American Journal of Applied Sciences 4 (8): 613-618, 2007
ISSN 1546-9239
© 2007 Science Publications

A Dynamic Object Fragmentation and Replication Algorithm In Distributed Database Systems

Azzam Sleit, Wesam AlMobaideen, Samih Al-Areqi, and Abdulaziz Yahya
King Abdulla II School for Information Technology, University of Jordan, Amman, Jordan

Abstract: This paper proposes an enhancement for the ADRW algorithm to achieve dynamic fragmentation and object allocation in distributed databases. The algorithm adapts to the changing patterns of object requests with the objective to dynamically adjust the allocation schemes of objects in order to minimize the total servicing cost of all requests. Objects may be replicated or fragmented depending on patterns of reads and writes. Qualitative analysis was used to characterize the performance of the enhanced algorithm.

Keywords: Distributed Database, ADRW algorithm, Fragmentation, E-ADRW

INTRODUCTION

Distributed systems are an important development in computing technology which is concerned with the delivery of constantly expanding data to points of query. Collections of data in the forms of partitions or fragments can be distributed or replicated over multiple physical locations. Local autonomy, synchronous and asynchronous data distributions are examples of distributed database design schemas which can be implemented depending on business needs and data sensitivity/confidentiality. Data reliability and availability are basic requirements for system design. Reliability is the possibility that a system is running at a certain point in time while availability is the probability that the system is continuously available during a time interval. Both data reliability and availability can be enhanced by distributing data and DBMS software over several sites. The database administrator (i.e.; DBA) carries the responsibility of ensuring that the distributed nature of the system is transparent. Users need not know that they deal with multiple disparate systems, instead of one big repository. Consequently, during the database design process, extra care must be taken into account to minimize the impact of the disconnected nature of the database on the overall performance of the system. For example, join-operation may become prohibitively expensive when performed across multiple platforms. Several research activities were conducted to improve distributed database techniques and to cope with their challenges to solve the abovementioned challenges. The Adaptive Distributed Request Window (i.e. ADRW)[1] is one example of such techniques. ADRW captures the requests for an object and the number of read/write requests with its servicing cost to make the decision concerning replication. A replication may be eliminated depending on the write requests from other requesters in case of excessive overload due to updating replica. In case write requests do not affect the replication, it will be sent to the server that will propagate the update to the others. While working on requests, a server which hosts a particular object may find that another server intensively asks to update the object whereas the hosting server rarely accesses the object. Assuming that the object is not duplicated elsewhere, the server may decide to reallocate this object to the server with intensive requests and announce to other servers to deal with this new allocation. The window mechanism as illustrated in the ADRW algorithm helps to keep track of all transactions taking place between the owner of an object and other servers. This paper proposes an enhanced ADRW algorithm (E-ADRW) which opens two windows per object per requester at the data server. The first window keeps track of requests made for the object by the requesting server and the other window keeps track of requests by other servers. The algorithm uses the windows related to an object to make decisions concerning reallocation of the object whenever necessary. Section 2 discusses related work while section 3 introduces the ADRW algorithm. Section 4 proposes an enhancement for ADRW (i.e. E-ADRW) based on fragmentation. Section 5 analyzes the E-ADRW algorithm and provides a case study example. Section 6 provides a comparison between the E-ADRW algorithm and other dynamic fragmentation algorithms.

The single data allocation problem has been shown to be intractable which means that as the problem size increases, problem search space increases exponentially[5, 6, 7, 8, 15]. Static data allocation implies that no change in data allocation as a function of time, while dynamic data allocation tends to relocate data as necessary[9, 10, 11]. Initial studies on dynamic data

Corresponding Author: Azam Sleit, King Abdulla II School for Information Technology, University of Jordan, Amman

allocation gave a framework for data redistribution [12, 13]. Brunstorm proposed a dynamic data allocation algorithm for non-replicated database system, but no model was proposed to analyze the algorithm[14]. Heuristic approaches to dynamically allocate data were proposed based on data replication [1,2]. Other researches proposed allocation algorithms based on fragmentation for distributed database systems [3, 4].

Fragmentation: A fragment (horizontal, vertical) of a database object in an object-oriented database system contains subsets of its instance objects (or class extents) reflecting the way applications access the database objects. Allocating well-defined fragments of classes to distributed sites has the advantage of minimizing transmission costs of data to remote sites as well as minimizing retrieval time of data needed locally. A re-fragmentation of the data is needed when application access and schema information have undergone sufficient changes. The importance of fragmentation in distributed database and subsequent allocation to distributed sites (relations or classes) has been argued by many works[3]. Most distributed database designs are static based on a priori probabilities of queries accessing database objects in addition to their frequencies which are available during the analysis stage. It is more effective for a distributed system to dynamically check the goodness of an object fragmentation scheme to determine whenever re-fragmentation is necessary.

ADRW Algorithm: The goal of the ADRW algorithm is to dynamically adjust the replication and allocation of objects in order to minimize the total servicing cost of the requests coming to the distributed database system (DDS) [1]. The servicing cost is defined to consist of three components as follows:

C_c: Cost of sending the query for the object from the requesting (i.e. non-data) processor to the (i.e. data) processor.

$C_{i/o}$: Cost of fetching/updating the object to/from the local memory of the processor that hosts the object [Assumed to be one unit of time].

$C_{d:}$ Cost of transferring the object from the main memory of the hosting (i.e. data) processor to the requesting (i.e. non-data) processor.

S(o): Initial allocation servers for object *o*.

The processor is considered a data processor for a particular object if the object is hosted in the local memory of the processor. All other processors are non-data processors for the object. Assuming we have three processors p_1, p_2, and p_3 and p_2 is the data processor for object *o*. The cost for p_2 to access object *o* is one unit of time. Moreover, p_2 will create a k-bit size window corresponding to object o. For every new request coming to p_2 for object o from p_1, a 0 is added to Win(o, p_1), while a 1 is added to Win(o, p_3) for every new request coming to p_2 from p_3 for object o. If another process, say p_3, is writing to the object *o*, then *p2* will add 1 to the window. So, if the number of read, *Nr*, from *p1* is greater than the write, *Nw*, from *p3*, then *p2* will make a replication for *o* to p_1 with its window and add p_1 to the *data_list(o)* which is a list of all the processors that have a replica of the object *o*. p_1 now is a data processor. It will save the object in its local memory and access it directly. If any write to the object arrived to p_2 then it will update the object and send the update to all the processors that hold the object found in the *data_list(o)*. Now, if processor p_1 reads the object, it will add 0 to the window and 1 if others write to it. If the number of writes is greater than the number of reads, then it will delete the replication and return the window to the owner processor p_2.

E-ADRW Algorithm: In the ADRW algorithm, read requests play an important role in deciding whether to replicate an object or remove a replica of it. On the other hand, the E-ADRW algorithm handles write transactions with special care. Let's say that processor p_i is a non-data object processor for object *o* and wants to write to it. A write request is issued to the nearest processor in S(o). Assume it is processor p_j. When p_j receives a write request, it first checks if it is the first request in order to create the request windows for p_i. The E-ARDW algorithm creates two-windows for object o. *Win1(o, p_i)* is created for processor p_i of size k_1 bits to record read and write requests (N_{ir} of p_i is the number of read requests and N_{iw} is the number of write requests). Also, *Win2(o, p_a)* of size k_2 bits is created to record the reads and writes from the other processors. When p_i sends a write request to p_j, (1 + Cd) time is needed as per the ADRW algorithm (Cd units is needed to transfer the update from pi to p_j and Ci/o to locally write the object). As p_i writes to object *o*, there may be other processors which write to the same object and have similar effect but C_d is different from one processor to another. When p_i increases the frequency of writes to object *o* while the need of this object is rare by other processors, then it is better to give pi the whole object as a fragment to reduce overhead and cost. This is assured if the object is not replicated by checking the R_data_list(o) which holds the id's of the processors that have a replication of the object.

The algorithm checks if

$$(1+Cd1)Niw >> (1+Cd2)Naw + Nar \quad (3)$$

then, p_j will consider p_i as a write-intensive node. In this case, p_j will add p_i to the fragment data list F_data_list(o) which includes the list of processors where the object is fragmented and sends the object to p_i as a fragment. In addition, the E-ADRW algorithm sends the new allocation of object o to all the processors through broadcasting which will change the allocation to p_i and delete the copy of o from p_j as well as from other locations. The algorithm is illustrated in figure 1.

```
Algorithm: Write Request
If (Req is Write request for object $o$)
{If ($p_i == p_j$) {EXIT ;} // Write is locally
  Else
  {If (Req is the first request from $pi$)
    {Generate:
          an initial Wind1 ($o, p_i$);
          an initial Wind2 ($o, p_a$);
    }
    Insert 1 into Wind1 ($o, p_i$)
    Insert 1 into Win2 ($o, p_b$) ;
     // inserted into all existing windows
    // for object $o$ in pj Except Wind2 (o, $p_a$)
   If $(1+Cd1)Niw >> (1+Cd2)Naw + Nar$
   {If (o not in R_data_list ($o$))
          Fragment ($o, p_i$) {
        Send $o$ to $p_i$;
          Add processor pi into
          F_data_list ($o$);
        Delete $o$ from $p_j$
        Bcast ($o, p_i$) // send the new allocation
                    // to others.
          }
    }
}
```

Fig. 1: Write Request

The purpose of the broadcast function (i.e. Bcast) is to notify all other processors about the new allocation.

```
Bcast ($o, p_i$)
{
      Add the new allocation $p_i$ of object $o$ to $S (o)$;
      Delete the old allocation;
}
```

The E-ADRW algorithm handles read requests following the same mechanism of the ADRW algorithm as illustrated in figure 2. The difference is that the E-ADRW algorithm gives priority to fragmentation. When the replication condition is the algorithm checks the fragmentation condition. If the fragmentation condition is also satisfied, fragmentation occurs rather than replication. However, if the replication condition is satisfied while fragmentation condition is not, replication occurs.

```
Algorithm: Read Request
If (Req is read request)
 {If ($p_{is} == p_j$) {EXIT ;}
   Else
    {If (Req is the first request from $p_i$)
          {Generate:
                    an initial Wind1 ($o, p_i$);
                    an initial Wind2 ($o, p_a$);
          }
     Insert 0 to Win1 ($o, p_i$);
     Insert 0 to Win2 ($o, p_b$);
     If ((Cc + $C_d$) $N_{iR} \geq (1 + C_d) N_{aW}$)
          If $(1 + Cd1)Niw >> (1 + Cd2)Naw + Nar$ And
             ($o$ not in R_data_list ($o$))
                    Fragment ($o, p_i$)

          Else {
            Indicate $pi$ to enter $Ao$;
            Add processor $pi$
            into R_data_list ($o$);
            Transfer Wind1 ($o, p_i$)
            and Wind2 ($o, p_a$) to $p_i$;
            Delete Wind1 ($o, p_i$)
            and Wind2 ($o, p_a$) in $p_j$;
            $p_i$ saves object $o$
           and changes its relative
           status to 1; /*data processor*/
          }    }}
```

Fig. 2: Read Request

Object Re-allocation: The status of the object after replicated or fragmented in the new processor p_i will be:

- If the object is replicated then as illustrated in[1], it will follow the TEST-and-EXIT algorithm in the processor p_i so the object may exit the allocation.
- If it was fragmented then pi will act as the owning processor; i.e. p_i will apply the E-ADRW algorithm and re-allocate the fragment or replicate the object/fragment to other processors who have the priority and so on.

Cost Model: In this section we discuss the cost model of the E-ADRW algorithm for the servicing costs of the read and write requests. While this algorithm is following the same mechanism of the ADRW algorithm, the model is the same except that we add the fragmentation cost in case of write requests. As presented in the ADRW algorithm[1], the servicing read request is:

$$\text{Cost}_{\text{E-ADRW}}(R_o^{pi}) \quad (1)$$

$$= \begin{cases} 1 & \text{if } p_i \in A_o \\ 1 + Cc + Cd & \text{if } p_i \notin A_o, \text{! replica or fragm.} \\ 2 + Cc + Cd & p_i \notin A_o, R_o^{pi} \textit{ is } \text{saving - read} \end{cases}$$

A_o is the allocation scheme. The first case is when p_i is a data processor where the object is either fragmented or replicated. In the second case, the object is not replicated or fragmented and p_i is non-data processor. The cost is 1 unit for I/O servicing and $C_c + C_d$ for control and data transfer costs respectively. The third case is when p_j makes a replica or fragment for the object in p_i. Therefore, the cost is 1 unit for saving the object in the local memory of p_i. Consequently, p_i becomes a data processor and acts as per the first case. The servicing cost for the write request is not as in ADRW algorithm. In our algorithm we focus on the effect of the write transaction which causes fragmentation. As in [1], we considered the servicing cost of a write request with an allocation scheme A_o on the request and $|A_o^`|$ for the allocation scheme after servicing the request. The cost is as follows:

Cost $_{\text{E-ADRW}}$ (W_o^{pi})

$$= \begin{cases} 1 & p_i \in A_o \text{ fragmented} \\ |Ao|C_d + |A_o^`| & p_i \notin A_o \text{ and not replicated or fragmented} \\ (|A_o|\text{-}1)\, C_d + |A_o^`| & p_i \in A_o \text{ and replicated} \end{cases} \quad (2)$$

When the object is in p_i and it issues a write request, it must be propagated to all the servers that hold the object except pi itself. The cost of saving the object in their local memory locations is $|A_o^`|$. If the object is not replicated or fragmented in pi then the cost is the data transfer cost to all the servers in A_o. The last situation occurs when the object is fragmented then it is just cost of I/O which is 1 unit. The summary of the cost model is in the table 1.

Table1: Cost of read and write for the E-ADRW algorithm

	Non-replicated	)bject Replicat	Object Fragmente
Read requests p_i	$1+C_c+C_d$	1	1
Write requests p_i	$\|A_o\|C_d + \|A_o^`\|$	$(\|A_o\|\text{-}1)C_d + \|A_o^`\|$	1

E-ADRW Analysis: In[1], competitive analysis is used to quantify the performance of the ADRW algorithm. It stated and proved that the ADRW is $[k + \frac{1+C_c+2C_d}{C_d} + \frac{C_d}{t}]$ competitive, where t is the object availability. In the E-ADRW algorithm, write requests are used to make decision whether fragmentation is needed or not. So, we extend the algorithm by adding a specific condition for achieving fragmentation. Following the same analysis used in ADRW, the following can be concluded:

$$K_1 = N_{ir}+N_{iw} \quad (4)$$
$$K_2 = N_{ar}+N_{aw} \quad (5)$$
$$K_3 = N_{ir} + N_{aw} \quad (6)$$

The fragmentation condition is exactly the same as (3) as follows:

$$(1+C_{d1})\, N_{iw} >> (1+C_{d2})\, N_{aw} + N_{ar} \quad (3)$$

From (4), (5) and (6)

$$N_{ar} = K_2 - N_{aw}$$
$$N_{aw} = K_3 - N_{ir}$$
$$N_{ir} = K_1 - N_{iw}$$

Using (3) and substituting the above equations,

$(1+C_{d1})\, N_{iw} >> (1+C_{d2})(K_3 – (K_1 - N_{iw}))+ K_2 – (K_3 – (K_1 - N_{iw})),$ *equivalently*
$(1+C_{d1})\, N_{iw} >>(1+C_{d2})(K_3–K_1) + (1+C_{d2})\, N_{iw} + K_2 –K_3 + K_1 - N_{iw}$, *equivalently*
$(1+C_{d1})\, N_{iw} + N_{iw} -(1+C_{d2})\, N_{iw} >> (1+C_{d2})(K_3–K_1) + K_2 –K_3 + K_1$

Therefore,

$$N_{iw} >> \frac{(C_{d2} (K_3-K_1) + K_2)}{(1+ C_{d1} - C_{d2})} \quad (7)$$

Using the condition in (7), the fragmentation of object o to processor p_i may occur. The total cost before object o is fragmented on p_j is as follows:

$$\text{Cost}_{\text{E-ADRW}} (W_o) = (1+C_{d1})\, N_{iw}+(1+C_{d2})\, N_{aw} + N_{ar} \quad (8)$$

where, $(1+C_{d2})\, N_{aw}$ is the total cost of all the other processors and computed as ($N_{aw} + \sum C_{d2}$). The servicing cost of the fragmented object is reduced. As mentioned in the cost model, when the processor is a data processor where the object is fragmented, the cost locally is a unit cost. Using (8),

$$\text{Cost}_{\text{E-ADRW}} (W_o) = N_{iw} + (1+C_{d2})\, N_{aw} + N_{ar} \quad (9)$$

This is clearly less than the servicing cost before fragmentation. In case of read requests, the total servicing cost before replication is similar to that of the ADRW algorithm[1].

$$\text{Cost}_{\text{E-ADRW}} (R_o) = (C_c + C_d)\, N_{ir} + (1+Cd)\, N_{aw} \quad (10)$$

Example: In this section, we present an example which illustrates the impact of applying the E-ADRW algorithm on reducing service cost. Assume that there are six processors located in Syria, Iraq, Kuwait, Riyadh, Jeddah, and Jordan as per figure 3. The processor in Jordan is the data processor for object o. Let the cost of transferring data (C_d) from Jordan to other requesting processors be as follows: Syria= 12, Iraq= 4, Kuwait= 7, Riyadh= 8 and Jaddah= 10 units. Processor Syria (p_i) requests Jordan (p_j) for object o. Jordan creates the two windows win1 and win2. After k_1 requests by Syria, the following is obtained using (4), (5), and (6):

$$N_{ir} = 30,$$
$$N_{iw} = 75,$$
$$N_{ar} = 10, \textit{and}$$
$$N_{aw} = 35$$

Table 2: The servicing cost using replication and fragmentation

Result	**Reduction Ratio**	**TCAF**	**TCBF**	**Reduction Ratio**	**TCAR**	**TCBR**	N_{aw}	N_{ar}	N_{iw}	N_{ir}
Fragmentati will occur	67%	376	1125	---------	--------	Replication condition is not satisfied	35	10	75	30
Replication will occur	------------	----------	Fragmentation condition is not satisfied	61%	420	1105	35	10	30	75
Fragmentati will occur	79%	580	2730	66%	600	1750	50	50	200	100
Fragmentati will occur	91%	83	956	75%	96	382	8	12	80	10

TCBR: Total Cost Before Replicated, TCBF: Total Cost Before fragmentation, TCAR: Total Cost After Replication, TCAF Total Cost After Fragmentation

Table3: Comparison between E-ADRW and Other Algorithms

	Optimal Algorithm	**Threshold Algorithm**	**E-ADRW Algorithm**
Mechanism	Creates an array for each fragment in its associated node (server) whenever the fragment migrates its associated counters	Creates an array for each fragment in its associated node with only one counter per fragment	Creates two windows for each object requested by a remote processor
Fragmentation condition	Counter for remote node is greater than that counter of the owning node	Counter associated with the fragment greater than the threshold value	Number of write requests for remote node is much greater tha all other node requests
Cost	- Extra storage space is needed for the access counter matrix - Generates overhead on the network because of the condition for transferring the fragment	- Extra storage space is needed (less than Optimal algorithm) - Generates overhead on the network because of the condition for transferring the fragment	- Space is only needed for the fragment which has write remote access - The fragmentation condition more strict

When Syria issues a read request for the object, Jordan will decide based on the replication and fragmentation conditions (i.e. $((Cc + C_d)\, N_{iR} \geq (1 + C_d)\, N_{aW}))$, $(1+C_{d1})\, N_{iw} >> (1+C_{d2})\, N_{aw} + N_{ar}$ respectively) whether to give Syria a replica or fragment. Evaluating the replication condition 360 is not greater than 385. However, if the request is a write request, Jordan will check for the possibility of fragmenting *o* to processor Syria. Evaluating the fragmentation condition results in 825 is much greater than 300.

The total service cost before fragmentation using (8),

$$\text{Cost}_{E\text{-}ADRW}(W_o) = (1+C_{d1})\, N_{iw} + (1+C_{d2})\, N_{aw} + N_{ar}$$

$$\text{Cost}_{\text{E-ADRW}}(W_o) = 1125 \text{ units}$$

The service cost after fragmentation using (9),

$$\text{Cost}_{\text{E-ADRW}}(W_o) = N_{iw} + (1+C_{d2})\, N_{aw} + N_{ar}$$

$$\text{Cost}_{\text{E-ADRW}}(W_o) = 375 \text{ units}$$

In table 2, we summarize various combinations of N_{ir}, N_{iw}, N_{ar}, and N_{aw} of Syria to demonstrate reduction in service cost as a result of using the E-ADRW algorithm.

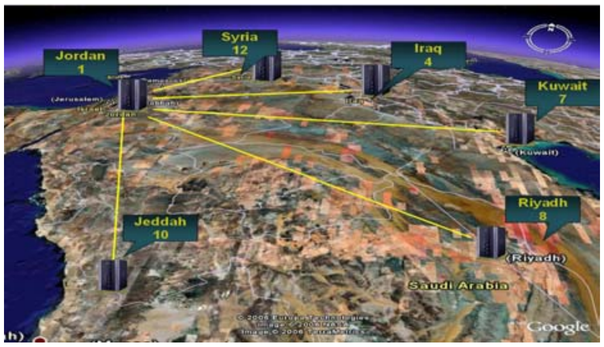

Fig. 3: Example for the connection to processor Jordan

Comparison analysis: Few algorithms are currently available for dynamic object fragmentation in distributed database systems. The Optimal algorithm utilizes an mXn matrix, where *n* denotes the number of servers and *m* the number of fragments. It offers a solution based on counting number of accesses of each server for a fragment. A row of the matrix shows the access counts of all nodes to a particular fragment, whereas a column shows the access counts of all fragments for a particular node[2]. For each fragment, the server with the highest access counter value is the current owner server of the fragment in which case the fragment is stored in this server. The potential storage requirements and the static allocation for the local fragments at the first step with counter 0 are two major drawbacks.

The threshold algorithm overcomes the Optimal algorithm, where only one counter per fragment is stored to decrease the potential storage. The counter value is increased for each remote access to the fragment and reset to zero for a local access. The counter reflects the number of successive accesses. When the counter exceeds the threshold value, the owner transfers the fragment to the last server which accessed the fragment. Compared with our algorithm, the window is created when the first request is issued. Therefore, the storage is potentially less than that required by the Optimal and Threshold algorithms as per table 3. The E-ADRW algorithm takes into account the type of operation (read/write) when deciding replication versus fragmentation. On the other hand, the threshold algorithm grants the fragment to the last processor accessing the fragmented object, whereas the E-ADRW algorithm gives the priority to the processor with the largest number of requests. Furthermore, the E-ADRW algorithm may replicate an object under certain conditions.

CONCLUSION

This research presented an enhancement for the ADRW algorithm by allowing dynamic fragmentation of objects. The E-ADRW algorithm is studied and evaluated in terms of the effect of write requests. The advantage of the E-ADRW algorithm is that it requires less storage when compared with the Optimal and threshold algorithms. The reduction in servicing cost of the E-ADRW algorithm due to replication and fragmentation was analyzed.

REFERENCES

1. Wujuan, L. and Veeravalli, B, 2003. An Adaptive Object Allocation and Replication Algorithm In Distributed Databases, IEEE Proc. 23rd Int'l Conf. Distributed Computed System.
2. Ulus, T. and Uysal, M, 2003. Heuristic Approach to Dynamic Data Allocation in Distributed Database Systems, Pakistan Journal of Information. And Techn., Asian Network for Scientific Information.
3. Ezeife, C.I. and Zheng, J, 1998. Measuring the Performance of Database Object Horizontal Fragmentation Schemes, Supported by NSERC of Canada.
4. Menon, S, 2005. Allocating Fragments in Distributed Database, IEEE transactions on parallel and distributed systems, 16:577 – 585.
5. Porcar, H, 1982. File Migration in a Distributed Computing System, PhD thesis, University of California.
6. Daudpota, N, 1998. Five steps to construct a model of data allocation for distributed database system, Journal of intelligent information systems, 11:153-168.
7. Apers, P, 1988. Data allocation in distributed database systems, ACM transactions on database systems, 13: 263-304.
8. Ceri, S, Navathe, S. and Gio, 1983. Distributed Design Of Logical Database Schemas, IEEE Transactions on Computers, SE-94:487-504.
9. Theel, O. E. and Pagnia, H, 1996. Bounded Dynamic Data Allocation in Distributed Systems, The 1996 3rd International Conference on High Performance Computing, pp. 126-131.
10. Wolfson, O, Jajodia, S. and Huang, Y, 1997. An Adaptive Data Replication Algorithm, ACM Transactions on Database Systems, 22(2):255-314.
11. Levin, K.D., and Morgan, H.L, 1982. A Dynamic Optimization Model for Distributed Databases, Operations Research, 26(5):824-835.
12. Wilson, B. and Navathe, S. B, 1986. An Analytical Framework for the Redesign of Distributed Databases, Proceedings of the 6th Advanced Database Symposium, Tokyo, Japan, pp. 77-83.
13. Rivera-Vega, P.I., Varadarajan, R. and Navathe, S.B, 1990. Scheduling Data Redistribution in Distributed Databases, IEEE Proceedings of the Sixth International Conference on Data Engineering, 166-173.
14. Brunstrom, A, Leutenegger, S.T. and Simha, R, 1995. Experimental Evaluation Of Dynamic Data Allocation Strategies In A Distributed Database With Changing Workloads, Proceedings of the 1995 International Conference on Information and Knowledge Management, Baltimore, MD, USA, pp. 395-402.
15. Ahmad, I, Karlapalem, K, Kwok, Y. K. and So, S. K, 2002. Evolutionary Algorithms for Allocating Data in Distributed Database Systems, Distributed and Parallel Databases, 11: 5-32.