering

Sites under consideration are clustered using hierarchical clustering. In the event that any site has the same cost with more than one cluster, the deciding factor utilized to move a site (let's say, S_j) to its appropriate cluster is the average of communication costs over each site. On the other hand, the first two clusters are consistently created by choosing a pair of sites at random when their transmission costs are the highest [28, 29]. This step is taken as a result of running numerous site grouping trials and discovering that many pairs would qualify as clusters on their own at the initial stage, which would contribute to maintaining the cluster number at a minimum. The rest of the clusters are grouped using the agglomerative hierarchical clustering. It goes without saying that the fragment allocation procedure must consider the communication costs inside and across clusters, especially in the non-replication situation.

It is also worth noting that during experiments, site clustering is carefully done so that the actual Transmission Cost (TC) is accurately reflected. In turn, TC has been observed to be gradually increasing as the traffic load and data exchange within the network increased aggressively. So, it was assumed that each cluster (CS) had a centroid (representative) that served to approach each other cluster when distributed queries were being handled. This assumption is made to reflect the actual communication between clusters of sites. As a result, two types of communication costs were anticipated: (1)

Inside (CS): Any site that can reach out to remote clusters via its owner's centroid incurs extra communication costs from that site to centroid. Consequently, a precise mathematical method to minimize these costs is strongly recommended. (2) Outside (CS): Each CS is able to communicate with its peers directly through a representative.

Multi-relations fragmentation and allocation

formulation

Given a table set $T = \{ T_1, T_2, \ldots, T_n\}$ as a relational database. One table, T_i is assumed to be defined (by DBA) as a primary table (PT), and the others are defined as its follower tables (FT). T_i is set to be fragmented horizontally using our proposed approach into $\{F_{1i}, F_{2i}, F_{3i}, \ldots, F_{fi}\}$. $\forall$ F_{ki} is allocated (and may be replicated) into several sites $(S_1, S_2, \ldots, S_m)$. $\forall$ F_{ki}, $\ni F_{hw}$ where $h \in \{T/PT\}$ and $w \in \{T/FT\}$. $\forall$ Tuple $t_1, t_2, \ldots, t_b \in F_{ik}$, $\exists\, t_1, t_2, \ldots, t_h \in FT$. The task is to place all tuples $\in$ FT with their relative in F_{ik} in the same site where F_{ik} is already allocated. In other words, the task is to fragment FT and place fragments into sites in which their relatives in F_{ik} of PT are already exist. Algorithm 2 provides the proposed algorithm used to fragment multi relations based on the primary relations.

To further demonstrate the problem, suppose we have Emp, Dept, and Proj tables, with Emp defined as the primary table (PT) and Dept and Proj defined as their follower tables (FT). PT is fragmented into $emp_1, emp_2, \ldots, emp_f$. $\forall$ emp_f, $\exists\, t_1, t_2, \ldots, t_b \in emp_f$. $\forall$ $t_i \in emp_f$, $\exists\, t_1, t_2, \ldots, t_b \in$ Dept and $t_1, t_2, \ldots, t_b \in$ Proj. The task is to extract these tuples in Dept and Proj that have PK and FK relationships with Emp tuples so that each $F \in emp$ is placed in Sj along with its Dept and Proj follower tuples.

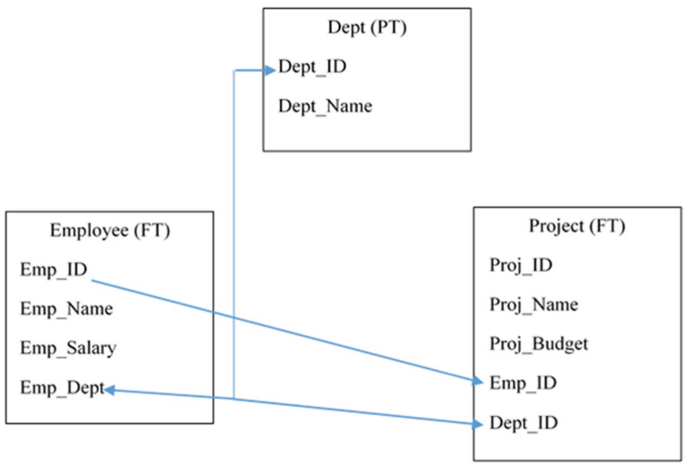


Fig. 3 Relational database diagram

Table 3 Dept dataset

Dept-Id	Dept-name
1	A
2	B
3	C
4	D

Table 4 Employee dataset

Emp-ID	Emp-Name	Emp.Salary	Emp.Dept
300	Ali	1600	1
306	Bush	1600	1
113	Ali	2900	2
118	Jasmine	3200	2
202	Coddy	1300	1
200	Beng	2500	3
109	Brad	2500	4
188	Obama	2900	4
278	Bush	1300	1
367	Jasmine	2700	2
121	Coddy	1200	–
122	Bin	2100	–

The proposed multi-relations fragmentation algorithm

```
Algorithm
Input: Primary Fragments List (PT); Follower Relations (FT); Relationships (PK- FKs)
Output: Primary Fragments List (PT); Follower Fragments List (F List)

1.  h=1;
2.  For I=1 to NF (PT)  // (number of fragments of primary relation (table), PT)
3.  {Frag = { }            // initiate empty fragment
4.   For j=1 to NFT (number of follower relations, tables, FT)
5.      {For k=1 to F[I].size    // to initiate pointer to tuples of each fragment primary
                // each tuple in PT.F[i] must scan all tuples of each follower relation FT
6.   {For w=1 to R[J].size    // initiate pointer to the follower relation
7.      {If F[I].Tuple[k].PK=R[j].Tuple[w].FK // define the relationship between PT and FT
                          // define linkage guarantees that the entire FT is fragmented as each tuple in PT
                               has at least one linked tuple in FT; PK is the primary Key, and FK is the foreign Key//
8.            Frag= Frag U R[j].Tuple[w]     // filling Frag with all tuples holds true on all predicates
9.      } for w
10.    } for k
11.  }// for j
12.     F[h] = Frag              // move frag tuples into fragment F as FT is being fragmented
13.     h++
14.   }//for I
```

Algorithm 2 **Primary Relation-Based Follower Relations FragmentationIllustrative example**

Suppose we have the diagram given in Fig. 3. The diagram exhibits the database consists of three relations. The Dept. relation (Table 3) has been identified as the primary relation and Employee (Table 4) and Project are defined as follower relations for Dept. The

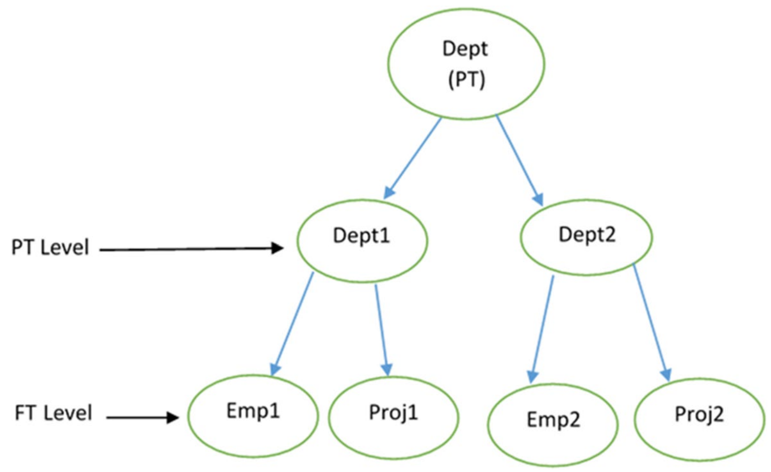


Fig. 4 Fragmentation results (RDDBS)

Table 5 Dept-Fragment (1)

Dept-Id	Dept-name
1	A
2	B

Table 6 Dept-Fragment (2)

Dept-Id	Dept-name
3	C
4	D

department ID defines the relationship between a department and its followers, and the employee ID defines the relationship between an employee and a project. Supposedly, DBS decides to appoint Dept as PT, so it is being split using the procedure drawn out in this paper. Employee and Project, as followers, are split using the proposed algorithm along with the help of defined relationships as drawn in Fig. 4 and the next example.

Assuming that each employee was appointed to one department and one or several projects. Similarly, each department was assigned to one or more projects. While it is the DBA's responsibility to specify which tables are PT and which are FT, Fig. 4 shows that Dept is the PT for both the Emp and Proj tables, and Emp is also the PT for the Proj table. Suppose we had the next queries running against both tables (Dept and Emp):

Q1: Select * from emp, dept where F-emp.id = dept.id and dept.name in ('A', 'B') and emp.salary > 2000;
Q2: Select * from emp, dept where F-emp.id = dept.id and and emp.salary > 1500;
Q3: Select * from emp, dept where F-emp.id = dept.id and Emp.name in ('Ali, 'Jasmine');
Q4: Select * from emp, dept where F-emp.id = dept.id and dept-id in (3, 4);

Table 7 Emp-Fragment (1)–On Dept-Fragment (1)

Emp-ID	Emp-Name	Emp.Salary	Emp.Dept
113	Ali	2900	2
118	Jasmine	3200	2
367	Jasmine	2700	2
300	Ali	1600	1
306	Bush	1600	1
113	Ali	2900	2
118	Jasmine	3200	2
202	Coddy	1300	1
278	Bush	1300	1
367	Jasmine	2700	2

Table 8 Emp-Fragment (2)–On Dept-Fragment (2)

Emp-ID	Emp-Name	Emp.Salary	Emp.Dept
200	Obama	2900	3
109	Bush	1300	4
188	Jasmine	2700	4

Table 9 Company database description

Attributes	Attribute's ID	Type	Length (bytes)	Value range
Personnel-ID	A1	Numerical	19	Integer values
Personnel-Name	A2	Categorical	40	Alphabet Letters
Personnel-Salary	A3	Categorical	6	1000–7000
Personnel-Rank	A4	Nominal	8	Manager—clerk—employee
Personnel-Sex	A5	Numerical	2	(0) or (1)
Personnel-Dept.	A6	Categorical	6	(10–20–30–40–50)

Moreover, data in both Tables (Dept and Emp) were given as follows;

Table 4 also included the employee dataset as a follower (FT) for the department table (PT) (Tables 5, 6).

On the other hand, regarding the FT fragmentation, on Depth-Fragment (1), Employee fragmentation was made simple by using relationship connection keys in "row groups," as shown in Tables 7, 8 and 9. For each PT. fragment, its follower tuples in each FT were being transferred to the same site. Moreover, if PT was replicated, its followers were also replicated in a number equal to the number of PT replicas. In fact, this is the major problem the RDDBS system has been suffering from. However, the proposed mechanism to tackle this issue is already given in "Multi-relations fragmentation and allocation" section.

However, if Dept-Fragment (1) was replicated in two sites, for example, Emp-Fragment (1) would be replicated in both sites as well.

Table 10 Company dataset tuples

A1	A2	A3	A4	A5	A6
111	Ali Mohammed	3500	Clerk	1	5
112	Hassan Mustaf	4000	Clerk	1	10
113	Adel Ahmed	2000	Employee	1	5
114	Jasmin Yaser	2200	Clerk	0	15
115	Salma Mohammed	1700	Employee	0	15
116	Jhon Bush	1250	Employee	1	20
117	Burney Mich	6000	Manager	1	5
118	Lilly Eriks	1800	Employee	0	10
119	Ali Ali	2100	Clerk	1	10
120	Yaser Twfik	2000	Employee	1	15
121	Nehmi Ahmed	1570	Employee	1	15
122	Mohsen Saleh	3200	Employee	1	20
123	Ebtesam Ismail	4500	Manager	0	10
124	Ibrahem Mohammed	4100	Clerk	1	15
125	Ibrahem Ali	1100	Employee	1	20
126	Ahmed Adel	2000	Employee	1	20
127	Mustafa Adel	1800	Employee	1	20
128	Robin Shobes	6600	Manger	0	15
129	Ted Mosbey	5100	Clerk	1	25
130	Don Michel	1500	Employee	1	25
131	Nasem Yasen	2200	Clerk	1	25
132	Asma Ahmed	2200	Clerk	0	30
133	Jasmin Saleh	1750	Employee	0	30
134	Salam Salam	1800	Employee	0	30
135	Natalia ali	2000	Employee	0	25
136	Tamer Hassan	4600	Manager	1	20
137	Faten Salem	3200	Clerk	0	25
138	Salem Ali	7000	Manager	1	25
139	Yaser Hussien	3000	Clerk	1	30
140	Hussien Ahmed	7000	Manager	1	30

Results and discussion

A C#-programmed software application is created to test the overall performance of the anticipated RDDBS and examine the behavior of the proposed approach. A 3.75 GHz Intel (R) Dual Core (TM) i5 CPU, 4 GB of main memory, and a 500 GB hard drive runs this application. With a six-site network, only the queries under consideration are needed, as was previously stated. A company dataset has been proposed (Table 10) in accordance with the description in Table 9 to conduct the illustrative experiments. Just for the first experiment which was limited to the retrieval-type queries against the six-attribute company dataset. The dataset was filled out with 30 records just for illustration purposes. For ease of computation and understanding, the attributes are given IDs as given in Table 9.

For the first problem, we considered eight queries as the most active and making up 40%–80% of all database requests against the company database.

Q1: Select A_1, A_2, A_5, A_6 from company where $A_6 < 10$ and $A_3 > 4000$;

Table 11 Query predicates set

Predicate number	Predicate content
P1	$A6 < 10$
P2	$A6 < 20$
P3	$A6 > 20$
P4	$50 > A6 > 30$
P5	$A6 \leq 15$
P6	$A6 > 40$
P7	$A3 > 4000$
P8	$A3 < 4000$
P9	$A3 = 4000$

Table 12 Predicates distribution over original queries

Query number	Predicate content
Q1	P1 and P7
Q2	P2 and P7
Q3	P3
Q4	P4 and P7
Q5	P5 and P8 and P9
Q6	P6 and P9
Q7	P8 and P1
Q8	P8 and P1

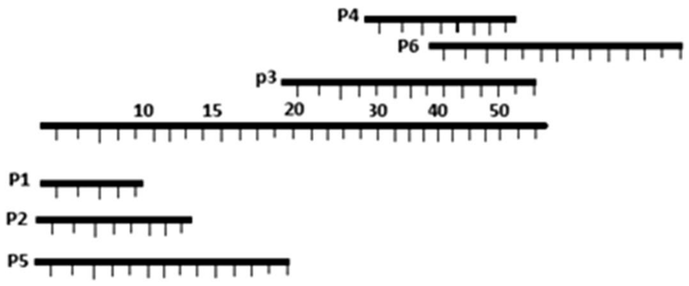


Fig. 5 A_6's attribute predicates filtering

Q2: Select A_3, A_5 from company where $A_6 < 20$ and $A_3 > 4000$;
Q3: Select A_2, A_4, A_5 from company where $A_6 > 20$;
Q4: Select A_1, A_3, A_6 from company where $50 > A6 > 30$ and $A3 > 4000$;
Q5: Select A_1, A_2, A_5 from company where $A6 \leq 15$ and $A3 < 4000$;
Q6: Select A_3, A_4, A_6 from company where $A6 > 40$ and $A3 = 4000$;
Q7: Select A_2, A_6 from company where $A3 < 4000$ and $A6 < 10$;
Q8: Select A_1, A_3, A_5, A_6 from company where $A3 < 4000$ and $A6 > 20$;

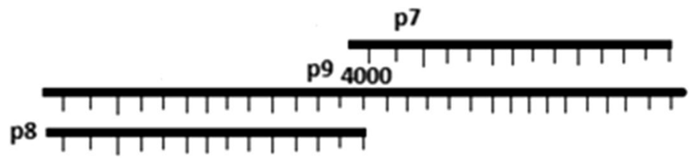


Fig. 6 A3's attribute predicates filtering

Table 13 Filtered predicate set

Predicate number	Predicate content
P1	A6 < 20
P2	A6 > 20
P3	A3 ≥ 4000
P4	A3 < 4000

Table 14 QPUM

Query/predicate	P1	P2	P3	P4
Q1	1	0	1	0
Q2	1	0	1	0
Q3	0	1	0	0
Q4	0	1	1	0
Q5	1	0	0	1
Q6	0	1	1	0
Q7	1	0	0	1
Q8	0	1	0	1

Now, employing the considered queries, their predicates are defined in Table 11. Table 12 shows the distribution of predicates based on Table 11 and their owner queries.

Predicate filtering process

The filtering was done for the A_6 attribute according to the line number that was originally drawn to compare with all of A's predicates (Fig. 5). In order to concurrently produce and combine as many of A's predicates as possible into individual predicates, the comparison was carefully made, resulting in a significantly smaller number of predicates. This action was taken to effectively remove the unnecessary predicates. It was clear when A_6's (Fig. 5) was addressed, the next actions were resulted: (1) P_2 comprised P_1 and P_5, therefore P_1 and P_5 were incorporated into P_2 to form one predicate; and (2) P_3, P_6, and P_4 were combined.

For the A_3 attribute (Fig. 6), the line number was also drawn to clarify the comparison process, which used as many A_3 predicates as possible to fuse into a single predicate.

Table 15 QDM

Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Q1	0	0	3	2	2	2	2	4
Q2	0	0	3	2	2	2	2	4
Q3	3	3	0	1	3	1	3	1
Q4	2	2	1	0	4	0	4	2
Q5	2	2	3	4	0	4	2	2
Q6	2	2	1	0	4	0	4	2
Q7	2	2	3	4	2	4	0	2
Q8	4	4	1	2	2	2	2	0

Table 16 QDM's final results

Query	Q1257	Q3468
Q1257	0	2
Q3468	2	0

From Fig. 6, it was manifestly clear that P_7 and P_9 could be integrated into one predicate while P_8 remained uncombined. The final results of this process are proven to be highly beneficial, so that predicates were substantially reduced. In this example, out of nine predicates, just four predicates were finally considered to perform fragmentation; eliminating almost 60% of the original space of predicates. Such a reduction rate attests the undeniable quality of the proposed filtering process. Table 13 pointed out the newly-formed predicates.

Hierarchical clustering process

Based on Table 13, along with Eq. (11), Query Predicate Usage Matrix (QPUM) was formed. This matrix basically replaced the original predicates with their contained queries to form the numerical pattern of queries, as shown in Table 14. Then, secondly, using QPUM along with Eq. (14), Query Difference Matrix was constructed as presented in Table 15.

$$QPUM_{ij} = \begin{cases} 1, & Q_i contain\, P_j \\ 0, & otherwise \end{cases} \tag{13}$$

$$QDM_{ij} = \begin{cases} Value, & Dif\,(Q_i, Q_j) \\ 0, & otherwise \end{cases} \tag{14}$$

Every element (mqd_{ij}) indicates the difference between Q_i and Q_j. The agglomerative hierarchical process was then initiated with QDM as initial values. The query pairs that shared the most predicates were grouped together in one cluster. By strictly following this process, the optimal fragmentation was bound to be achieved. The final results of the clustering process are shown in Table 16.

Table 17 Final distribution of predicates over fragments

Fragment	Query contained	Predicate contained	Data derivation
F1	Q1 and Q2	P1 and P3	$A6 < 20$ and $A3 \geq 4000$
F2	Q5 and Q7	P1 and P4	$A6 < 20$ and $A3 < 4000$
F3	Q6 and Q3	P2 and P3	$A6 > 20$ and $A3 \geq 4000$
F4	Q8 and Q4	P2 and P4	$A6 > 20$ and $A3 < 4000$
F5	Else	–	$A6 = 20$

Table 18 Final fragments

A1	A2	A3	A4	A5	A6
F1					
112	Hassan Mustaf	4000	Clerk	1	10
117	Burney Mich	6000	Manager	1	5
123	Ebtesam Ismail	4500	Manager	0	10
124	Ibrahem Mohammed	4100	Clerk	1	15
128	Robin Shobes	6600	Manger	0	15
F2					
111	Ali Mohammed	3500	Clerk	1	5
113	Adel Ahmed	2000	Employee	1	5
114	Jasmin Yaser	2200	Clerk	0	15
115	Salma Mohammed	1700	Employee	0	15
118	Lilly Eriks	1800	Employee	0	10
119	Ali Ali	2100	Clerk	1	10
120	Yaser Twfik	2000	Employee	1	15
121	Nehmi Ahmed	1570	Employee	1	15
F3					
129	Ted Mosbey	5100	Clerk	1	25
138	Salem Ali	7000	Manager	1	25
140	Hussien Ahmed	7000	Manager	1	30
F4					
130	Don Michel	1500	Employee	1	25
131	Nasem Yasen	2200	Clerk	1	25
132	Asma Ahmed	2200	Clerk	0	30
133	Jasmin Saleh	1750	Employee	0	30
134	Salam Salam	1800	Employee	0	30
135	Natalia ali	2000	Employee	0	25
137	Faten Salem	3200	Clerk	0	25
139	Yaser Hussien	3000	Clerk	1	30
F5					
116	Jhon Bush	1250	Employee	1	20
122	Mohsen Saleh	3200	Employee	1	20
125	Ibrahem Ali	1100	Employee	1	20
126	Ahmed Adel	2000	Employee	1	20
127	Mustafa Adel	1800	Employee	1	20
136	Tamer Hassan	4600	Manager	1	20

Table 19 Data fragments' statistics

FID	Fname	Cardinality	Size (byte)	Selectivity in regard with company dataset (%)
1	F_1	5	405	16.67
2	F_2	8	648	26.67
3	F_3	3	243	10.00
4	F_4	8	648	26.67
5	F_5	6	486	20.00
Total	5	30	2430	100

Table 20 FPM

Predicate/fragment	F1	F2	F3	F4
P1	1	1	0	0
P2	0	0	1	1
P3	1	0	1	0
P4	0	1	0	1

Table 21 QUM

Query/site	S_1	S_2	S_3	S_4	S_5	S_6
Q_1	1	1	0	0	0	0
Q_2	0	0	1	1	1	0
Q_3	0	1	1	1	0	0
Q_4	1	0	0	1	1	0
Q_5	1	0	0	0	1	1
Q_6	0	1	1	0	0	1
Q_7	0	0	0	0	1	1
Q_8	1	0	1	0	0	0

Fragment generation process

The relative predicates in the newly created clusters were determined for each cluster in Table 16 by comparing the predicates of the original queries with the filtered predicates in Table 13. The overlap checker, which was involved in the predicates filtering, would then sort predicates in a way that entirely avoided data overlapping as fragments formed after these defining predicates. In an effort to drastically reduce the production of empty fragments, no conflicting predicates were expected to be united in the same group of predicates. Table 17 clearly expressed the final distribution of predicates over fragments (14).

Given that we have examined [8], it is important to mention that the results obtained are comparable to Amer's results. But our proposed approach demonstrated its superiority by producing the identical outcomes at the initial stage of RDDBS design without even mentioning query frequencies. Additionally, the adjustment method suggested in [8] is no longer needed, which has significantly reduced the process' complexity. This contribution has been done by the automatic fragment

Table 22 FSM

Site/fragment	F_1	F_2	F_3	F_4
S_1	1	1	0	2
S_2	1	0	2	0
S_3	1	0	2	1
S_4	1	0	1	1
S_5	1	2	0	1
S_6	0	2	1	0

Table 23 SPM

Site/predicate	P_1	P_2	P_3	P_4
S_1	2	2	1	3
S_2	1	2	3	0
S_3	1	3	3	1
S_4	1	2	2	1
S_5	3	1	1	3
S_6	2	1	1	2

Table 24 TSPM

Site/predicate	P_1	P_2	P_3	P_4
S_1	51	63	75	39
S_2	70	62	50	82
S_3	57	51	47	61
S_4	69	68	72	65
S_5	64	81	82	63
S_6	59	62	64	57

Table 25 TSFM

Site/fragment	F_1	F_2	F_3	F_4
S_1	126	90	139	102
S_2	120	152	112	144
S_3	104	118	98	112
S_4	141	134	140	133
S_5	146	127	163	144
S_6	123	116	126	119

generation after completing the clustering procedure. The obtained fragments are finally depicted in Table 18.

When the fragmentation process completed, some statistics for data fragments recorded (Table 19) to be used for data allocation and performance evaluation.

Table 26 CPM

Cluster/predicate	P_1	P_2	P_3	P_4
C_1	5	3	2	6
C_2	2	5	6	1
C_3	3	3	3	3

Table 27 TCPM

Cluster/predicate	P_1	P_2	P_3	P_4
C_1	16	31	36	11
C_2	37	27	22	42
C_3	18	26	28	16

Table 28 TCFM

Cluster/fragment	F_1	F_2	F_3	F_4
C_1	52	27	67	47
C_2	59	79	49	64
C_3	46	34	54	44
THV	52.3	46.67	56.67	51.67

Table 29 Fragment's distribution over clusters, and their sites

Scenario/fragment	F_1	F_2	F_3	F_4
Full replication	All	All	All	All
Partial replication	C2(S2)	C2(S2)	C1(S5)	C2(S2)
Non-replication	C2(S2)	C2(S2)	C1(S5)	C2(S2)

Table 30 Query frequency matrix (QFM)

Query/site	S_1	S_2	S_3	S_4	S_5	S_6	TQF
Q_1	2	0	0	0	1	3	6
Q_2	2	2	0	0	3	0	7
Q_3	0	0	3	3	0	0	6
Q_4	0	0	0	1	0	3	4
Q_5	0	2	0	0	0	2	4
Q_6	0	1	1	3	0	0	5
Q_7	2	0	0	1	0	0	3
Q_8	0	1	1	0	2	1	5

Allocation process

First of all, Fragment Predicate Matrix (FPM) was built based on the outcomes of filtering process of predicates and final generated fragments. FPM (Table 20) is a matrix

Table 31 FSM

Fragment/site	S1	S2	S3	S4	S5	S6
F1	4	2	0	0	4	3
F2	2	2	0	1	0	2
F3	0	1	4	6	0	0
F4	0	1	1	1	2	4

Table 32 Site predicate matrix (SPM)

Predicate/site	S1	S2	S3	S4	S5	S6
P1	6	4	0	1	4	5
P2	0	2	5	7	2	4
P3	4	3	4	6	4	3
P4	2	3	1	2	2	6

Table 33 TSPM

Site/predicate	P_1	P_2	P_3	P_4
S_1	94	111	126	79
S_2	136	105	156	85
S_3	123	111	135	99
S_4	131	109	137	103
S_5	154	161	175	140
S_6	83	121	142	62

Table 34 TSFM

Site/fragment	F_1	F_2	F_3	F_4
S_1	220	173	237	190
S_2	292	221	261	190
S_3	258	222	246	210
S_4	268	234	246	212
S_5	329	294	336	301
S_6	225	145	263	183

of predicate usage over fragments according to Table 17. In the **Initial allocation, we** assume that the DBA provided the Query Usage Matrix (Table 21) so that each point showed whether Q_i was released from S_j or not.

Since each fragment was essentially represented (accessible) by its original queries/predicates, a fragment site matrix (FSM, Table 22) was generated using QUM with the aid of FPM. Table 23 was created to produce the site-predicate matrix using Eq. (6) and the FPM and FSM matrices.

Then, SPM and CSM were combined using Eq. (7) to create TSPM in Table 24. After that, using Table 13 and TSPM, TSFM was depicted in Table 25. Contrarily, SPM was

Table 35 CPM, TCPM, TCFM

Cluster/predicate (CPM)	**P_1**	**P_2**	**P_3**	**P_4**
C_1	10	2	8	4
C_2	4	7	7	4
C_3	6	11	9	8
Cluster/predicate (TCPM)	**P1**	**P2**	**P3**	**P4**
C1	32	57	53	36
C2	74	54	76	52
C3	36	32	44	24
Cluster/fragment (TCFM)	**F1**	**F2**	**F3**	**F4**
C1	85	68	110	93
C2	150	126	130	106
C3	80	60	76	56

Table 36 TTSFM and TTCFM

Site/fragment (TTSFM)	**F_1**	**F_2**	**F_3**	**F_4**
S_1	89,100	112,104	57,591	123,120
S_2	118,260	143,208	63,423	123,120
S_3	104,490	143,856	59,778	136,080
S_4	108,540	151,632	59,778	137,376
S_5	133,245	190,512	81,648	195,048
S_6	91,125	93,960	63,909	118,584
Cluster/fragment (TTCFM)	**F_1**	**F_2**	**F_3**	**F_4**
C_1	34,425	44,064	26,730	60,264
C_2	60,750	81,648	31,590	68,688
C_3	32,400	38,880	18,468	36,288
THV	42,525	54,864	25,596	55,080

Table 37 Fragment's Distribution over Clusters of Sites, and their Sites

Scenario/fragment	**F_1**	**F_2**	**F_3**	**F_4**
Full replication	All	All	All	All
No replication	C2(S2)	C2 (S3)	C2(S2)	C2(S3)
Partial replication	C2(S2)	C2(S3)	C1(S5), C2(S2)	C1(S5), C2(S3)

combined using Eq. (8) to create CPM (Table 26). Last but not least, CPM and CCM were combined using Eq. (9) to create TCPM, as shown in Table 27.

Finally, TCFM was identified in Table 28 utilizing TCPM and Table 13 together. The final allocation decision was made using TCFM, and it is shown in Table 29.

Post allocation process

If S_i targets fragment F_j using queries Q_k and Q_h, S_i would target fragment $(k+h)$ times given QFM in Table 30. S_1 would access F_1 five times, for instance, if Q_1 and Q_2

Table 38 problem addressed statistics

Problem	Queries#	Retrieval queries in percentage (%)	Update queries in percentage (%)
P1	8	88	12
P2	24	78	22
P3	48	68	32
P4	120	48	52
P5	240	30	70

Table 39 Fragments and relation distribution over all clusters of sites

Cluster	Site	F1	F2	F3	F4	Whole relation
Full replication allocation scenario (imposed clustering)						
C1	S1					
	S5	1	1	1	1	
C2	S2	1		1		
	S3		1		1	
C3	S4	1	1			
	S6			1	1	
Non-replication allocation scenario (imposed clustering)						
C1	S1					
	S5					
C2	S2	1		1		
	S3		1		1	
C3	S4					
	S6					
Partial replication allocation scenario (imposed clustering)						
C1	S1					
	S5			1	1	
C2	S2	1		1		
	S3		1		1	
C3	S4					
	S6					
Whole relation allocation scenario (for both imposed/relaxed clustering)						
C3	Site					
	S6					1

are released with 2 and 3 access, respectively. Based on this hypothetical situation, the TQF values—which stand for the total frequency of each query—are defined.

Instead of utilizing QUM at the initial stage of design, QFM would be used to compute the total access cost of all sites and their pertinent fragments as shown in the fragment site matrix (FSM, Table 31). A QFM matrix would serve as the foundation for the FSM. Then, Table 32's SPM was produced using FSM. The same process would be used to create TSPM and TSFM (Tables 33, 34).

The same procedure was also followed in the same pattern to find TCFM (Table 35) based on CPM and TCPM.

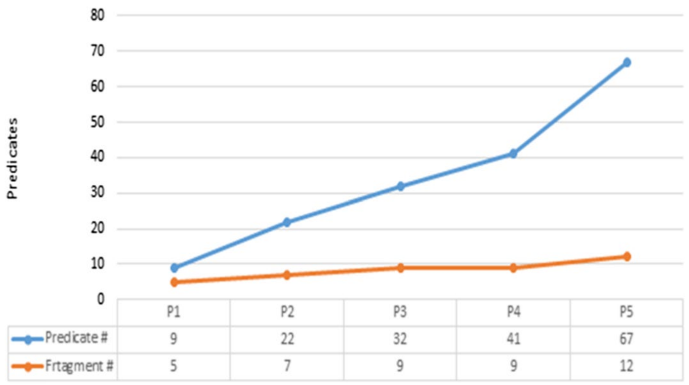


Fig. 7 Relationship between predicates and resulted fragments

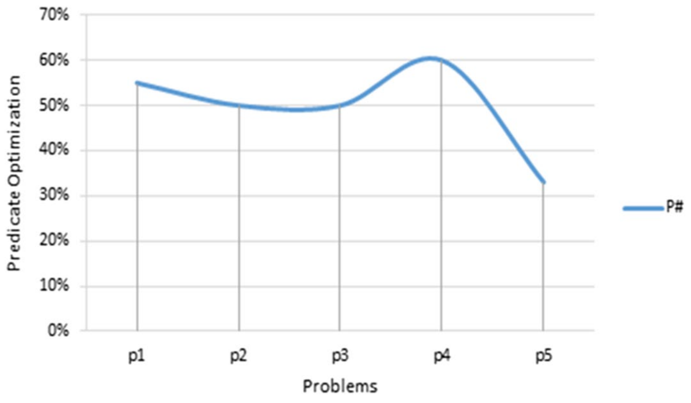


Fig. 8 Predicate filtering optimization

The fragment size list was then multiplied by TSFM to obtain the TTSFM's real transmission costs (Table 36). The final fragment allocation decision is shown in Table 37 after TCFM (Table 36) multiplied by the fragment size list to obtain the actual transmission costs over the clusters shown in TTCFM (Table 36).

Performance evaluation and discussion

This work is painstakingly designed to intelligently fragment data and assign the fragmented data into the sites where they are intensively required. Consequently, communication expenses and response time—two important performance indicators—are sharply decreased. To verify that, a thorough evaluation has been conducted to verify this claim. Five problems are addressed with queries: 8, 24, 48, 120, and 240, which run on the same "company" dataset with 1000, 2000, 5000, and 25,000 records, respectively. The retrieval queries make up the majority of the considered queries in the second and third problems. However, the update queries have a higher percentage in the final two problems. These problems and their basic statistics are drawn in Table 38. The distribution of the already-obtained fragments as per four data allocation scenarios (using which the evaluation is affected) is also given in Table 39.

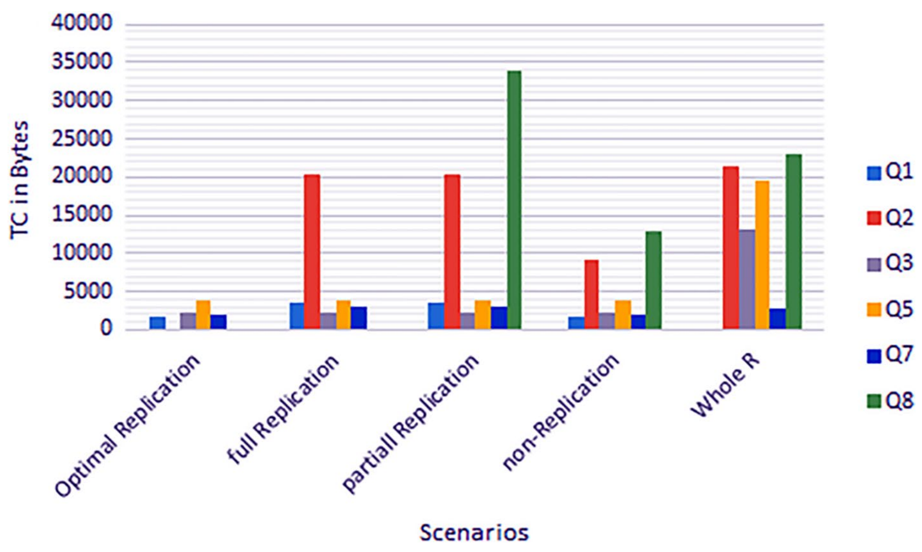


Fig. 9 TC reduction in percentage over all scenarios—Problem 1

Many performance-related factors are considered during the evaluation process. Among these factors are, the number of data clusters (fragments) generated for each problem, the reduction rate recorded in predicates resulted from the proposed filtering process, and the transmission costs incurred due to running the queries against the distributed fragments. Figure 7 shows that when the predicates space increased, the query clusters (fragments) would negligibly grew. In other words, fragments number was significantly lower than predicates in all problems. This contribution is met because the initial predicates were subjected to the filtering process, which greatly contributed to this reduction. The final version of predicates was thus always observed to be much lower than the initial ones. Consequently, the final predicates result in a small number of data fragments, as shown in Fig. 8. Therefore, the filtering process can be a drastic solution capable of minimizing the number of predicates (out of hundreds or even thousands) by which the final fragments are set to be found. Indeed, the smaller the number of fragments, the better the DDBS performance, because the distributed queries are severely limited to being handled in a smaller number of sites/clusters.

Figure 9 also showed the impact of the predicate filtering process on the fragmentation process. The axis represented the percentage of data fragments obtained compared with the number of predicates used. In problem (1), for example, there were nearly ten predicates; however, using the proposed filtering process, nearly 40% of the predicates were reduced, resulting in only 60% used to generate half of the predicate-driven fragments.

The ultimate goal has been to minimize the transmission costs (TC) which is successfully accomplished. According to the objective function (Eq. 1), the minimization of (TC) for each problem is recorded. Every query, among those under consideration, was run upon the company dataset in accordance with five scenarios of data allocation: (1) the optimal allocation scenario (in which it was supposed that the query and the vast majority of data, if not all, were placed in the same place so the query is locally processed), (2) full replication-based allocation (each fragment was replicated in all clusters); (3) partial replication-based allocation (some fragments were replicated while others were not), (4)

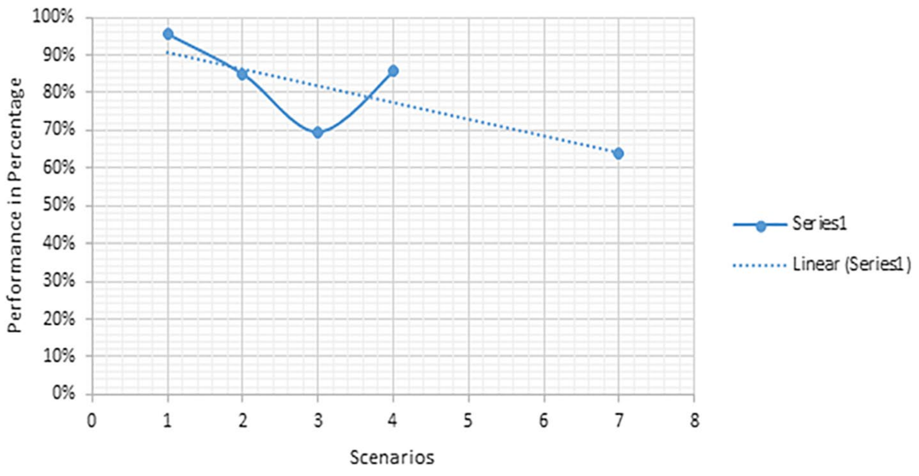


Fig. 10 Performance in percentage over all scenarios—Problem 1

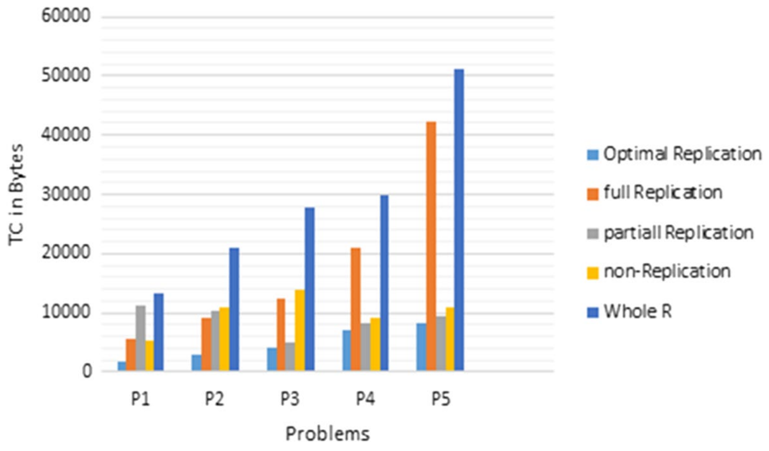


Fig. 11 TC reduction in percentage over all scenarios—all Problems

non-replication-based allocation (each fragment was only placed into the cluster with the highest access costs); and (5) the whole relation-based allocation. The best scenario, which came the closest to the optimal results for each scenario, was then chosen to be integrated into technique design after the results of each scenario compared to those of an optimal scenario (used as a baseline scenario). As shown in the following discussion and graphs, several trials were carried out to conclude which fragmentation and allocation scenario best suited DDBS performance (Figs. 9, 10, 11). It is worth pointing out that, during discussion, the term "performance" means the percentage of overall transmission costs (including communication costs between sites/clusters) that were recorded to be incurred as distributed queries were handled. Figures 9 and 10 illustrate the results obtained for problem 1.

According to Fig. 9, when scenarios taken individually, the non-replication-based scenario had the best reduction in TC in all queries except for Q_8. The full and partial replication, on the other hand, were observed to have almost the same reduction rate. Except for Q_1, the entire relation scenario was the worst scenario in all queries. However, when

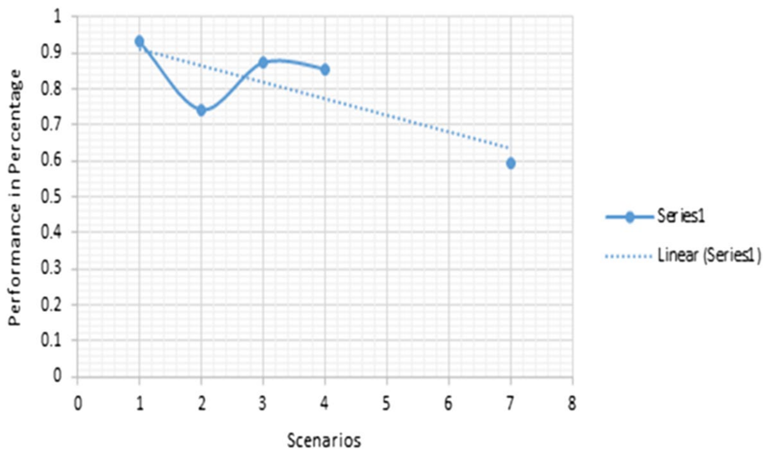


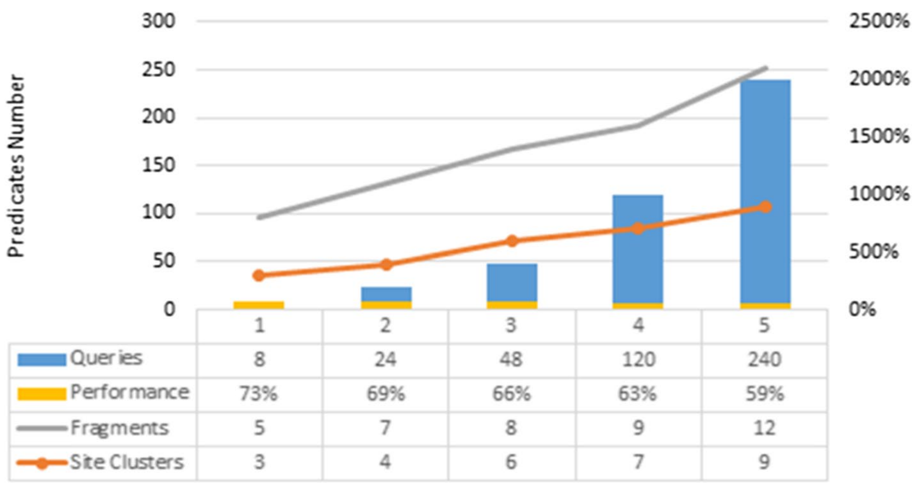
Fig. 12 TC reduction in percentage over all scenarios—all Problems

these findings were aggregated across all queries (for this case), the leader in TC reduction was the full replication with 1% higher score than non-replication. The relation-based scenario as a whole came in the second rank, followed by the partial replication. Additionally, the data depicted in Fig. 10 support the conclusions on Fig. 9. In terms of performance rate, the full replication scenario is 11% closer to the optimal scenario, followed by non-replication at 12% and non-replication at 27%. However, the whole relation scenario is observed to have the worst performance ever, with a performance deterioration rate of 40%.

It was unsurprising that the fully replicated scenario outperformed all others. That is because, in the full replication-based scenario, the retrieval queries were higher than update queries, and the replication was the crucial factor in supporting this scenario [33]. This scenario, on one hand, affected the RDDBS's performance badly as the number of update queries were growing among the original considered queries. The non-replication allocation scenarios, however, showed a significantly lower TC as update queries took up more space. Partial replication, however, was observed to deservedly outperform all other scenarios in both cases of retrieval or update queries being larger. This scenario reduced TC by 88% when compared to scenarios (2) with 70% and scenario (3) with 84% combined (77%). This claim is proven to be totally correct in Figs. 11 and 12, where all problems addressed saw heightened DDBS performance when partial replication considered. The forth scenario of whole relation allocation was the worst ever.

As problem P1 was handled, it was evident from Figs. 10 and 11 that both full replication and non-replication-based scenarios had the upper hand over all other possibilities. The complete replication-based scenario in P2 had similar trend. Partial replication, however, began to take the lead over the rival situations, followed by the non-replication scenario when update queries were progressively increasing in problems (P3–P5). On the other hand, the whole relation-based scenario still produces the worst outcomes ever for all problems. The findings depicted in Fig. 12 further confirmed these assertions.

In summary, regardless of whether retrieval or update queries occupy the majority of the space among all queries under consideration, the partial replication has significantly minimized the communication costs. Four further trials, with: 24, 48, 120, and



	1	2	3	4	5
Queries	8	24	48	120	240
Performance	73%	69%	66%	63%	59%
Fragments	5	7	8	9	12
Site Clusters	3	4	6	7	9

Fig. 13 Performance after applying proposed approach

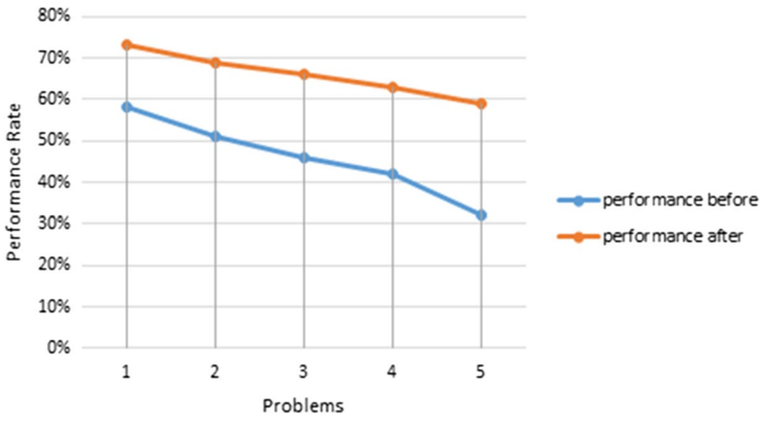
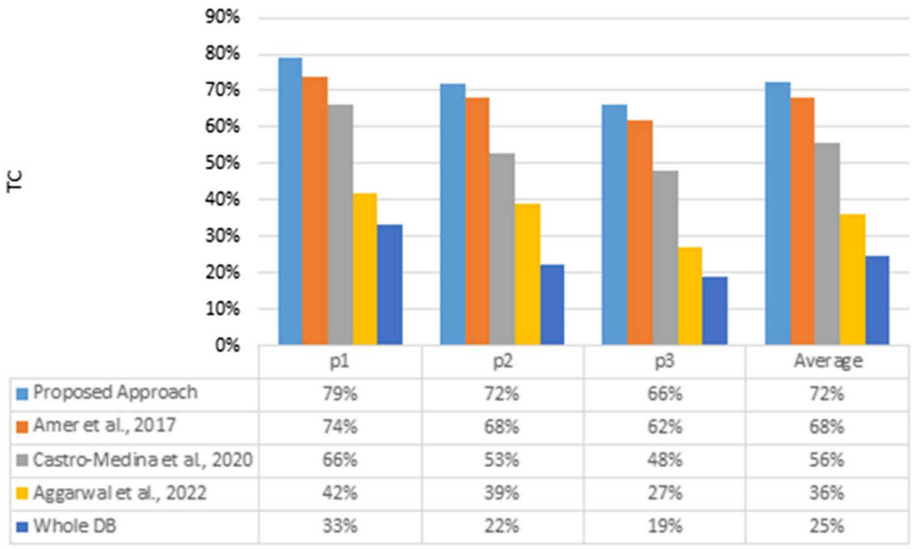


Fig. 14 Performance before and after fragmentation

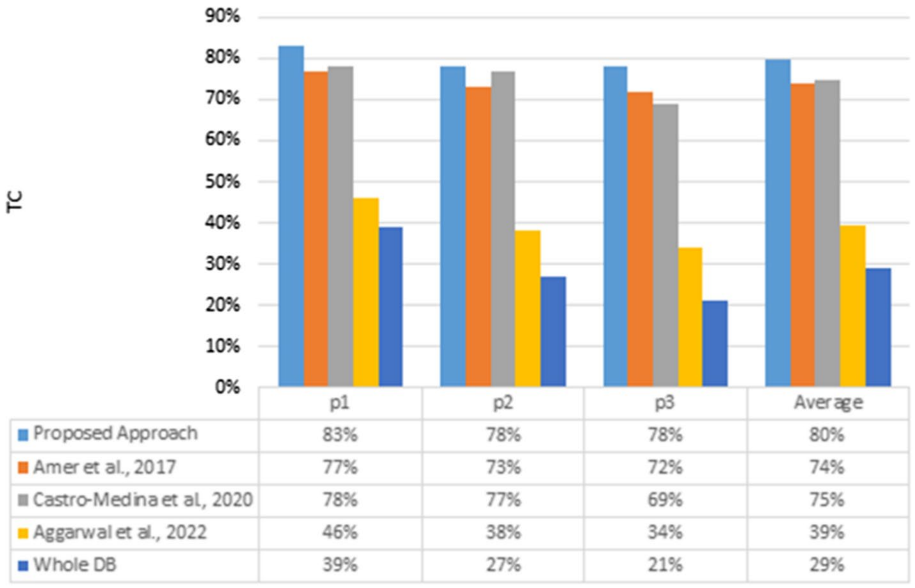
240 queries, have been conducted for further proof. Figure 12 shows that the replication-based scenario is the best choice because the retrieval queries occupied a large portion, as with P1, P2, and P3. However, the scenario of non-replication was by far the best option when update queries constituted the large percentage, P4 and P5. The partial replication scenario was also by far better than those two former scenarios, neglecting the portion of retrieval or update queries among all considered queries, as presented in all problems. Moreover, from Fig. 13, it is shown that the partial replication scenario has been so close to the optimal allocation scenario. As a general rule, the data replication scenario accompanied by negative effects on the communication cost reduction due to the larger update queries.

Finally, Figs. 13 and 14 show the performance improvement before and after implementing the proposed approach. Before fragmentation, performance was dropping steeply. After fragmentation, however, performance had fallen gradually as the number of queries was growing steadily. Furthermore, the number of the site clusters and fragments were seen to have a significant impact on the DDBS performance. In short, the



	p1	p2	p3	Average
Proposed Approach	79%	72%	66%	72%
Amer et al., 2017	74%	68%	62%	68%
Castro-Medina et al, 2020	66%	53%	48%	56%
Aggarwal et al., 2022	42%	39%	27%	36%
Whole DB	33%	22%	19%	25%

Fig. 15 Scenario 1—Rate of TC reduction—adult DS



	p1	p2	p3	Average
Proposed Approach	83%	78%	78%	80%
Amer et al., 2017	77%	73%	72%	74%
Castro-Medina et al, 2020	78%	77%	69%	75%
Aggarwal et al., 2022	46%	38%	34%	39%
Whole DB	39%	27%	21%	29%

Fig. 16 Scenario 2—Rate of TC reduction—adult DS

larger the query size, site clusters, and data fragments, the small deterioration of DDBS performance recorded using our proposed approach.

External evaluation

Our proposed approach has been externally compared with [8, 9, 19]. Three problems (P1–P3) are generated to execute the final three tests on the Adult and bank marketing datasets. They both dataset are retrieved from the machine learning repository in order to verify the proposed approach's competence on real datasets. The Adult database has 48,842 records with 14 attributes, is 3.8 MB in size, and has a publicly available

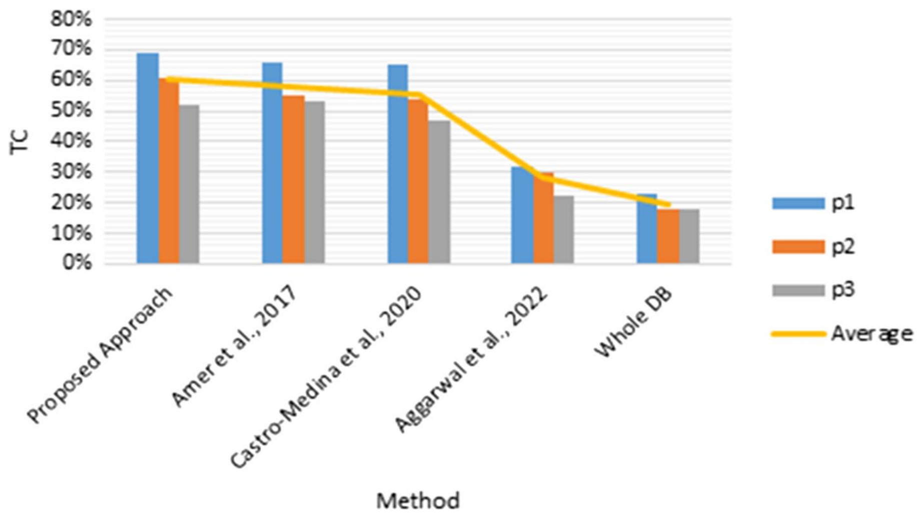


Fig. 17 Scenario 1—Rate of TC reduction—bank DS

description.[1] The Bank Marketing has 45,211 records with 16 attributes, is 3.8 MB in size, and has a publicly available description.[2]

With 50, 70, and 80 queries used in each of the three trials, the query set (200 queries) for the Adult dataset is constructed in the same manner of the first experiment's queries in the result section. The select-type queries are represented by 75% (150 queries), and 25% are updates (insert and delete, 50 queries). However, according to [27], not all queries across all experiments should use all features; instead, 60% of queries used 9 attributes, while 40% used only 6. In both scenarios, the TC results are accumulated across all three experiments and averaged in the same way that the given-above company dataset's results evaluated for all three works under consideration [8, 9, 19]. Two scenarios are examined. In the first case, the fragmented database is used with partial replication (which has five fragments, F1–F5). In the second scenario, non-replication was used for the whole DB and for all works. Figures 15 and 16 depict the findings for both scenarios, which demonstrate that our proposed approach continues to outperform [9, 19], with [19] being superior to [9], and comparable to [8]. However, the whole DB before fragmentation had the poorest performance. It is worth referring that the DS in all following figures stand for the dataset. Our proposed approach and [8] both performed similarly in the first scenario (Fig. 15), with the proposed approach outperforming [8] slightly, as the TC minimization average reached 72% in the average, while it was 68% in [8], 56% in [19], and 36% in [9], respectively. This is due to the update queries that had a detrimental impact on RDDBS performance as each update query had to be multiplied by the quantity of the sites where the query in question was executed. Both works (our approach and [8]) were comparable in the first scenario. The proposed work, however, demonstrated significant TC minimization as the query set expanded. The proposed work has the lead over its rivals in the averaged results, and is a superior to the whole DB. The whole DB and [9] had the worst performance in TC minimization, with 36% and 25%, respectively.

[1] http://archive.ics.uci.edu/ml/datasets/Adult.

[2] https://archive.ics.uci.edu/dataset/222/bank+marketing.

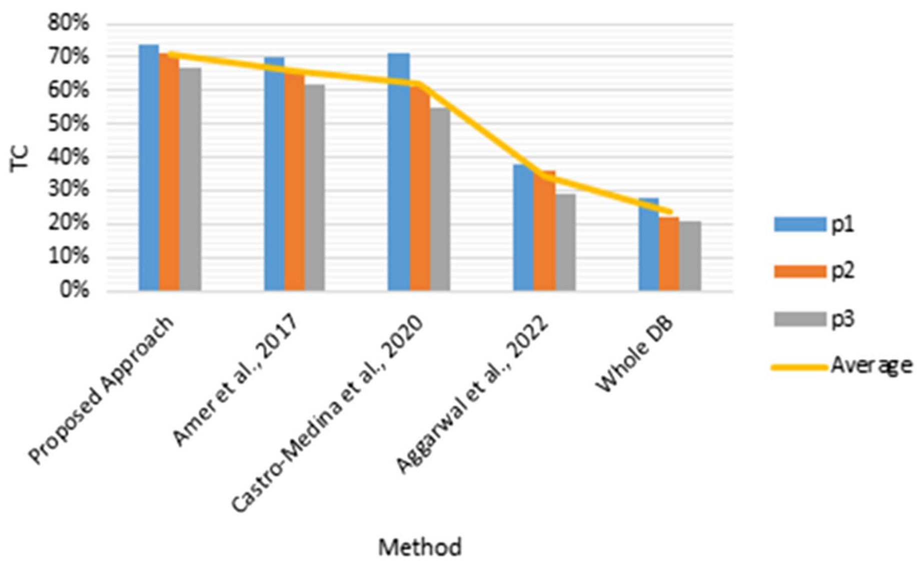
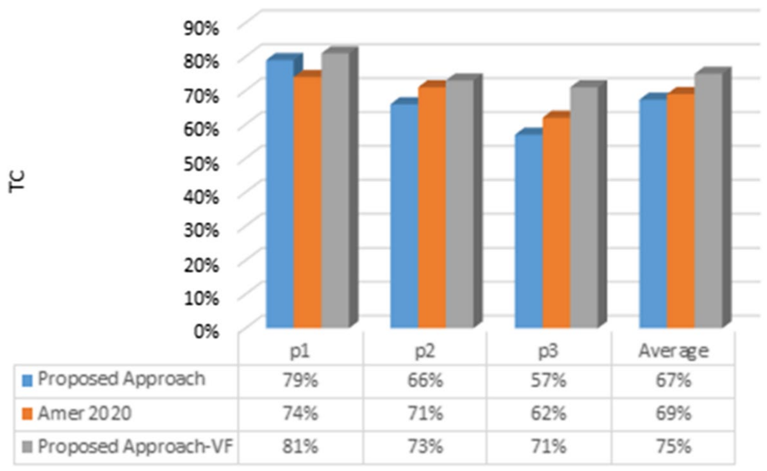


Fig. 18 Scenario 2—Rate of TC reduction—bank DS

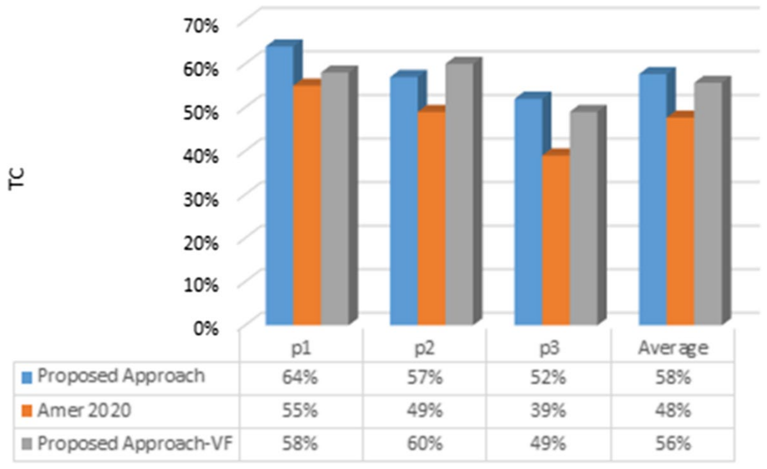
Overall, our proposed approach performed significantly better than [8] and was far superior to [9] in the second scenario (Fig. 12), and the initial TC minimization average was 83%, 78%, and 78%, respectively. Castro-Medina et al. [19]'s approach achieved 78%, 77%, and 69%, respectively, making its performance closely equivalent to our proposed one. The main factor that accounts for the comparable performance of our proposed approach and that of [19] is that, unlike in scenario 1, no replication scenario was used. So, the update queries that adversely affected RDDBS performance were not multiplied by the number of sites. The averaged findings from all three experiments demonstrated that our proposed work and [19] were both appreciably better than [8, 9] and the whole DB. In turn, the whole DB performed the worst with only 29% in TC minimization, which is identical to scenario 1. As our proposed approach increased from 54% in the first scenario to 59.70% in the second scenario, the approach has a substantial improvement with a very perceptible difference in its behavior in both scenarios. Lastly, it is noted that all approaches perform significantly better in scenario 2 than scenario 1.

Following the same examination procedure applied on the Adult dataset, we find the results on the Bank Marketing datasets attest the competitiveness of our proposed approach as well. Both Figs. 17 and 18 depict the findings for both scenarios. Similarly, the whole DB before fragmentation had the poorest performance. In the first scenario, both works [8, 19] performed nearly equally well (see Fig. 15), with [8] slightly outperforming [19] since its average TC reduction rate reached 58% while it is 55% in [19]. With an averaged TC of 61%, our proposed approach significantly surpassed all of these other approaches. In the first problem, both works [8, 19] were comparable. But when the query set grew, the proposed work showed significant TC minimization. The proposed approach outperforms the whole DB and its competitors in the averaged results. With 28% and 20%, respectively, the performance of the whole DB and [9] was the poorest in TC minimization. Similar to what happened in Adult, the update queries were to blame for the poor overall performance of RDDBS. Our proposed work still demonstrates its



	p1	p2	p3	Average
🟦 Proposed Approach	79%	66%	57%	67%
🟧 Amer 2020	74%	71%	62%	69%
⬜ Proposed Approach-VF	81%	73%	71%	75%

Fig. 19 Partial replication—adult DS



	p1	p2	p3	Average
🟦 Proposed Approach	64%	57%	52%	58%
🟧 Amer 2020	55%	49%	39%	48%
⬜ Proposed Approach-VF	58%	60%	49%	56%

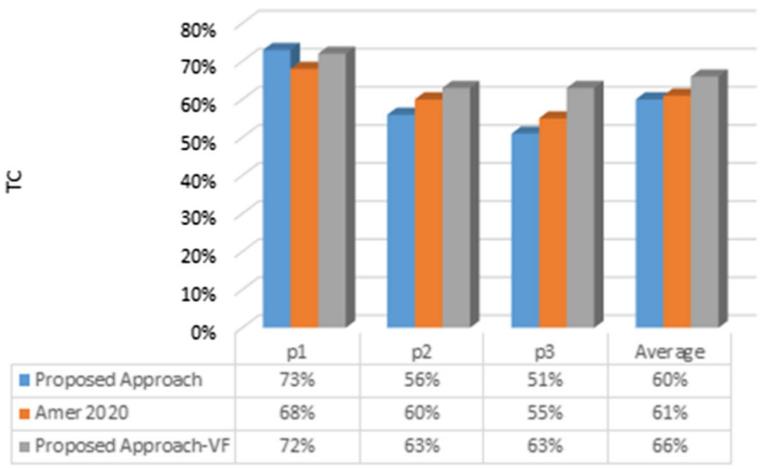
Fig. 20 Full replication—adult DS

superiority in the second scenario, attaining 71% in for averaged TC, compared to 66% and 62% in [8, 19], respectively. Both works [8, 19] were equivalent to scenario 1, with [8] marginally outperforming [19]. With 34% and 24%, respectively, the performance of the whole DB and [9] was the poorest in TC minimization. Overall, all works performed better in scenario 2 than in scenario 1.

To conclude, compared to that of the adults, all approaches' performance on Bank Marketing dataset is noticeably worse. We think this is due to using an increasing number of attributes. With the Bank Marketing, we employed 11 attributes as opposed to the adult, where we only used 9. This may lead to the conclusion that the more attributes used, the higher the deterioration the DDBS performance may suffer.

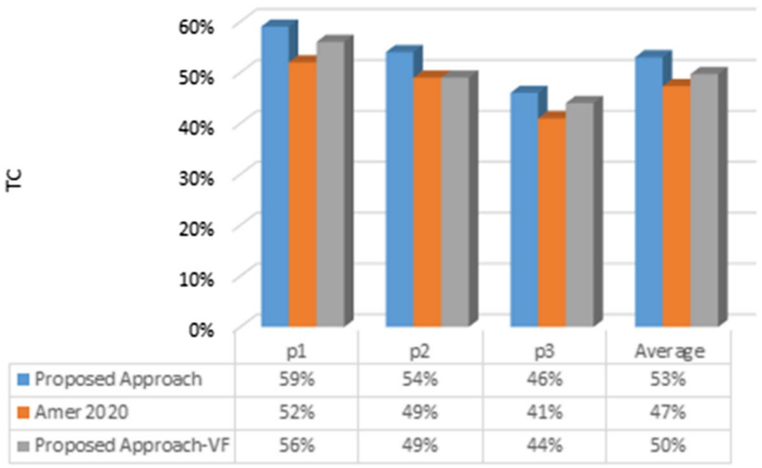
A brief ablation study

We have continued by contrasting our recent findings with [33], whose author employed the k-means for vertical fragmentation. Additionally, in our second variation of the



	p1	p2	p3	Average
🟦 Proposed Approach	73%	56%	51%	60%
🟧 Amer 2020	68%	60%	55%	61%
■ Proposed Approach-VF	72%	63%	63%	66%

Fig. 21 Partial replication—bank DS



	p1	p2	p3	Average
🟦 Proposed Approach	59%	54%	46%	53%
🟧 Amer 2020	52%	49%	41%	47%
■ Proposed Approach-VF	56%	49%	44%	50%

Fig. 22 Full replication—bank DS

study, we used the identical allocation process but switched to the k-means clustering approach rather than the hierarchical one. To put it another way, we divided the target datasets using the hierarchical and k-means clustering algorithms. Therefore, we compared the vertical fragmentation with that based on hierarchical clustering. The communication and processing costs for all works were compared while modifying the number of sites or clusters, the replication scenario (partial and full), and the kind of queries (update/retrieval). We employed two situations (partial replication and full replication) and the same inspection process that was used with the Adult and Bank datasets in the previous sub-section. 150 queries were run on each dataset, totaling 300. The queries were built in the same way as those from the result section of the previous experiments. Sixty percent (90 queries) of the total are select-type queries, and forty percent (60 queries each for insert and delete). In contrast to the Bank dataset, which used 11 attributes, the adult dataset only included 9 attributes. In the first scenario, the fragmented database—which also has five fragments, F1–F5—was used with partial replication. Full replication was applied to the second scenario.

Figures 19 and 20 show the results for both situations, showing how well the VF-based approach we propose outperforms both [33] and our HC-based strategy in the partial replication scenario (Fig. 19). In particular, the robust greedy-based allocation process of the proposed study, which differs significantly from that of [33], makes our proposed strategy the best with VF. We believe that besides the increasing number of attributes used, the update queries, which had a negative effect on RDDBS performance because each update query had to be multiplied by the number of sites where the query in question was conducted, are to blame for the retrogress of our HC-based strategy. In the second scenario, our proposed HC-based approach outperforms both [33] and our VF-based strategy (Fig. 20). With full replication, the request on the instances (tuples) would heavily impact the TC costs, making the horizontal fragmentation more suitable for DDBS design. Surprisingly, the same results on Adult are secured with the bank dataset, yet with lower percentage for all works (Figs. 21, 22). But as the query set grew, the proposed work showed a significant average TC minimization, chiefly in the second scenario. Overall, it has been found that the VF-based fragmentation us much better than the horizontal one using the partial replication, and vice versa.

Overall, the results can directly imply that, in the partial replication situation, it would be desirable to construct the DDBS with vertical fragmentation. Otherwise, the best fragmentation is horizontally based.

Features of proposed approach

The following characteristics feature our proposed approach: (1) It yields the same initial results (at the initial stage of RDDBS design) as the partition algorithm proposed in [8] without the need for any statistics pertaining to the database log, (2) It uses the hamming distance instead of the predicate affinity process to cluster predicates and then find fragments, (3) It takes into account the restricted-type queries, (4) It addresses the fragment allocation problem without the need for additional complexity. The proposed approach also enjoys offering a method for data filtering, combining site grouping and fragmentation processes using clustering, utilizing the Knapsack algorithm for allocation and replication, and, finally, testing against some competitors both internally and externally in accordance with four allocation scenarios.

Approach complexity

For the purpose of satisfying an efficient data fragmentation and allocation approach, many algorithms have been combined. As a result, the time complexity of the included (complex) components limits the complexity of the proposed approach. The following formula is drawn to calculate each component's temporal complexity, as follows:

1. The difficulty of defining and filtering predicates is $O(NP^2)$, Where NP is the total number of predicates to be considered.

2. The complexity of the query/predicate clustering procedure is $O(N_2 * \log^2{}_n)$, where N is the total number of queries.
3. The complexity of clustering sites is $O((M^2)*\log^2{}_m)$, where M is the total number of sites taken into consideration by the DDBS.
4. In both scenarios, the complexity of knapsack-based allocation is $O(CN*m*fn) + O(m^2 *fn)$.

Accordingly, the time complexity of this approach is constrained by $(O(NP^2) + O((M^2) * \log^2{}_m) + O(N^2 * \log^2{}_n) + (O(CN*m*fn) + O(m^2 *fn))$ at the worst case and by $O(m^2 *fn)$ at the best case based on complexity computations of the included entities. On the other hand, this complexity is viewed as being unmatched for such a comprehensive heuristic algorithm-combining approach.

Work limitation

The domain of relational databases is the exclusive focus of this research. Furthermore, this research did not clearly address the response time and the bandwidth. As a result, in the future, rather of implicitly incorporating such measures with the objective function (see Eq. 1), it will be expressed explicitly using a newly proposed equation.

Conclusions and prospective

This paper presents a well-tuned heuristic approach for horizontal fragmentation and allocation. All DDBS design strategies (e.g., fragmentation, allocation, replication, site clustering, etc.) have been carefully combined into a single approach. It includes all heuristics, procedures, algorithms, and demonstrative examples. In the fragmentation process, the database was divided into smaller partitions using the proposed predicate-based hierarchical clustering process. The query predicates were first identified, and then filtered to be used for acquiring the query clusters (data fragments). The proposed technique's ability to effectively and greatly reduce the predicates space of each problem to the minimum final fragments has been one of its most distinguishing features. On the other hand, the data allocation mechanism is schemed based on the Knapsack algorithm-driven cost model, which utilized the capacities of the sites while drastically reducing network traffic. Two allocation phases and four replication scenarios, were carefully examined, and the best-fit scenario was then incorporated into the final RDDBS architecture. According to our findings, irrespective of retrieval or update queries that occupy the space of all queries under consideration, the partial replication has significantly lessened the communication costs, making it the optimal scenario.

To verify the proposed approach's competitiveness and to draw the best design for RDDBS, a good number of trials were carried out in completely unrelated circumstances. It is important to note that these efforts are intended to produce a thoughtful solution for both the initial and advanced stages of RDDBS design. Most importantly, an internal and external evaluation of the proposed approach have been made against the closely-relevant state-of-the-art competitors. The results demonstrated our approach's superiority and effectiveness over both synthetized and real datasets. It is important to note that we have made every effort to test our methods in many settings without favoring the proposed approach. To demonstrate the effectiveness of the proposed method, we conducted a thorough experimental study using multiple experiments. We have considered various numbers of sites and clusters, a large number of data fragments in each experiment, varying the query space between retrieval and update ones, taking into account the fragmentation of multi-relational relationships, addressing two allocation phases and four replication scenarios, and finally using communication costs and response time as performance test indicators. To demonstrate the competitiveness and general applicability of the proposed approach, each of these settings in each experiment is carefully investigated. Furthermore, we can say that all of the experimental study's settings show that our approach performs favorably. As a result, its performance is viewed as promising, reflecting the fact that the proposed approach can effectively generalize to any new dataset. Finally, the results of our quick ablation investigation directly suggest that it would be preferable to build the DDBS with vertical fragmentation in the partial replication scenario. Otherwise, horizontally based fragmentation is the best.

In conclusion, it is important to note that this approach has been developed with the goal of allowing it to process variety of relations at once, while taking RDDBS scalability into consideration. In reality, the extension of a single site is always subject to a predetermined restriction for every database. With scalable DDBS, databases can be dynamically expanded beyond a single site, so adding or removing sites makes it easier to respond to the needs of the targeted DDBS. In future work, as follow-up work, the issue of DDBS scalability and concurrency control [28, 29], as well as the network bandwidth [30] will be addressed. Moreover, one of our future goals is to test the methodology utilizing the lookup and query execution [31, 32] framework.

Acknowledgements
The authors would like to express their highly appreciated gratitude to the Research Office (Zayed University) for providing us with the tools needed to complete this work. The authors would also like to express their sincere thanks to the editor and the anonymous reviewers for their insightful feedback and constructive recommendations.

Author contributions
HIA has been a key contributor in conception, analyzing the results of all experiments, and the preparation, writing, and revising of the manuscript. AAA has been the key contributor in conception and design, implementing the approach, analyzing the results of all experiments, and the preparation, writing, and revising of the manuscript. SDR has been a key contributor in the preparation, writing, and revising of the manuscript.

Funding
This work has been supported by Research Incentive Fund (RIF) Grant Activity Code: R22083—Zayed University, UAE.

Availability of data and materials
The data used are publicly available in the UCI repository.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 18 June 2023 Accepted: 28 October 2023
Published online: 18 November 2023

References

1. Ortiz-Ballona AO, Rodríguez-Mazahua L, López-Chau A, Abud-Figueroa MA, Romero-Torres C, Castro-Medina F. A brief review of vertical fragmentation methods considering multimedia databases and content-based queries. In: International conference on software process improvement. Cham: Springer; 2021, October. p. 55–68).
2. Nashat D, Amer AA. A comprehensive taxonomy of fragmentation and allocation techniques in distributed database design. ACM Comput Surv (CSUR). 2018;51(1):1–25.
3. Castillo-García A, Rodríguez-Mazahua L, Castro-Medina F, Olivares-Zepahua BA, Abud-Figueroa MA. A review of horizontal fragmentation methods considering multimedia data and dynamic access patterns. In International conference on software process improvement. Cham: Springer; 2021, October. p. 69–82.
4. Mazumdar S, Seybold D, Kritikos K, Verginadis Y. A survey on data storage and placement methodologies for Cloud-Big Data ecosystem. J Big Data. 2019;6(1):1–37.
5. Sreedhar C, Kasiviswanath N, Chenna Reddy P. Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop. J Big Data. 2017;4(1):27.
6. Fauzi AAC, Rahman WFWA, Fauzi A, Weigelt F. Managing fragmented database in distributed database environment. J Math Comput Sci. 2021;7(1):8–14.
7. Amer AA, Mohamed MH, Al-Asri K. ASGOP: an aggregated similarity-based greedy-oriented approach for relational DDBSs design. Heliyon. 2020;6(1):1.
8. Amer AA, Sewisy AA, Elgendy TM. An optimized approach for simultaneous horizontal data fragmentation and allocation in Distributed Database Systems (DDBSs). Heliyon. 2017;3(12):e00487.
9. Aggarwal M, Bajaj SB, Jaglan V. Performance analysis of degree of redundancy for replication in distributed database system. In: 2022 1st international conference on informatics (ICI), New York: IEEE; 2022, April. p. 176–80.
10. Ge YF, Zhan ZH, Cao J, Wang H, Zhang Y, Lai KK, Zhang J. DSGA: a distributed segment-based genetic algorithm for multi-objective outsourced database partitioning. Inf Sci. 2022;612:864–86.
11. Singh A, Khehra BS, Mavi BS. Simplified-BBO for non-redundant allocation of data in distributed database design. In: 2021 IEEE international midwest symposium on circuits and systems (MWSCAS). New York: IEEE; 2021, August. p. 544–8.
12. Lotfi N, Tamouk J. A hybrid method based on SA and VNS algorithms for solving DAP in DDS. Comput Sci J Moldova. 2021;86(2):184–205.
13. Singh A. SBBO based replicated data allocation approach for distributed database design. Int J Eng Res Technol. 2020;13(9):2461–73.
14. Chen M, An W, Liu Y, Dong C, Xu X, Han B, Zhang P. Modeling and performance analysis of single-server database over quasi-static rayleigh fading channel. IEEE Trans Veh Technol. 2023;2023:1.
15. Ahmed ZJ, Alluhaibi ST. Hybrid data fragmentation using genetic killer whale optimization-based clustering model. J Pharmaceut Neg Results. 2022;2022:290–8.
16. Che Fauzi AA, Noraziah A, Mohd WMBW, Amer A, Herawan T. Managing fragmented database replication for Mygrants using binary vote assignment on cloud quorum. In: Applied mechanics and materials, vol. 490. Trans Tech Publications Ltd; 2014. p. 1342–6.
17. Castro-Medina F, Rodríguez-Mazahua L, Abud-Figueroa MA, Romero-Torres C, Reyes-Hernández LÁ, Alor-Hernández G. Application of data fragmentation and replication methods in the cloud: a review. In: 2019 international conference on electronics, communications and computers (CONIELECOMP). New York: IEEE; 2019, February. p. 47–54.
18. Castro-Medina F, Rodriguez-Mazahua L, López-Chau A, Abud-Figueroa MA, Alor-Hernández G. FRAGMENT: a web application for database fragmentation, allocation and replication over a cloud environment. IEEE Lat Am Trans. 2020;18(06):1126–34.
19. Castro-Medina F, Rodríguez-Mazahua L, López-Chau A, Alor-Hernández G, Juárez-Martínez U, Sánchez-Ramírez C. An improvement to FRAGMENT: a web application for database fragmentation, allocation, and replication over a cloud environment. In: Proceedings of 6th international congress on information and communication technology. Singapore: Springer; 2022. p. 685–96.
20. Tatarnikova TM, Arkhiptsev ED. Determine the number of distributed Big Data storage replicas. In: 2023 XXVI international conference on soft computing and measurements (SCM). New York: IEEE; 2023, May. p. 223–6.
21. Ortiz-Ballona AO, Rodríguez-Mazahua L, López-Chau A, Castro-Medina F, Abud-Figueroa MA, Rodríguez-Mazahua N. A vertical fragmentation method for multimedia databases considering content-based queries. In: Handbook on decision making: volume 3: trends and challenges in intelligent decision support systems. Cham: Springer; 2022. p. 3–23.
22. Ali A, Naeem S, Anam S, Ahmed MM. A state of art survey for Big Data processing and NoSQL database architecture. Int J Comput Digit Syst. 2023;2023:1.
23. Yang Y, Sun J. Distributed database design and performance Tuing. In: 2023 IEEE 13th international conference on electronics information and emergency communication (ICEIEC). New York: IEEE; 2023, July. p. 1–3.
24. Harikumar S, Ramachandran R. Hybridized fragmentation of very large databases using clustering. In: IEEE signal processing, informatics, communication and energy systems (SPICES); 2015. p. 1–5. https://doi.org/10.1109/SPICES.2015.7091488.
25. Aggarwal M, Bajaj SB, Jaglan V. An improved Vogel's approximation method (IVAM) for fragment allocation and replication in distributed database systems. Indian J Comput Sci Eng (IJCSE). 2022;2022:1.
26. Mahi M, Baykan OK, Kodaz H. A new approach based on greedy minimizing algorithm for solving data allocation problem. Soft Comput. 2023;2023:1–20.
27. Abdalla HI. A brief comparison of K-means and agglomerative hierarchical clustering algorithms on small datasets. In: International conference on wireless communications, networking and applications. Singapore: Springer; 2022. p. 623–32.
28. Di Sanzo P, Quaglia F. On the effects of transaction data access patterns on performance in lock-based concurrency control. IEEE Trans Comput. 2022;2022:1.
29. Ge YF, Wang H, Bertino E, Zhan ZH, Cao J, Zhang Y, Zhang J. Evolutionary dynamic database partitioning optimization for privacy and utility. IEEE Trans Dependable Secure Comput. 2023;2023:1.

30. Mathiason G, Andler SF, Jagszent D. Virtual full replication by static segmentation for multiple properties of data objects. Proc RTIS. 2005;2005:11–8.
31. Sreedhar C, Kasiviswanath N, Chenna Reddy P. Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop. J Big Data. 2017;4(1):1–19.
32. Mosharraf SIM, Adnan MA. Improving lookup and query execution performance in distributed Big Data systems using Cuckoo Filter. J Big Data. 2022;9(1):1–30.
33. Amer AA. On K-means clustering-based approach for DDBSs design. J Big Data. 2020;7(1):31.

Publisher's Note