Journal of Data Mining and Knowledge Discovery
ISSN: 2229–6662 & ISSN: 2229–6670, Volume 3, Issue 1, 2012, pp.-58-64.
Available online at http://www. bioinfo. in/contents. php?id=42

DISTRIBUTED DATABASE: FRAGMENTATION AND ALLOCATION

BHUYAR P.R.[1*] AND GAWANDE A.D.[2]

[1]Computer Science and Engineering Department, Sipna COET, Sgbau Amravati, MS, India.
[2]Department of I.T., Sipna COET, Sgbau Amravati, MS, India.
*Corresponding Author: Email- priyanka.bhuyar@gmail.com

Received: February 21, 2012; Accepted: March 15, 2012

Abstract- The purpose of this paper is to present an introduction to Distributed Databases which are becoming very popular now a days with the description of fragmentation and allocation. Today's business environment has an increasing need for distributed database and Client/server applications as the desire for consistent, scalable, reliable and accessible information is Steadily growing. Data fragementation and allocation are two of the critical aspects of distributed database. The data fregementation and fragement allocation problems in distributed database design are NP-Hard in nature and difficult to solve, which makes developing good solution methods a high priority. Data allocation is typically treated independentally of fragementation. The fragment allocation design is an essential issue that improves the performance of the applications processing in the Distributed Database systems (DDBs). The database queries access the applications on the distributed database sites and should be performed effectively. Therefore, the fragments that accessed by queries are needed to be allocated to the DDBs sites so as to reduce the communication cost during the applications execution and handle their operational processing. We present a method for grouping the sites of the DDBs according to their communication cost in order to determine the fragment allocation to a group of sites instead of allocating the fragments to site by site.
Keywords- Distributed database, data fragmentation, fragment allocation, sites.

Citation: Bhuyar P.R. and Gawande A.D. (2012) Distributed database: Fragmentation and allocation. Journal of Data Mining and Knowledge Discovery, ISSN: 2229–6662 & ISSN: 2229–6670, Volume 3, Issue 1, pp.-58-64.

Introduction

A distributed database is a collection of data that logically belongs to the same system but is spread over the sites of a computer network. A distributed database management system (DDBMS) is defined as the software system that provides the management of the distributed database system and makes the distribution transparent to the users [1, 2]. It is not necessary that database system have to be geographically distributed. The sites of the distributed database can have the same network address and may be in the same room but the communication between them is done over a network instead of shared memory.

The primary concern of DBMS design is the fragmentation and allocation of the underlying database. The distribution of data across various sites of computer networks involves making proper fragmentation and placement decisions. The first phase in the process of distributing a database is fragmentation which clusters information into fragments. This process is followed by the allocation phase which distributes, and if necessary, replicates the generated fragments among the nodes of a computer network. The use of data fragmentation to improve performance is not new and commonly appears in file design and optimization literature.

DDBS Architecture

1. The Hardware

Due to the extended functionality the DDBS must be capable of, the DDBS design becomes more complex and more sophisticated. At the physical level the differences between centralized and distributed systems are:

a. Multiple computers called sites.

b. These sites are connected via a communication network, to enable the data/query communications. Figure 1.1 illustrates this architecture.

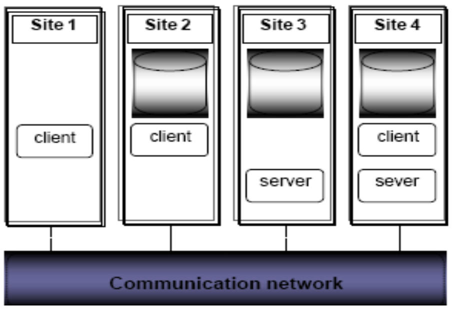

Fig. 2.1- Client/server architecture [1]

Networks can have several types of topologies that defines how nodes are physically and logically connected. One of the popular topologies used in DDBS, the client-server architecture is described as follows: the principle idea of this architecture is to define specialized servers with specific functionalities such as: printer server, mail server, file server, etc. these serves then are connected to a network of clients that can access the services of these servers. Stations (servers or clients) can have different design complexities starting from diskless client to combined server-client machine. This is illustrated in Figure 2.1. The server-client architecture requires some kind of function definition for servers and clients. Th e DBMS functions are divided between servers and clients using different approaches. We present a common approach that is used with relational DDBS, called centralized DMBS at the server level. The client refers to a data distribution dictionary to know how to decompose the global query in to multiple local queries. The interaction is done as follows:

1. Client parses the user's query and decomposes it into independent site queries.
2. Client forwards each independent query to the corresponding server by consulting with the data distribution dictionary.
3. Each server process the local query, and sends back the resulting relation to the client.
4. Client combines (manually by the user, or automatically by client abstract) the received subqueries, and do more processing if needed to get to the final target result.

2. The Software

In a typical DDBS, three levels of software modules are defined:

a. The server software: responsible for local data management at site.
b. The client software: responsible for most of the distribution functions; DDBMS catalog, processes all requests that require more than one site. Other functions for the client include: consistency of replicated data, atomicity of global transactions.
c. The communications software: provides the communication primitives, used by the client/server to exchange data and commands Figure 2.2.

Advantages of Client/Server architecture include: More efficient division of labor, horizontal and vertical scaling of resources, better price/performance on client machines, ability to use familiar tools on client machines, client access to remote data (via standards), full DBMS functionality provided to client workstations, and overall better system price/performance.

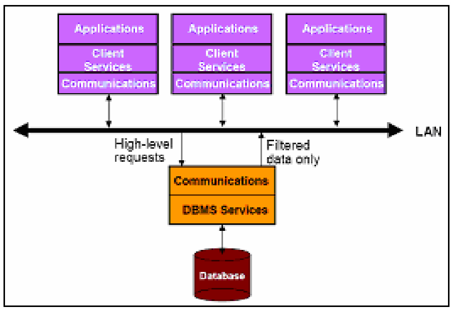

Fig. 2.2- Client/Server Software [2]

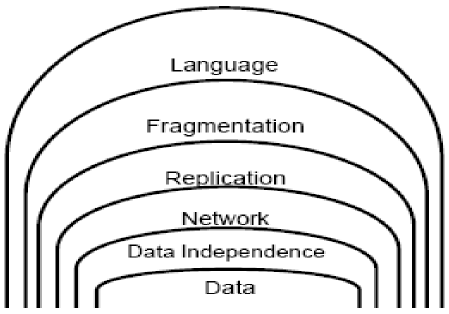

Fig. 2.3- Layers of transparency

Disadvantages of Client/Server architecture include: server forms bottleneck, server forms single point of failure, and database scaling is difficult [2]. It is preferable for a DDMBS to have the property of distribution transparency (Figure 2.3), where the user's can issue a global queries without knowing or worrying about the global distribution in the DDBS.

Fragmentation

Primary concern of distributed database system design is to making fragmentation of the relations in case of relational database or classes in case of object oriented databases, allocation and replication of the fragments in different sites of the distributed system, and local optimization in each site. Fragmentation is a design technique to divide a single relation or class of a database into two or more partitions such that the combination of the partitions provides the original database without any loss of information This reduces the amount of irrelevant data accessed by the applications of the database, thus reducing the number of disk accesses. Fragmentation can be horizontal, vertical or mixed/hybrid.

1. Horizontal Fragmentation

Horizontal fragmentation (HF) allows a relation or class to be partitioned into disjoint tuples or instances. Intuition behind horizontal fragmentation is that Every site should hold all information that is used to query at the site and the information at the site should be fragmented so the queries of the site run faster. Horizontal fragmentation is defined as selection operation, $\sigma\_p(R)$.

Computing horizontal fragmentation (idea)

a. Compute the frequency of the individual queries of the site q1, . . . , qQ

b. Rewrite the queries of the site in the conjunctive normal form (disjunction of conjunctions); the conjunctions are called minterms.
c. Compute the selectivity of the minterms
d. Find the minimal and complete set of minterms (predicates)

- The set of predicates is complete if and only if any two tuples in the same fragment are referenced with the same probability by any application.
- The set of predicates is minimal if and only if there is at least one query that accesses the fragment

e. There is an algorithm how to find these fragments algorithmically (the algorithm CON MIN and PHORIZONTAL (pp 120-122) of the textbook of the course) DDB

An example on horizontal fragmentation is the PROJ table.
Horizontal fragmentation of PROJ relation into
PROJ1: projects with budgets less than 200, 000
PROJ2: projects with budgets greater than or equal to 200, 000

Table 1- PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Development	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Table 2- PROJ1

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Development	135000	New York

Table 3- PROJ2

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Fig. 3.1- Horizontal Fragmentation

2. Vertical Fragmentation

Vertical fragmentation (VF) allows a relation or class to be partitioned into disjoint sets of columns or attributes except the primary key. Each partition must include the primary key attribute(s) of the table. This arrangement can make sense when different sites are responsible for processing different functions involving an entity.
Objective of vertical fragmentation is to partition a relation into a set of smaller relations so that many of the applications will run on only one fragment.

a. Vertical fragmentation of a relation R produces fragments R1, R2, . . . , each of which contains a subset of R's attributes.
b. Vertical fragmentation is defined using the projection operation of the relational algebra: Π _A1, A2,. .., An(R)

Vertical fragmentation of PROJ relation into
PROJ1: information about project budgets
PROJ2: information about project names and locations

Table 4- PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Development	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Table 5- PROJ1

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Development	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris
P5	CAD/CAM	Boston

Table 6- PROJ2

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

Fig. 3.2- Vertical Fragmentation

3. Hybrid Fragmentation

Combination of horizontal and vertical fragmentations is mixed or hybrid fragmentations (MF). In this type of fragmentation scheme, the table is divided into arbitrary blocks, based on the needed requirements. Each fragmentation can be allocated on to a specific site. This type of fragmentation is the most complex one, which needs more management. In most cases simple horizontal or vertical fragmentation of a DB schema will not be sufficient to satisfy the requirements of the applications.
Mixed fragmentation (hybrid fragmentation) Consists of a horizontal fragment followed by a vertical fragmentation, or a vertical fragmentation followed by a horizontal fragmentation. Mixed Fragmentation is defined using the selection and projection operations of relational algebra:
Π_p(_A1,. .., An(R))
Π _A1,. .., An(_p(R))
A fragment of a relation is a relation itself. Fragments can be further fragmented
Projects$_1$ = Π _PNo, PName, Location(Projects)
Projects$_2$ = Π _PNo, Budget(Projects)
Projects$_{1:1}$ = σ _Location='Saarbr.'(Projects$_1$)
Projects$_{2:1}$ = σ _Location='Munich'(Projects$_{1)}$
Projects$_{3:1}$ = σ _Location='Paris'(Projects$_1$)
Projects =
(Projects$_{1:1}$ [Projects$_{1:2}$ [Projects$_{1:3}$) on Projects$_2$

Correctness Rules of Fragmentation

a. **Completeness-** Decomposition of relation R into fragments R1, R2, . . . , Rn is complete iff each data item in R can also be found in some Ri.
b. Reconstruction- If relation R is decomposed into fragments R1, R2, . . . , Rn, then there should exist some relational operator ∇ that reconstructs R from its fragments, i.e., R=R∇.. . ∇Rn
i. Union to combine horizontal fragments

ii. Join to combine vertical fragments.

c. Disjointness- If relation R is decomposed into fragments R1, R2, . . . , Rn and data item di appears in fragment Rj , then di should not appear in any other fragment Rk, k 6= j (exception: primary key attribute for vertical fragmentation)

i. For horizontal fragmentation, data item is a tuple

ii. For vertical fragmentation, data item is an attribute

4. Fragement Allocation

The fragment allocation design is an essential issue that improves the performance of the applications processing in the Distributed Database systems (DDBs). The database queries access the applications on the distributed database sites and should be performed effectively. Therefore, the fragments that accessed by queries are needed to be allocated to the DDBs sites so as to reduce the communication cost during the applications execution and handle their operational processing.

We present a method for grouping the sites of the DDBs according to their communication cost in order to determine the fragment allocation to a group of sites instead of allocating the fragments to site by site. Optimizing the cost of the fragment allocation functions to reduce the queries processing time and determining the fragments to be allocated in the DDBs sites are also main objectives in our research

A. Grouping sites

Grouping sites (clustering) is a method of grouping sites according to a certain criteria to increase the system I/O performance and reduce storage overheads. Grouping sites into clusters helps in reducing the communication costs between the sites during the process of data allocation. We proposed a method for clustering sites according to their communication cost, which determines whether or not a set of sites is assigned to a certain cluster, and it considered as a fast way to determine the data allocation to a set of sites rather than site by site.

Two sites (Si, Sj) are grouped in one cluster if the communication cost between them is less than or equal to a Communication Cost Range (CCR); the number of communication units which is allowed for the maximum difference of the communication cost between the sites to be grouped in the same cluster, this number is determined by the network of the DDBs (Hababeh I. et al. [12]).

Following is the definition of our clustering algorithm:

```
Input: Sites communication cost matrix
CCR value
The sites of DDBs
Output: the set of clusters and their respective sites
Begin
Repeat
For I = 1 to the number of sites in the database
 For J = 1 to the number of sites in the database
 If I ≠ J and communication cost between site I and site J <= CCR then
 Sites I and J are grouped together in the same cluster; set 1 to the cluster entry
 Else
 Sites I and J are grouped in different clusters; set 0 to the cluster entry
 End if
 End for
End for
Until all sites in the database have been processed
End.
```

B. Data (fragment) allocation

To determine the fragment allocation at the DDBs clusters, we propose a new method developed from our approach (Hababeh I. et al. [13]) and based on the data allocation and query processing cost functions that find out precisely whether the fragment is allocated to or omitted from the cluster. This method attempts to minimize the communication costs by distributing the global database over the sites and increasing availability and reliability where multiple copies of the same data are allocated. Initially, fragments are allocated to all clusters having applications using that fragments, and the decision value (D) of allocating a fragment to a cluster is computed as a logical value for the difference bet ween the cost of not allocating the fragment to the cluster and the cost of allocating the fragment to the cluster. If the *Cost of Not Allocating* the fragment to the cluster (the fragment handled remotely) is greater than or equal to the *Cost of Allocating* the fragment to the cluster then the decision value is True and the fragment is allocated to the cluster, and If the cost of not allocating the fragment to the cluster is less than the cost of allocating the fragment to the cluster then the decision value is False and the fragment is cancelled from the cluster.

i. Cost of Allocating a Fragment to a Cluster

The cost of allocating the fragment Fi to the cluster Cj is computed as the sum of the following:

- The average cost of local retrievals at cluster Cj times the average number of frequency of retrieval issued by the transaction T k to the ragment Fi at cluster Cj.
  CLRsum(T k, Fi, Cj) = CLR(T k, Fi, Cj) * FREQLR(Tk, Fi, Cj) (1)

- The average cost of local updates at cluster Cj times the average number of frequency of update issued by the transaction T k to the fragment Fi at the cluster Cj.
  CLUsum(T k, Fi, Cj) = CLU(T k, Fi, Cj) * FREQLU(T k, Fi, Cj) (2)

- The cost of space occupied by the fragment Fi in the cluster Cj times the size of the fragment Fi (in bytes).
  CSPsum(T k, Fi, Cj) = Csp(Tk, Fi, Cj) * Fsize(T k, Fi) (3)

- Remote updates sent from other clusters Cx; the average cost of local updates at cluster Cj times the average number of frequency of update issued by the transaction Tk to the fragment Fi for each cluster other than the current one.
  CRUsum(Tk, Fi, Cj) = CLU(T k, Fi, Cj) * FREQRU(Tk, Fi, Cj) (4)

- Remote communications from other clusters Cx; the update ratio (Unit Update/Unit Communication) times the average number of frequency of update issued by the transaction Tk to

the fragment Fi at the cluster Cj times the average cost of comm unication between clusters other than the current one.
CRCsum(T k, Fi, Cj) = Uratio * FREQLU(T k, Fi, Cj) * CRC(T k, Fi, Cj) (5)

According to the previous formulas the Cost of Allocation CA(T k, Fi, Cj) is defined as the sum of the following costs: local retrievals, local updates, space, remote update, and remote communication.
CA(Tk, Fi, Cj) = CLRsum(T k, Fi, Cj) + CLUsum(Tk, Fi, Cj) + CSPsum(T k, Fi, Cj) + CRUsum(Tk, Fi, Cj) +CRCsum(T k, Fi, Cj) (6)

ii. cost of Cost of Not Allocating a Fragment to a Cluster

The cost of not allocating the fragment Fi to the cluster C j is computed as the sum of the following:
The average cost of local retrievals at cluster Cj times the average number of frequency of retrieval issued by the transaction Tk to the fragment Fi at the cluster Cj. It is the same as defined in previous section 4.1.
Remote retrievals from other clusters Cx; the retrieval ratio (Unit Retrieval/UnitCommunication) times the average number of frequency of retrieval issued by the transaction Tk to the fragment Fi at the cluster Cj for each cluster other than the current one times the average cost of communication between clusters.
CRRsum(T k, Fi, Cj) = Rratio * FREQRR(T k, Fi, Cj) * CCC (7)

According to the previous formulas the Cost of Not Allocation CN (Tk, Fi, Cj) is defined as the sum of cost of local retrievals and sum of cost of remote retrievals.
CN(T k, Fi, Cj) = CLRsum(T k, Fi, Cj) + CRRsum(T k, Fi, Cj) (8)

iii. The Decision Value of Allocating a Fragment to a Cluster

The decision value of allocating the fragment Fi to the cluster Cj is a logical value and computed as follows:
D(T k, Fi, Cj) = (CN(Tk, Fi, Cj) >= CA(T k, Fi, Cj)) (9)

We define our fragment allocation algorithm as follows:

```
Input: Number of transactions issued in the database
Number of fragments used for allocation in the database
Number of clusters used for allocation in the database
Output: The fragments that are allocated to the clusters
Begin
For k = 1 to the number of transactions do
  For i = 1 to the number of fragments do
    For j = 1 to the number of clusters at fragment I do
      CRUsum(T k, Fi, Cj) = 0;
      CRCsum(T k, Fi, Cj) = 0;
      CRRsum(T k, Fi, Cj) = 0;
      For x = 1 to the number of clusters at fragment I do
        If x ? j Then
          CRUsum(T k, Fi, Cj) = CRUsum(T k, Fi, Cj) + CLU(Tk, Fi, Cx) * FREQRU(T k, Fi, Cx)
          CRCsum(T k, Fi, Cj) = CRCsum(Tk, Fi, Cj) + Uratio * FREQLU (T k, Fi, Cx) *
          CRC(Tk, Fi, Cx)
          CRRsum(T k, Fi, Cj) = CRRsum(Tk, Fi, Cj) + Rratio * FREQRR (Tk, Fi, Cx) * CCC
        End if;
      End for;
      CA(Tk, Fi, Cj) = CLRsum(Tk, Fi, Cj) + CLUsum(Tk, Fi, Cj) + CSPsum(T k, Fi, Cj) +
      CRUsum(T k, Fi, Cj) + CRCsum(T k, Fi, Cj)
      CN(Tk, Fi, Cj) = CLRsum(Tk, Fi, Cj) + CRRsum(Tk, Fi, Cj)
      D(T k, Fi, Cj) = (CN(T k, Fi, Cj) >= CA(T k, Fi, Cj))
      If D(Tk, Fi, Cj) = True Then
        Allocate the fragment to the current cluster
      Else
        Cancel the fragment from the current cluster
      End if;
    End for;
  End for;
End for;
End.
```

We illustrate our fragment allocation method in the following example, in which we propose the fragments and their number of frequencies of retrieval and update requested from each cluster and its res pective sites (table 4), the costs of space, retrieval, and update (table 2), and the following number of bytes which required for the computation of the update and retrieval ratios according to their use in the DDBs: 2 bytes in each unit of retrieval, 3 bytes in each unit of update, and 5 bytes in each unit of communication.

Table 1- Fragments and their frequencies of retrievals and updates in the clusters and their respective

Fragment #	Cluster #	Site #	Retrieval Frequency	Update Frequency
F1	C1	S1	80	10
		S2	60	26
	C2	S3	60	16
		S4	0	0
	C3	S5	35	5
		S6	25	5
F2	C1	S3	20	4
		S4	20	6
	C2	S5	5	30
		S6	105	20
F3	C1	S1	0	20
		S2	0	10
	C2	S3	30	0
		S4	0	0
	C3	S5	40	30
		S6	30	10
F4	C1	S1	10	20
		S2	10	20
	C2	S3	65	12
		S4	5	12
F5	C1	S1	70	20
		S2	6	10
	C2	S3	20	10
		S4	20	10
	C3	S5	35	10
		S6	45	20
F6	C1	S1	0	10
		S2	0	0
	C3	S5	25	5
		S6	5	5
F7	C2	S3	25	5
		S4	35	10
	C3	S5	10	0
		S6	30	0
F8	C1	S1	10	20
		S2	80	20
	C2	S3	20	0
		S4	60	10
	C4	S5	0	20
		S6	20	0

Table 2- Cost of space, retrieval, and update

Cluster #	Site #	Cost of space	Cost of Retrieval	Cost of Update
C1	S1	0.004	0.15	0.25
	S2	0.006	0.25	0.35
C2	S3	0.005	0.15	0.25
	S4	0.007	0.17	0.27
C3	S5	0.003	0.13	0.23
	S6	0.005	0.15	0.25

After applying the formulas described in 4.1, 4.2 and 4.3 on the given data, we determine the allocated and cancelled fragments in all clusters. Table 3 describes the allocated and cancelled fragments in all clusters.

Table 3- Allocated and cancelled fragments in all clusters

Fragment #	Cluster #	Cost of Allocation	Cost of not Allocation	Decision value	Allocation status
F1	C1	59.45	177.24	1	Allocated
	C2	74.83	74.76	0	Cancelled
	C3	85.5	74.16	0	Cancelled
F2	C2	74.26	49.84	0	Cancelled
	C3	30.01	135.96	1	Allocated
F3	C1	60.32	0	0	Cancelled
	C2	103.23	37.38	0	Cancelled
	C3	54.72	86.52	1	Allocated
F4	C1	47.13	25.32	0	Cancelled
	C2	68.73	87.22	1	Allocated
F5	C1	86.56	96.21	1	Allocated
	C2	92.66	49.84	0	Cancelled
	C3	86.80	98.88	1	Allocated
F6	C1	15.46	0	0	Cancelled
	C3	18.31	37.08	1	Allocated
F7	C2	7.41	74.76	1	Allocated
	C3	34.71	37.08	1	Allocated
F8	C1	59.22	113.94	1	Allocated
	C2	95.63	99.68	1	Allocated
	C3	80.12	24.72	0	Cancelled

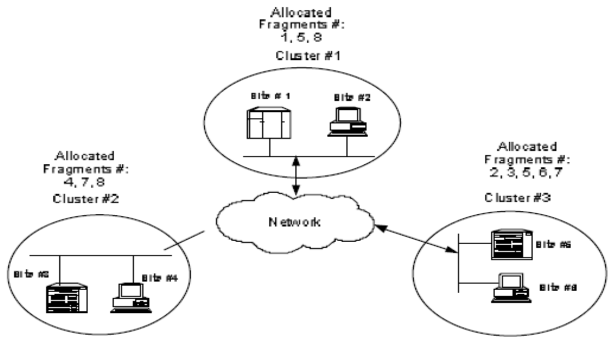

Fig. 4.1- shows the distribution of the fragments over the clusters.

iv. Performance Evaluation

Grouping sites into clusters minimizes the communication costs bet ween the sites and improves the system performance. The average communication cost between clusters and sites, and the average number of retrievals and updates are considered in the computations of our fragment allocation method because the processing time needed for average computations is less than the processing time when other techniques are used which depend on sorting sites according to specific fields. The system performance is enhanced by removing (cancel) the redundant fragments from the database clusters and by increasing availability and reliability where multiple copies of the same fragment are allocated, this will reduce the communication costs where the fragments are needed frequently. Table 7 shows the performance of allocating fragments to the DDBs clusters before and after applying our method.

Table 7- Performance evaluation of fragment allocation

Cluster #	Initial # of alloc. frag.	Final # of alloc. frag.	Improvement %
C1	6	3	50%
C2	7	3	57.14%
C3	7	5	28.57%

Before applying our clustering method, allocating fragments to all clusters having applications requesting those fragments generates 20 allocations, while 11 allocations are generated after applying our clustering algorithm, which improves the system performance by 45.00 %. Figure 4 shows the improvement of the system performance achieved by our clustering and allocating methods on clusters.

Conclusion

We presented an introduction to distributed database design through a study that targeted two main parts: Fragmentation and allocation. Distributed design decides on the placement of (parts of the) data and programs across the sites of a computer network. We also described architecture Fragmentation, allocation also in order to make readers completely aware about the topic
being described here. Fragment allocation method is designed to meet the requirements of clustering sites and determining fragment allocation in distributed database system, minimizing the communication cost between sites, and enhancing the performance in a heterogeneous network environment system.
Clustering method is developed to group the sites into clusters, which helps in reducing the communication costs between the sites during allocation process. Fragment allocation method is developed to enhance system performance by increasing availability and reliability where multiple copies of the same fragments are allocated.

References

[1] Ramez Elmasri and Navathe S.B. (1999) *Fundamentals of Database Systems*.
[2] Tamer Özsu and Patrick Valduriez (1998) *Distributed Database Management Systems*.
[3] Ozsu M.T. and Valduriez P. (1999) *Principles of Distributed Database Systems*.
[4] Ceri S. and Pelagatti G. (1984) *Distributed Databases Principles and System*.
[5] Navathe S., Karlapalem K. and Ra M. (1995) *Journal of Computer and Software Engineering,* 3(4), 395-426.
[6] Ezeife C.I. and Ken Barker (1998) *International Journal of Distributed and Parallel Databases*, 6(4), 327-360.
[7] Kamalaakar Karlapalem, Shamkant Navathe and Magdi Morsi (1994) *Distribut ed Object Management*.
[8] Jin Hyun Son and Myoung Ho Kim (2003) *The Journal of*

Systems and Software.
[9] Wai Wai Gen Yee, Donahoo M.J. and Navathe S.B. (2000) *CIKM*.