Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Computer Standards & Interfaces 27 (2004) 1–18
COMPUTER STANDARDS & INTERFACES
www.elsevier.com/locate/csi

A distributed queue approach to resource locations in broadband distributed computing environments

G. Gabrani*

Department of Computer Engineering, Delhi College of Engineering, Bawana Road, New Delhi 110042, India

Received in revised form 5 February 2004; accepted 13 February 2004
Available online 11 March 2004

Abstract

In broadband distributed computing environments, the importance of data resource (DR) migration is increasing because of its potential to improve performance of the system, especially for transaction processing. In such environments, mobile data resources relocate themselves from one computer site to the other. Hence, it is important to have mechanisms that manage the locations of each data resource. In this paper, a location management algorithm is presented in which sites are logically organized as multiple rooted tree structures. The rooted tree structures are used to move the requests for locating the data resources. The algorithm makes use of a distributed queue strategy to define the path a data resource takes while migrating from one site to the other. The proposed algorithm enhances its effectiveness by continuously updating its information regarding the site to which the request is to be forwarded so as to reduce the number of messages needed by the requests to locate and access the data resources. The performance of the proposed algorithm is evaluated and is also compared with one of the existing location management algorithms by simulation studies under several system parameters such as frequency of requests generation, frequency of data resource migrations and network topology. The experimental results show the effectiveness of the proposed algorithm in all cases.

Keywords: Distributed computing; Distributed queue; Data resource migration; Transaction processing

1. Introduction

A distributed system is composed of multiple geographically dispersed computer sites connected via a communication network. These sites do not share a common memory but communicate with each other by sending messages over the network. Each computer site has its own memory, processing and communication capabilities as well as the necessary resources. The resources considered in this paper are data resources (DRs) such as databases and files. The DRs owned and controlled by a site are said to be local to it, while the DRs owned and controlled by other sites as well as those that can be accessed through a network are said to be remote. With the availability of appropriate communication network and control protocols, the users and applications have an access to local and remote system resources in an integrated manner. One of the methods, normally employed in traditional (narrowband) networks, to

* FB-26, Tagore Garden, New Delhi 110027, India. Tel.: +91-9811047227; fax: +91-01125540951.
E-mail address: ggabrani@yahoo.co.in (G. Gabrani).

0920-5489/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2004.02.001

access remote DRs is fixed processing. In this method, DRs are fixed at a particular computer site and all other sites access the DR at that site by sending operation and control messages to it. Another method called DR migration method [8,9] makes use of recent broadband environments. In this method, all the DRs required by a transaction are migrated to the transaction initiation site so that transaction initiation site can access the DRs locally without sending any operation requests over the network. Therefore, the sites where the DRs are migrated have the advantage of quick and inexpensive retrieval of information, whereas getting the information from remote sites by fixed processing incur communication costs and transmission delays. Let us take an example of an airline reservation system [4]. A few hours before and after a flight stops at an airport, a surge in access activity from airport to the airline DR is observed. During those periods, if the requested DR is remotely located, huge volume of long distance communication traffic is generated (for example intensive queries and updates of reservation, ticketing, crew, fare information, baggage handling, etc.) and this results into the deterioration of communication services and increase in overall operating costs. When the amount of traffic is substantial enough, the system can benefit from migrating the DR to the cities at which a flight stops a few hours before the flight arrives there. However, in practice, migration of DRs is possible with the use of broadband networks, because broadband networks are different from traditional networks in that the propagation delay in the latter is small as compared to the transmission delay. As an illustration, for example, propagation delay across a country say United States at the speed of light is about 20 ms [2]. In broadband networks, at say 1 Gb/s, it will only take 1 ms to transmit a 1-megabit file, resulting in a total delay of 21 ms. For a traditional network, like the internet, operating at 50 Kb/s, the transmission delay is 20 s, giving a total delay of 20,020 ms. Therefore, in conventional narrowband networks, most of the existing algorithms have focused on minimizing the volume of data to be transmitted, whereas in broadband networks it has become possible to transmit a great volume of data in a very short period of time. From this viewpoint, migration of DRs has become more viable and is expected to be one of the most useful mechanisms in broadband distributed systems. The use of DR migration in distributed systems has been demonstrated in Refs. [1,4,5,11–14,20]. As already mentioned, one of the motivations to use DR migration method is to trade transaction throughput with the available bandwidth. It has been shown in Ref. [9] that the use of DR migration mechanism highly contributes to the performance improvement in transaction processing in broadband networks such as ATM and hence can be considered as one of the primitive DR operations. Further, the researches in Refs. [2,6] have suggested the techniques to utilize the advantages of broadband networks for transaction processing. Some of the other applications of mobile DRs include replication and allocation of DRs to various sites so as to reduce communication costs and to increase reliability [10], query processing strategies for high speed local area networks [24,25] or for load balancing among distributed file servers by relocating the distributed data [15,16,19,22,23]. (It may be pointed out here in these studies, as broadband networks are not assumed, the DR migration operations are executed in a limited controlled manner.) In addition, DR migration may also reduce system operating costs because different computer sites operate under different operating cost scales, differences in time zone permit the use of regular shift operations at remote sites instead of local late shifts, and remote computer sites are used during lean periods rather than local resources within peak hours [4]. Thus, DR migration operations have a very high potential to improve system performance and reduce system costs appreciably. Taking the above into account, the developments of distributed systems now incorporates DR migration operations as well as fixed processing as options for accessing remote DRs. Therefore, mobile DRs that move among various computer locations are emerging as a new form of building distributed network-centric applications. As DRs move from one site to the other, their locations keep on changing. Thus, deriving efficient strategies for managing the locations of mobile DRs (i.e. identifying their current locations) is an important research issue. Several such location management algorithms have been proposed in the literature [7,21]. In Ref. [21], authors have proposed a new location searching algorithm and, in Ref. [7], Hara et al. have suggested six location management methods for mobile DRs in broadband environments. Four out of these six (DF (Default

Forwarding), DQ (Default Query), CF (Chain Forwarding), CQ (Chain Query)) have been adopted from mobile computing [17]. The other two, ECF (Extended Chain Forwarding) and ECQ (Extended Chain Query) are improved versions of CF and CQ, respectively. In Ref. [7], it has been demonstrated through simulation experiments that the ECF algorithm gives the best performance among various distributed algorithms under different system parameters. The ECF algorithm is now explained below.

In ECF, the requesting site generates a request message to locate a DR (the DR to be located is called target/requested DR). The ECF algorithm ensures that a request message is forwarded successively along the migration track of the target DR, i.e. the chronological sequence of the sites at which the target DR has resided. This is achieved by maintaining a location table at each site. The location table is local to each site and the information in this table is only modified when a site either receives a request message, a DR or sends a DR. For example, whenever a DR migrates from, say, site i to site j, the location tables at both sites i and j are modified to point to site j, the new location of the DR. The location table at each site records the last known location of all the DRs along with their migration counts. The migration count of DR increments by one every time it migrates from one site to another. Therefore, the value i of migration count means that the corresponding location information represents the location of the DR after ith migration. Hence, newer information about the location of a DR can be recognized by comparing its migration counts. Whenever a requesting site intends to locate a DR, it sends a request message to a site according to its own location table. If a site receiving the request message does not hold the target DR, it forwards the request message to another site according to its own location table. The request message also carries the contents of the location table of the requesting site. At every site visited by the request message, migration counts of each DR given by the location table of both request and the site are compared and older values are replaced with newer values. This process of successive message forwarding continues till the request message reaches the target site (the site holding the target DR). The target site then sends the DR to requesting site and the entry regarding the current location of the target DR in location tables at both the target site and the requesting site is updated to requesting site and the migration count is updated to the value one higher than that of target site. In addition, ECF also uses another message called update message that is generated by the target site on the receipt of a request message. Each update message carries the contents of location table of the target site. This message is sent backwards from the target site to every site through which the corresponding request message has passed. It updates the location tables at all the sites it visits with the contents of location table it carries (updation is performed in a similar manner as by request message).

In this paper, an algorithm is proposed that makes use of a distributed queue data structure for the location management of mobile DRs. The design of the proposed algorithm is motivated by Ref. [3], in which Chang et al. have proposed an algorithm that employs distributed queue data structure to achieve mutual exclusion. The algorithm by Chang et al. is modified to fit the location management method for DRs in broadband networks. In the proposed algorithm, all the n sites of the system are configured as m (total number of DRs) logical trees. In other words, one logical tree corresponding to each DR is maintained in the system; therefore, there are m logical trees (one corresponding to each DR) in the system. The root of the tree holds the DR when no other site in the system requests for the migration of the DR. Also, the root is the last site among the current requesting sites to receive the DR when no message is in transmission. Each site points to the site where the request is to be forwarded for locating the DR and every requesting site only records the requesting site next to it to receive the DR; therefore the size of each local queue that stores the request for migration (migratory request) of a DR is one. This is in contrast with the algorithms where every requesting site adds all the incoming migratory requests in its queue. Therefore, in such algorithms, the size of the queue that stores the migratory requests, at each site has to be expanded to n (the number of the sites in the system). In the proposed algorithm, whenever a requesting site needs to access a DR, it sends a request to the site possibly holding the DR. Whenever a site receives a request, it forwards this request to another site possibly having the DR. After successive forwarding, the request message reaches the root site. (It

may be noted here that the request message in the proposed algorithm can be forwarded to $(n-1)$ sites in the worst case.) From the root site, the DR is then migrated to the site requesting for the DR. The migration of the DR from one site to the other takes place via the entries given by the local queues. Whenever a DR migrates, it also carries with itself the information of the site, which is pointed by the site from where it migrates. Finally, the performance of the proposed algorithm is evaluated and is compared with ECF. As ECF algorithm gives the best performance among various distributed algorithms [7], the proposed algorithm is compared with ECF. The improved performance of the proposed algorithm in terms of message traffic for two network topologies (binary tree and star) is demonstrated through simulation experiments. Moreover, in order to be widely used in system applicability, it is shown that the proposed algorithm has strong adaptability to different network topologies and generally enhances its effectiveness with greater connectivity.

The rest of the paper is structured as follows: In Section 2, the system model on which the proposed algorithm has been developed is described. Section 3 gives the basic idea of the algorithm. The working of the algorithm is illustrated with the help of an example in Section 4. A detailed description of the algorithm is given in Section 5. The algorithm is analyzed in Section 6. In Section 7, the simulation model and results are presented. Finally, some concluding remarks are given in Section 8.

2. System preliminaries

The distributed system considered in this paper consists of n homogeneous sites and m DRs with low location dependency, i.e. DRs can move without restrictions between the sites involved in the system. The sites are connected via an ATM network and general ATM environments similar to that used in Ref. [7] has been assumed. The communication network is assumed to be reliable (i.e. messages are neither lost nor duplicated and are transmitted error free) and sites do not crash. Whenever site i sends a message to the site j, the message is routed through intermediate ATM switches by making a Switched Virtual Connection (SVC). SVC is established dynamically according to the request generated by the site. Sometimes, overloading in the network causes congestion in the network, during which SVC is released. SVC is also released if connection has not been used for a predefined period of time.

All the sites in system can have an access to any of the m DRs. The sites can access these DRs either by fixed processing method or by DR migration method. Each site contending for the DR has equal priority and no central control is supported. Any site that wants to access a DR generates a request message and communicates it to the other site. The request message can be of two types: (i) access request message and (ii) migratory request message. In case of an access request message, the transaction initiation site requests the target site to allow it to perform operations on the DR by fixed processing method (on the site where the DR resides). Whereas in case of a migratory request message, the transaction initiation site requests the target site to migrate the DR to it. In response to this request, the DR is migrated from target site to the transaction initiation site and the operations on the DR are then performed at the transaction initiation site. A basic idea of how the proposed algorithm handles the request messages and the migration of DRs is now given in Section 3.

3. Basic idea

In the proposed algorithm, all the n sites of the system are configured as m logical trees (one logical tree corresponds to one DR). All directed edges of a tree point towards the root of the tree. The root holds the DR if no other site of the system is requesting for the migration of the DR and also is the last site to receive the DR, when no message is in transmission. Each site maintains a State Information (SI) table that stores the current state of the site with respect to all the DRs of the system. Each site can be in a requesting state for a DR, when the site has generated either an access or a migratory request for a DR; a non-requesting state for a DR, where the site does not hold the DR and does not produce any access or migratory request; or in a holding state for a DR, when the site holds the DR. Each site also maintains a father table that records the value of father pointers of that site with respect to all the DRs of the system. The

father pointer of a site for a DR points to the site that possibly holds the corresponding DR. Therefore, a requesting site i will send the request message for DR x to site j only if father pointer of site i for DR x points to site j. Further, each site maintains the following two queues for each of the m DRs of the system:

(i) Migratory_Request (M_R) queue: this queue stores the site_id of the site to which the corresponding DR would be migrated after the DR holding site finishes its operations on the DR. If M_R queue for a DR is empty (indicates that no other site has requested for the migration of the DR), then site i enters into the holding state for that DR.
(ii) Access_Requests (A_R) queue: this queue stores the site_id of the sites that have generated access requests for the DR.

From now onwards, for simplicity, the migratory request will be represented as $M\_Req_i[x]$ and access request will be represented as $A\_Req_i[x]$, which respectively implies a migratory request or an access request generated by site i for DR x.

In the proposed algorithm, whenever site i needs to access DR x, it checks whether it has DR x with itself, if it has, it accesses DR x. Otherwise, it invokes either $M\_Req_i[x]$ or $A\_Req_i[x]$ and enters into the requesting state for DR x. Site i then forwards its request to the site pointed by $father_i[x]$ (father of site i for DR x ($=j$, say)).

The case when a site receives $M\_Req_i[x]$ is discussed first and then the events that takes place whenever a site receives $A\_Req_i[x]$ are explained.

Whenever site i receives $M\_Req_j[x]$ and it holds DR x, it updates its father pointer for DR x to the requesting site j ($father_i[x]=j$). DR x is then migrated to the requesting site j along with the father pointer value of site i for DR x. Otherwise, site i forwards the request to the site given by $father_i[x]$, in the following situations:

(i) site i is in non-requesting state for DR x, or
(ii) site i is in requesting state for DR x and its M_R queue for DR x is not empty

Site i after forwarding the request updates its $father_i[x]$ pointer to site j, because according to site i's knowledge site j is going to be the new holder site of DR x in near future. If none of the above conditions is true, site i adds the request to its M_R queue for DR x and sets $father_i[x]$ to site j, as site j will receive DR x shortly. In this way, the request is forwarded from one father site to another and in a finite time, $M\_Req_j[x]$ will be forwarded to the root and at that time site j will become the new root.

If site i receives $A\_Req_j[x]$ and it holds DR x, then site i allows site j to access DR x by fixed processing at site i. The operation results are sent back to the site j. If site i does not hold DR x and is in non-requesting state for DR x, then it forwards the received access request to the site pointed by its $father_i[x]$. If site i is in requesting state for DR x, it does not forward the request further, instead adds the request to its A_R queue for DR x.

Whenever site i receives DR x in response to its migratory request, it first completes its operations on DR x and then caters to all the access requests, which are waiting at site i (given by its A_R queue for DR x). After all the access requests for DR x have been catered for, the operation results along with father pointer value of site i for DR x are send to the access request generator sites. The access request generator sites then update their father pointers for DR x. Site i then checks its M_R $queue_i[x]$. If some migratory request is waiting, DR x is migrated to the waiting site and if M_R $queue_i[x]$ is empty (that is no migratory request is waiting for DR x), then site i keeps DR x with itself and enters into the holding state for DR x. Therefore, the request is sequentially forwarded from father site to father site but DR x is directly migrated from the DR x holding site to the next requesting site.

In Section 4, the proposed algorithm is illustrated with the help of an example.

4. An example

Fig. 1 shows the initial state of an example, where there are eight sites (0,1,. . .,7) and three DRs (DR 0, DR 1 and DR 2). DR 0, DR 1 and DR 2 are assumed to initially reside on sites 0, 1 and 2, respectively. All the sites set their father pointers for DR 0, DR 1 and DR 2 to sites 0, 1 and 2, respectively. The arrows represent the direction of father pointers. M_R and A_R queues of all the sites are initialized to nil.

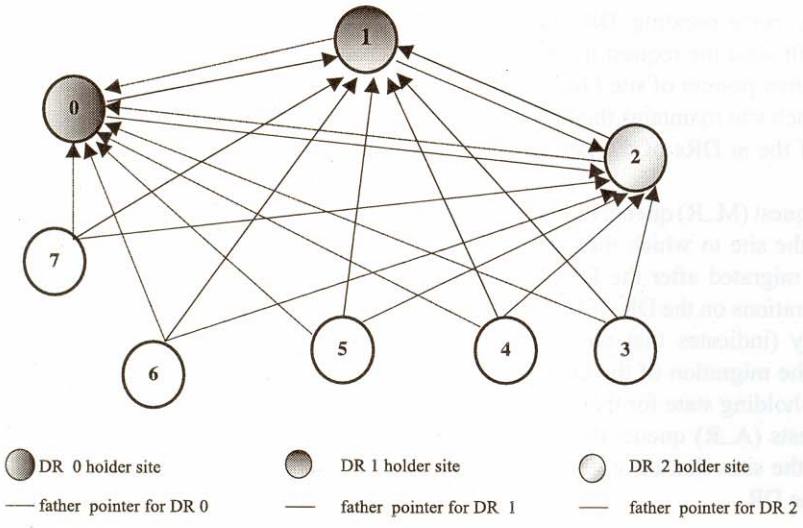

Fig. 1. Initial state of an example.

In this example, it is assumed that sites 3, 4, 5 and 6 generate migratory requests for DR 1 and sites 7 and 2 generate access requests for DR 1 in this order. Fig. 2(a–h) illustrates how these requests chase DR 1. The shaded circle shows the presence of DR 1. To maintain simplicity in the diagrams, the father pointers pointing towards DR 0 and DR 2 are omitted.

Whenever site 3 generates $M\_Req_3[1]$, it enters into the requesting state for DR 1. $M\_Req_3[1]$ is transmitted to site 1. Let us assume, at this moment site 1 is performing operations on DR 1, so it enters the incoming request to its M_R queue for DR 1 and updates its $father_1[1]$ to the requesting site, i.e. site 3, as shown in Fig. 2a. Site 4 now transmits $M\_Req_4[1]$ and sets itself to requesting for DR 1. This request is forwarded to the site pointed by $father_4[1]$, that is site 1. As site 1 is performing operations on DR 1 and its M_R queue for DR 1 is not empty, therefore, it forwards $M\_Req_4[1]$ to the site pointed by $father_1[1]$, i.e. site 3, where it is entered into its M_R queue (since site 3 is in requesting state and its M_R queue is empty). After forwarding $M\_Req_4[1]$ to site 3, site 1 modifies its $father_1[1]$ to the requesting site, site 4. This is done because site 4 will be receiving DR 1 in near future. Also site 3 on receiving $M\_Req_4[1]$ from site 1, updates its $father_3[1]$ to site 4. This is shown in Fig. 2b. At this stage, site 5 generates $M\_Req_5[1]$, enters into requesting state for DR 1 and forwards this request to $father_1[1]$, that is site 1. At this moment, site 1 is performing operations on DR 1 and its M_R queue for DR 1 is not empty, hence site 1 forwards this request to the site pointed by $father_1[1]$, i.e. site 4. Site 4 adds this request to its M_R queue for DR 1, as shown in Fig. 2c. Also, sites 3 and 4 update their father pointers for DR 1 to point to the requesting site, i.e. site 5. Likewise, when $M\_Req_6[1]$ arrives at site 1, it forwards $M\_Req_6[1]$ to the site pointed by its $father_1[1]$, that is site 5. Site 5 is in requesting state and as its M_R queue for DR 1 is empty, it adds $M\_Req_6[1]$ to its M_R queue for DR 1. Site 1 and site 5 also modify their father pointers for DR 1 to the requesting site 6. This is illustrated in Fig. 2d. In the mean time, sites 7 and 2 generate $A\_Req_7[1]$ and $A\_Req_2[1]$, respectively. These requests are also forwarded to site 1, where they are pushed to A_R queue for DR 1. This is shown in Fig. 2e. Site 1 after finishing its operation on DR 1 finds that its A_R queue for DR 1 is not empty. It therefore allows sites 7 and 2 to access DR 1. After sites 7 and 2 finish their operations on DR 1, site 1 sends an update message along with the information of its father pointer (corresponding to DR 1) to them. Sites 7 and 2 then update their father pointers, which now point to site 6. This is illustrated in Fig. 2f. Site 1 now checks for the waiting migratory requests in its M_R queue for DR

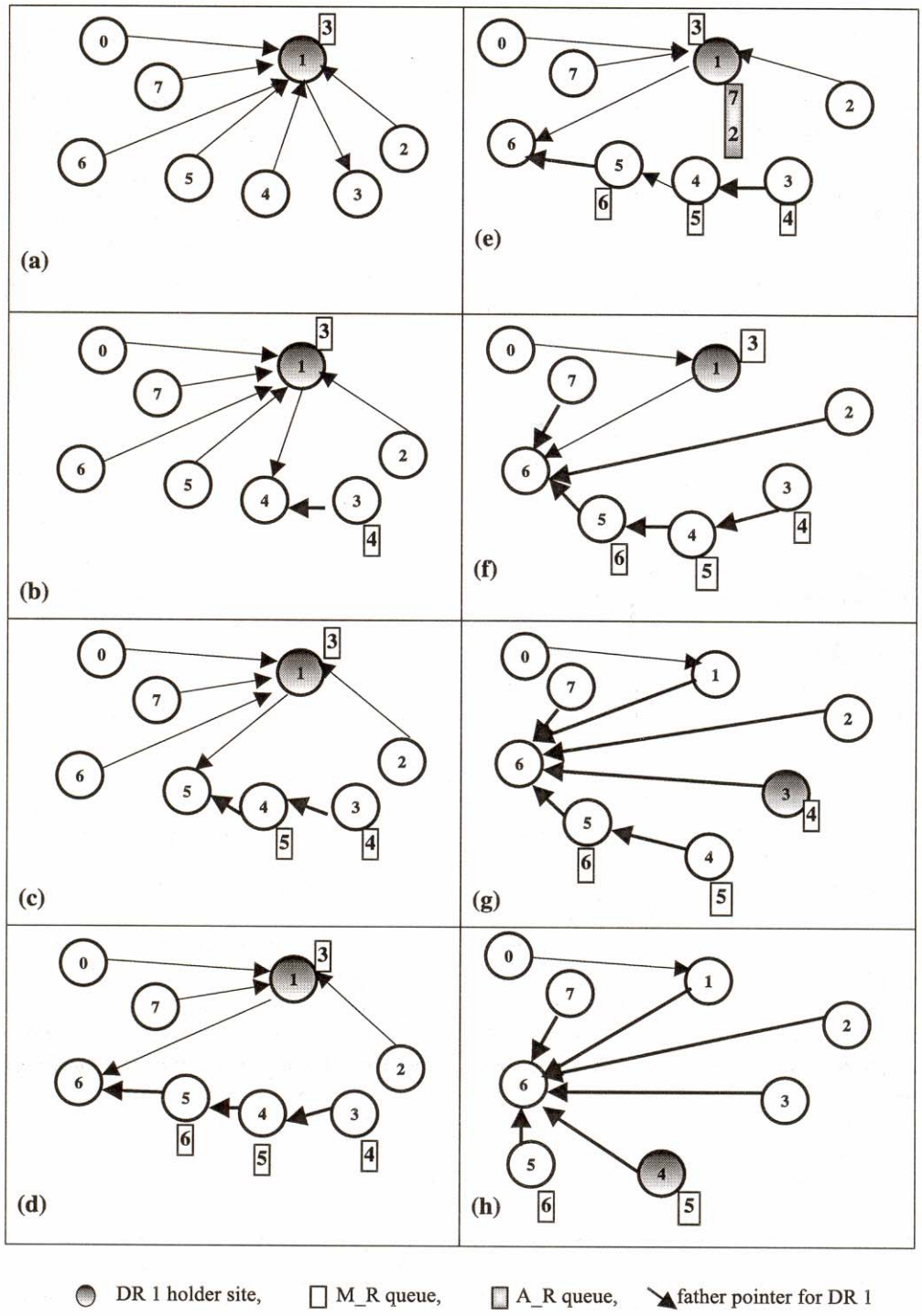

Fig. 2. An example.

1, which is site 3. So DR 1 along with the value of its father pointer for DR 1 is migrated to site 3 and the entry in the M_R queue for DR 1 is reset to nil. Site 3 on receiving DR 1 modifies its father pointer for DR 1 to site 6 as shown in Fig. 2g. Likewise, site 3, after finishing its operation on DR 1, migrates DR 1 (along with the value of its father pointer of DR 1) to site 4, in order to cater $M\_Req_4[1]$. Site 4 after receiving DR 1 updates its father pointer for DR 1 to site 6. This is shown in Fig. 2h. After all the above requests have been catered to, DR 1 will be finally migrated to site 6, which now becomes the new root. All the sites now have a new set of father pointers for DR 1 directed towards site 6.

5. The algorithm in detail

5.1. Data structures

For each site i, the following data structures are constructed to record the necessary information:

(i) *SI table*: This table has m entries (corresponding to m DRs) to record the state of the site with respect to each of the m DRs.
(ii) *father table*: This table has m entries to store the value of father pointers for all the m DRs.
(iii) *req_type (request type) table*: This table has m entries to record the type of request generated by the site for a DR. This parameter is valid only if the site is requesting for a DR. It is set to A for the access and M for the migratory request.
(iv) *M_R queue*: This is a distributed queue for each DR in which each site can store one value to record the site where the corresponding DR would be migrated next. In case there is no waiting migratory request for the DR, the queue is reset to nil.
(v) *A_R queue*: This queue is maintained for each DR. It records the sites that have requested site i for fixed processing on the DR.

Two operations are defined on this queue:

(a) push_A_R $queue_i[x]$(site_id): pushes the site_id of the requesting site in the A_R queue for DR x at site i.
(b) pop_ A_R $queue_i[x]$(): pops A_R queue for DR x at site i and returns the site_id of the site, which had generated an access request for DR x.

The following messages are assumed to exist and are used to exchange information among the sites in the system:

(i) *Request message:* This message is created and sent by any site i which intends to either migrate DR x or access it by fixed processing. The format of the request message is as follows: Reqst_msg(i,x,$req\_type_i[x]$)

The parameters have the following meaning.

- i: site_id of the requesting site.
- x: x is the DR which the site wants to either migrate or access by fixed processing.
- $req\_type_i[x]$: this is the information regarding the type of request (A or M).

(ii) *DR_migrate message*: This is DR x along with the father pointer information of the site from which DR x migrates. The format of the DR_migrate message is given below: DR_migrate($father_i[x]$) where $father_i[x]$ represents the value of the father pointer for DR x of the site that migrates DR x.
(iii) *Update message*: This message is invoked by DR x holding site after all the waiting access requests (in its A_R queue for DR x) finish their operations on DR x. This message informs all the sites about the completion of their access requests for DR x and also updates their father pointers for DR x. The format of the Update message is as follows: Update_msg($father_i[x]$) where $father_i[x]$ represents the value of the father pointer for DR x of the site that generates the Update message.

5.2. Initialization

In this section, the initialization process of the above mentioned data structures is explained. Initially, it is assumed that DR 0 resides at site 0, DR 1 at site 1, DR x at site x and so on. The data structures at all the sites are initialized as follows:

```
for(i=0;i<n;i++){
      for(j=0;j<m;j++){
            if(j=i){
                  father_i[j]=i;/*initially m DRs reside on m sites,
                  one on each site*/
                  SI_i[j]=H; /*m sites (having one DR each) are in
                  holding state*/
            }
            else{
                  father_i[j]=j; /*at each site, father pointers are
                  set for all DRs*/
                  SI_i[j]=NR; /*all sites for all DRs are set to
                  non-requesting state*/
            }
                  M_R queue_i[j]=nil; /*M_R queues at all sites for
                  all DRs are reset to nil*/
                  A_R queue_i[j]=nil; /*A_R queues at all sites for
                  all DRs are initialized to nil*/
                  req_type_i[j]=M; /*request type at all sites for
                  all DRs is initialized to migratory*/
      }
}
```

5.3. Pseudo-code of the algorithm

Now, the proposed algorithm is explained step by step, where each site i in the system is driven by the following events.

5.3.1. Site i generates a request for either migrating or accessing DR x by fixed processing

Site i either generates a migratory or an access request for DR x and sets its req_type for DR x to M or A, respectively. It then checks if it has DR x, if

```
Generate_Reqst_msg(i,req_type_i[x])
{

      if(migratory request for DR x)
            req_type_i[x]=M; /*if migration, req_type is set to M*/
      else
            req_type_i[x]=A;/*if fixed processing, req_type is set
            to A*/

      if(father_i[x]==i)/*checks if the site has DR x with itself*/
            DR x_access;/*access DR x*/
      else{  /*site i does not have DR x*/
            SI_i[x]=R; /*site i becomes requesting for DR x*/

            send Reqst_msg(i,req_type_i[x]) to
            site_id[father_i[x]];
            /*forwards the request to the next site*/
            if(req_type_i[x]==M) /*if migratory request, site i
            updates its father pointer for DR x to itself*/
            father_i[x]=i;
```

so, it finishes its operations on DR x. Otherwise, site i sets itself requesting for DR x and forwards its request along with its site_id to the site given by father$_i$[x]. After forwarding the request, it updates its father pointer for DR x to itself. The detailed description of the procedure is as follows.

/*This procedure is executed by site i when it wants to either migrate or access DR x*/

5.3.2. Site i receives a request message reqst_msg(i, req_type$_j$[x]) from site j for DR x

Whenever site i receives a request message, it takes following actions depending upon its current state and the type of request received:

(i) if it is in non-requesting state and does not hold DR x: it forwards the request message to the site pointed by father$_i$[x]. It then updates its father$_i$[x] to point towards the requesting site only in case of a migratory request. This is done because the requesting site will receive DR x soon (as it has generated a migratory request) and hence all further requests should be forwarded to this site.

(ii) if it is in requesting state: then the migratory request is forwarded further if its M_R queue for DR x is not empty, otherwise it is added to its M_R queue for DR x. If it is an access request, then the request is added to its A_R queue for DR x.

(iii) if it is holding DR x: DR x is migrated to the requesting site in case of a migratory request. While DR x migrates, it carries with itself the father pointer information of site i for DR x, so that the site receiving DR x can modify its father pointer for DR x. This father pointer information carried by DR x reduces the message traffic in subsequent requests for DR x. Similarly, if site i receives an access request for DR x, it allows the requesting site to access DR x by fixed processing at site i. After the operation finishes, site i sends an Update message to the requesting site that carries results as well as the father pointer information of site i for DR x, so that requesting site can update its father pointer for DR x.

/*This procedure is executed by site i when it receives a Reqst_msg(i,req_type$_j$[x]) from site j*/

```
Recv_Reqst_msg(j,req_type_j[x])

{
      if(father_i[x]!=i)&&SI_i[x]==NR){    /*site i does not hold DR
      x and is in non-requesting state*/
            send Reqst_msg(j,req_type_j[x])to site_id[father_i[x]];
            /*sends the received request to the site given by its
            father pointer for DR x*/
            if(Reqst_msg.req_type_j[x]==M) /*if the receive request
            is migratory, update the father pointer to the
            requesting site*/
            father_i[x]=Reqst_msg.j;
      }

      else{
            if(SI_i[x]==R){   /*if site i is in requesting state*/
                  if(Reqst_msg.req_type_j[x]==A) /*if the received
                  request is an access request, then enter the
                  request in A_R queue for DR x*/
                        Push A R queue_i[x](Reqst_msg.j);

            else{  /*received request is migratory*/
                  if(M_R queue_i[x]==nil){ /*if M_R queue is empty,
                  add the request to the M_R queue*/
                  M_R queue_i[x]=Reqst_msg.j;
                  father_i[x]=Reqst_msg.j;/*update the father
                  pointer for DR x to the requesting site*/
            }
```

```
            else{/*if M_R queue for DR x is not empty, forward the
            request further*/
                  send Reqst_msg(j,req_type_j[x]) to
                  site_id[father_i[x]];
                  father_i[x]=Reqst_msg.j;/*update the father
                  pointer for DR x to the requesting site*/
            }
            }
      }

      else{ /*site i holds DR x*/
            if(Reqst_msg.req_type_j[x]==A){ /*the received request
            is an access request*/
                  DR x_access;  /*access DR x by fixed processing*/
                  send Update_msg(father_i[x]) to
                  site_id[Reqst_msg.j]; /*send an Update message
                  to the requesting site*/

            }
            else{/*in case of a migratory request*/
                  send DR_migrate(father_i[x]) to
                  site_id[Reqst_msg.j];
            }
      }
}
```

5.3.3. Site i receives a message DR_migrate(father$_j$[x]) from site j

Whenever site i receives this message, the following two cases can arise:

(i) DR_migrate.father$_j$[x] ≠ site i: it indicates that the site given by DR_migrate.father$_j$[x] is the site that has generated the request after site i in the distributed queue. Therefore, the site represented by DR_migrate.father$_j$[x] will receive DR x after site i finishes its operation on DR x. Thus, site i updates its father pointer for DR x to the site given by DR_migrate.father$_j$[x].

(ii) DR_migrate.father$_j$[x] = site i: it implies that there is no other site requesting for DR x after site i, or the last requesting site has been updated to another requesting site k, when site i received a migratory request from site k. In the latter case, site k will get DR x in a finite time after site i finishes its operation on DR x. Therefore, site i does not update its father pointer for DR x.

After receiving DR x site i first finishes its operations on DR x, it then checks its A_R queue for DR x and if the queue is not empty, it allows the requests to access DR x at site i. After the operations on DR x are over, site i sends an update message along with the results and its father pointer value for DR x to all those sites whose access requests it has catered to. It then looks into its M_R queue for DR x, if there is an entry, it migrates DR x to the site given by M_R queue$_i$[x].

```
Recv_DR_migrate(father_j[x])
{
      if(DR_migrate.father_j[x]!==i)
            father_i[x]=DR_migrate.father_j[x];
            DR x_access; /*site i accesses DR x*/
            SI_i[x]=H;   /*site i enters holding state for DR x*/
      if(A_R queue_i[x]!==nil){/*if A_R queue for DR x at site i is
      not empty, cater to access requests*/
```

```
            for(i=0;i<n;i++){
                  s=pop A_R queue_i[x]();/*pop the site_id that has
                  generated an access request for DR x*/
                  DR x_access; /*site s accesses DR x at site i by
                  fixed processing*/
                  send Update_msg(father_i[x]) to site_id[s];
                  /*After site s finishes its operation on DR x,
                  send Update message to site s*/
            }
      }
      if(M_R queue_i[x]!==nil){/*if M_R queue for DR x at site i is
      not empty, migrate DR x to the next requesting site*/
            send DR_migrate(father_i[x]) to site_id[M_R queue_i[x]];
            /*migrate  DR x*/
            SI_i[x]=NR;/*site i enters non-requesting state for DR
            x*/
            M_R queue_i[x]=nil;/*reset the entry in M_R queue for
            DR x  at site i to nil*/
   }
}
```

After sending the DR, it resets the M_R queue$_i$[x] to nil and enters into non-requesting state. If M_R queue$_i$[x] does not have any entry, it keeps DR x with itself and enters into holding state for DR x.

/*This procedure is executed by site i when it receives a DR_migrate (father$_j$[x]) from site j*/

5.3.4. Site i receives an update_msg(father$_j$[x]) from site j

When site i finishes its operations on DR x by fixed processing, it receives an Update message. It then changes its state from requesting to non-requesting and updates its father pointer for DR x.

```
Recv_Update_msg(father_j[x])
{
      SI_i[x]=NR; /*site i enters non-requesting state for DR x*/
      if(Update_msg.father_j[x]!==i)
      father_i[x]=Update_msg.father_j[x];/*father pointer of site i
      for DR x updated with the value of father pointer brought
      by the Update message*/
}
```

/*This procedure is executed by site i when it receives an Update_msg(father$_j$[x]) from site j*/

6. Analysis of the algorithm

In this section, the proposed algorithm is analysed and it is proved that it is free from deadlock and starvation.

6.1. Deadlock

A distributed system is said to be in deadlock when none of the sites is performing operation on the DRs and no requesting site can ever perform operation on them. To show that such a situation never occurs and a requesting site eventually gets access to the DR, the two conditions must hold:

(i) the request message (whether access or migratory) from a requesting site i will arrive at site j satisfying father$_j$[x]=j (i.e., it either holds DR x or is going to receive it in near future), and

(ii) site j migrates DR x or sends an Update message to site i in a finite time. We use the following lemmas to prove that the above two conditions hold true for the proposed algorithm.

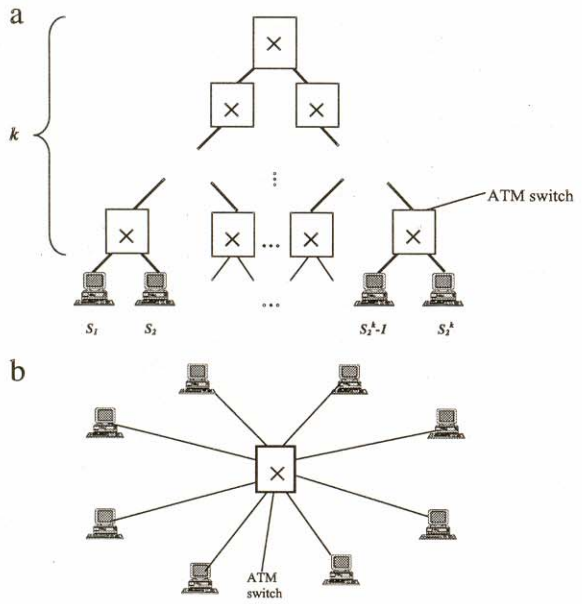

Fig. 3. (a) Binary Tree Topology and (b) Star Topology.

Lemma 1. *Any path constructed by father pointers for DR x always leads to a site that satisfies* $father_i[x] = i$.

Proof. It is assumed that initially every site sets its father pointer for DR x to the only site r that satisfies $father_r[x] = r$. Whenever a site generates a migratory request or receives a migratory request or a DR, its father pointer for the DR is updated. Whenever a requesting site i sends $M\_Req_i[x]$ to $father_i[x] = r$, it sets $father_i[x] = i$. After site r receives $M\_Req_i[x]$, it sets $father_r[x] = i$; therefore, there is a path from site r to site i that satisfies $father_i[x] = i$. At this instant, any other site, say, j can send $M\_Req_j[x]$ along the path made by $father_j[x] = r$ and $father_r[x] = i$. Assume that site r then receives $M\_Req_{j1}[x]$, $M\_Req_{j2}[x]$, ..., $M\_Req_{jk}[x]$ (the migratory requests from the sites j_1, j_2, ..., j_k) for DR x before it migrates DR x. During this period, site r will keep on forwarding the $M\_Req_{j1}[x]$ to $father_r[x] = i$ (then sets $father_r[x] = j_1$), ..., forwarding $M\_Req_{j(i+1)}[x]$ to $father_r[x] = j_i$ (then sets $father_r[x] = j_{(i+1)}$)... Finally, $father_i[x] = j_1$, $father_r[x] = j_k$ and $father_{j1}[x] = j_{(i+1)}$ $(1 = i = (k-1))$. At this moment, any path constructed by father pointers leads for DR x to j_k that satisfies $father_{jk}[x] = j_k$.

When site r (at this moment $father_r[x] = j_k$) finishes with the operation, it migrates DR x to $M\_R\ queue_r[x] = i$. After receiving DR x, site i updates its $father_i[x]$ pointer to the latest requesting site, i.e. j_k and so does every site j_i to site $j_{(i+1)}$ $(1 = i = (k-1))$. Therefore, any path constructed by father pointers, always leads to a site satisfying $father_i[x] = i$. Also, there is only one site that satisfies $father_i[x] = i$, when there is no request in the system. Similarly, whenever a requesting site i sends $A\_req_i[x]$ to $father_i[x] = r$, where it is entered into its A_R queue for DR x. When site r receives DR x (let us assume at this time $father_r[x] = j_k$), it allows access to site i and sends an Update message to site i. Site i then updates its father pointer for DR x to the last requesting site j_k. Now, if site i again sends another access request, it will now be transmitted to the site j_k that satisfies $father_{jk}[x] = j_k$. □

Lemma 2. *When there is no migratory or access request in the system, the entries in the M_R or A_R queue contain every requesting site.*

During initialization, every site resets its M_R queue for DR x to nil. A migratory request $M\_Req_i[x]$ will arrive at site j that satisfies $father_j[x] = j$. Since site j has not received any request that can be inferred from $father_j[x] = j$, it implies $M\_R\ queue_j[x] = nil$. In other words, site j is the last site in the distributed queue. Then site j sets $M\_R\ queue_j[x] = i$. Therefore, the queue constructed by

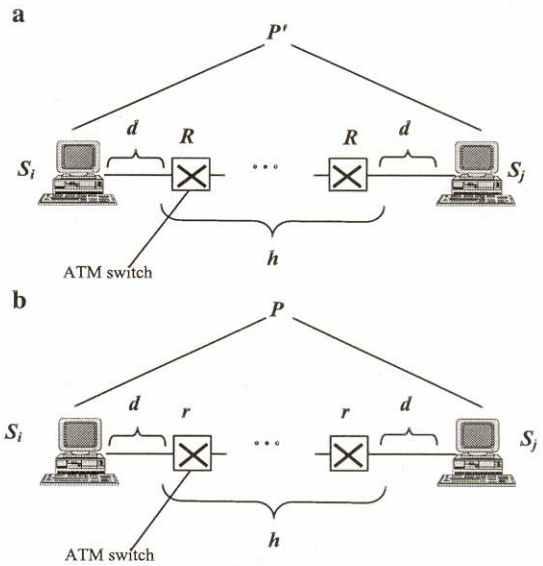

Fig. 4. (a) Channel Formation and (b) Message Propagation.

M_R contains the migratory request of every requesting site. Similarly, an access request A_req$_i[x]$ will arrive at site j that satisfies father$_j[x]=j$, where it gets entered to A_R queue irrespective of whether it has an entry or not. Hence, all the A_R queues contain access requests of every site.

From Lemma 1, a migratory or an access request will be forwarded along the path formed by father pointers to a site that satisfies father$_i[x]=i$; therefore, condition 1 is satisfied. From Lemma 2, the queue formed by M_R contains migratory requests of every requesting site and is finite, and the site holding the DR x will migrate DR x to its M_R queue[x] (if M_R queue[x] $\neq$ nil). Also the A_R queues contain access requests of every requesting site and is finite and the site holding DR x will send an Update message to the site given by A_R queue for DR x (if A_R queue[x] $\neq$ nil.). Hence, condition 2 is satisfied and therefore the algorithm is free from deadlock.

6.2. Starvation

Starvation is a condition when the requests of only few sites are catered to, while other sites wait indefinitely for their turns to do so. Since every migratory or access request will arrive at a site that is either holding or is going to receive DR x in a finite time, the only cause of starvation is the unfair decision made by the DR holding site. That is if M_Req$_i[x]$ has arrived at a site that satisfies

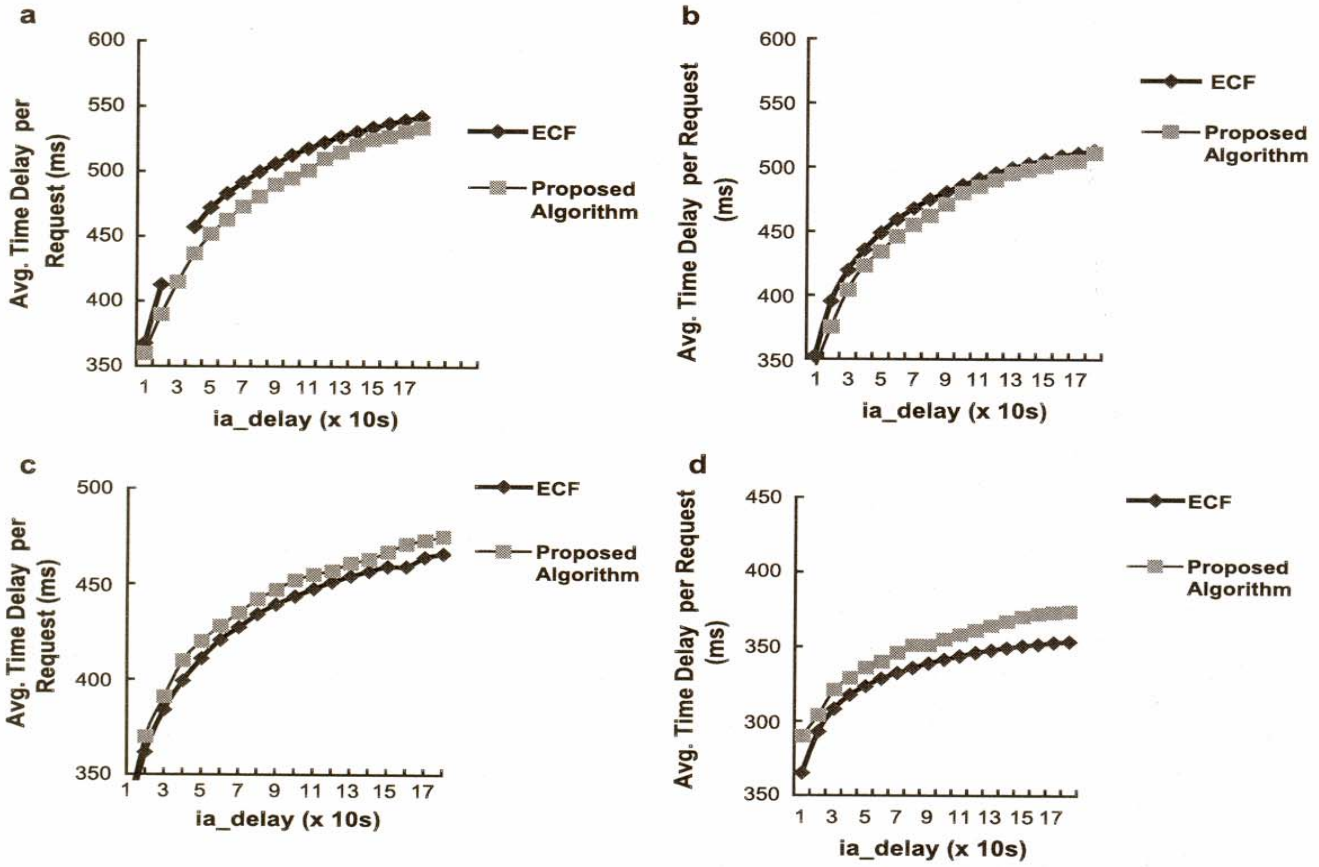

Fig. 5. (a) Time Delay for A/M ratio = 1 for Binary Tree Topology; (b) Time Delay for A/M ratio = 5 for Binary Tree Topology; (c) Time Delay for A/M ratio = 10 for Binary Tree Topology; (d) Time Delay for A/M ratio = 15 for Binary Tree Topology; (e) Time Delay for A/M ratio = 1 for Star Topology; (f) Time Delay for A/M ratio = 5 for Star Topology; (g) Time Delay for A/M ratio = 10 for Star Topology; (h) Time Delay for A/M ratio = 15 for Star Topology.

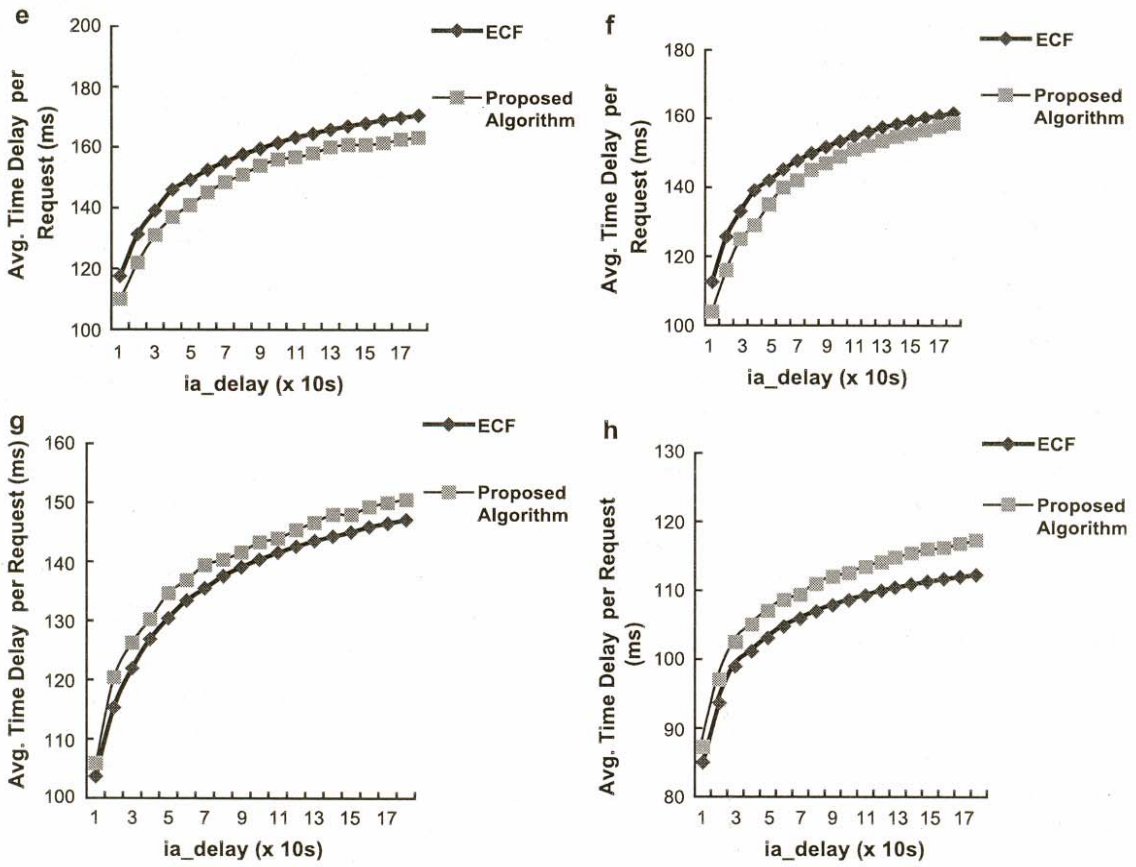

Fig. 5 (*continued*).

$father_j[x]=j$, and a finite number of other sites generate migratory requests for DR x after site i and obtain DR x before site i, starvation may occur. Or if some access requests are waiting in the A_R queue for DR x and the site migrates DR x to another site without satisfying them.

In the proposed algorithm, when $M\_Req_i[x]$ message arrives at site j that satisfies $father_j[x]=j$, site j will set $father_j[x]=i$ and $M\_R\ queue_j[x]$ to i. When some other migratory request $M\_Req_k[x]$, say from site k arrives at site j, site j will forward $M\_Req_k[x]$ to $father_j[x]=i$. Therefore, any requesting site (for migration) after site i will be added after site i in the distributed queue and DR x is migrated in the same order as sites in the distributed queue, site i will receive DR x in a finite time and before any requesting site (for migration) which is after it in the distributed queue. Similarly, when $A\_req_i[x]$ message arrives at site j that satisfies $father_j[x]=j$, site j will either allow site i to perform operations on DR x (if it is in holding state for DR x) or will enter the request in A_R queue for DR x. Now if some other access request $A\_req_k[x]$, say from site k arrives at site j, site j will enter it in A_R queue for DR x after the request of site i. Whenever site j receives DR x, A_R queue for DR x is first checked and the A_R queue is then read in FIFO manner. Hence, the access request arriving first is catered to first. Moreover, site j does not read its M_R queue for DR x, till all the access requests for DR x have been catered to. Therefore, DR x will not migrate to the next requesting site till all the previous access requests have been satisfied. Therefore, the algorithm is free from starvation.

7. Simulation model and results

The simulation model used in this paper is similar to that used in Hara et al. [7]. The algorithms for comparison (the proposed algorithm and ECF) are implemented on two network topologies: (i) a binary tree with depth k (number of sites $n=2^k$) having sites as leaves and ATM switches at every other level (Fig. 3a) and (ii) a star topology with a central ATM switch (Fig. 3b). The sites communicate by setting up SVCs. Whenever a site i sends a message to site j, the message is routed through intermediate ATM switches. The total time required for transmitting one message between two sites consists of two parameters: (a) the time required for setting up the SVC connection called Channel Formation (CF) delay (Fig. 4a) and (b) the time needed for the transmission of the message once the connection is made called Message Propagation (MP) delay (Fig. 4b). Let h represent the number of ATM switches between the two communicating sites, P' the total processing delay at both sites when SVC does not already exist, d the constant propagation delay between two arbitrary sites, R and r the constant route configuration time and the constant routing time respectively at an arbitrary switch, then $CF(h)=P'+2(h+1)d+h(r+R)$. If during transmission, P is the total processing delay at both sites when SVC already exist, then $MP(h)=P+(h+1)d+hr$. The parameter values of P, P', r, R, d are taken to be 30, 10, 2, 10, 5 ms, respectively, and are same as those given in Ref. [7]. The number of sites and DRs are assumed to be 32 and 20 and each DR say DR x is initially located at site x. For simplicity, each site generates requests in equal probability and the requested DR is also chosen among 20 DRs in equal probability. The intervals of accesses and DR migrations are based on exponential distributions and a SVC between two sites remains valid for 20 min after the setup.

As the performance measures are probabilistic in nature, so value of these variables for the proposed algorithm and ECF are collected for 5000 requests, in which the intervals between requests follow exponential distributions which are based on the mean access interval called inter access delay (ia_delay). The value of ia_delay is changed from 10 s to 3 min in steps of 10 s. The time delay of the algorithms has also been studied against the ratio of access to migratory requests generated in the system called A/M (Access to Migratory) ratio, which is varied from 1 to 15. The simulations were carried out in PARallel Simulation Environment for Complex systems (PARSEC), which is a C-based discrete event simulation language [18]. It adopts the process interaction approach to discrete events simulation. An object (or physical process) or set of objects in the physical system is represented by a logical process. Interaction among physical process (events) is modeled by time stamped message exchanges among the corresponding logical processes.

In the simulation experiments, two performance parameters viz., time delay and message traffic are measured for two network topologies. Time delay is the average time needed by the requesting site to locate the DR, which is the period of time between the instant a site generates a request and the instant when the site accesses the DR. The time delay is computed from the time the request message is sent until the time the DR is migrated to the requesting site in case of a migratory request or the Update message is received by the requesting site in case of an access request. Message traffic is the average number of messages needed by the requesting site to locate and perform operations on DR.

Fig. 5 shows the simulation results. In Fig. 5(a–h), the x-axis indicates the value of ia_delay and y-axis gives time delay for the proposed algorithm and ECF. The graphs are plotted for four different values of A/M ratio and for two network topologies namely binary tree and star. The method that gives the shortest time delay is also examined. It is observed that in both the methods (proposed algorithm and ECF), time delay increases as the mean access interval is increased from 10 s to 3 min for a fixed A/M ratio. Also, in both the methods, time delay increases with the increase in A/M ratio (from 1 to 15). Time delay for ECF and the proposed algorithm changes nearly from 0.265 to 0.542 and 0.290 to 0.534 s, respectively, for binary tree topology. For star topology, time delay varies approximately from 0.085 to 0.170 and 0.087 to 0.163 s for ECF and the proposed algorithm, respectively. The time delay for ECF and the proposed algorithm is comparable in star network at all the frequencies of DR migration, whereas the performance of the proposed algorithm

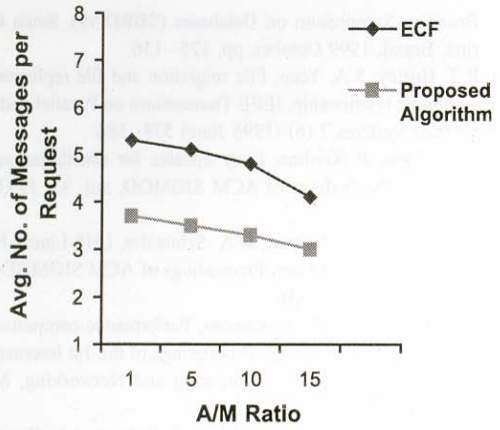

Fig. 6. Variation in Message Traffic with respect to A/M Ratio for ia_delay=90s.

improves for binary tree topology as compared to ECF as A/M ratio is decreased. Also, the proposed algorithm performs better than ECF when DR migration occurs frequently. This is because the proposed algorithm tries to direct the request to a requesting site, so that endless chasing of the DR is minimized. Moreover, there is an active contention for DR in ECF whereas in the proposed algorithm, there is a passive contention, that is, the site that is already requesting is allowed to access the DR first. Hence, in the proposed algorithm, fairness is ensured such that earlier requesting site gets to access the DR first.

Message traffic for the proposed algorithm as well as for ECF is also studied. It is observed that the message traffic is independent of the mean access interval but it varies with respect to A/M ratio. In Fig. 6, x-axis indicates the A/M ratio and y-axis gives message traffic for ECF and the proposed algorithm for both the network topologies. From the graph, it is observed that the proposed algorithm reduces the message traffic to a considerable extent. This is because in the proposed algorithm, the requests are stopped whenever they encounter a requesting site, thereby reducing the unnecessary forwarding of request messages. On the other hand, the message traffic in ECF is higher than that of the proposed algorithm, because in ECF, the request is not stopped anywhere, instead the intermediate sites keep on forwarding the request until the DR holding site is found. After the site holding the DR is found, the requesting and all the intermediate sites are updated with the latest DR location table contents. Further, both these methods give lesser message traffic when DR migration does not occur frequently. This is because there is a high probability that the site to which the requesting site sends the message is the DR holding site.

8. Conclusion

In this paper, an algorithm that uses a distributed queue strategy to simplify the data structure carried by the DR and the Update message is proposed. The amount of information carried by DR and the Update message is small as they carry only the father pointer value (corresponding to the requested DR) of the site that sends them. To speed up the search for the DR, the algorithm tries to keep up-to-date as much as possible the value of father pointers whenever a site receives any message. Moreover, the size of the M_R queue for each DR at each site is always one. It ensures fairness in the system that is while chasing a DR, if the request reaches a requesting site, it is stopped there instead of being forwarded. Hence, the requests generated earlier are catered to first.

Further, by simulation experiments, the performance of the proposed algorithm is compared with ECF. The simulation results show that in the proposed algorithm, the message traffic is reduced considerably as compared to ECF. The proposed algorithm reduces the message traffic by keeping track of the latest information about the site possibly holding the DR. Also, the time delay of the proposed algorithm is less as compared to ECF at lower A/M ratios, especially in star topology. Finally, it is concluded that the proposed algorithm performs best with networks of higher degrees of connectivity and when DR migration occurs frequently.

Acknowledgements

The author thanks the anonymous referees for their suggestions and comments on an earlier version of the paper. Their suggestions have greatly enhanced the quality of the paper.

References

[1] T. Akiyama, T. Hara, K. Harumoto, M. Tsukamoto, S. Nishio, Access skew detection for dynamic database relocation, Informatica: International Journal of Computing and Informatics 24 (1) (2000).

[2] S. Banerjee, V.O.K. Li, C. Wang, Distributed database systems in high-speed wide-area networks, IEEE Journal on Selected Areas in Communications 11 (4) (1993 May) 617–630.

[3] Y.I. Chang, M. Singhal, M.T. Liu, An improved O(logN) mutual exclusion algorithm for distributed systems, Proceedings of the International Conference on Parallel Processing, 1990, pp. III-295–III-302.

[4] B. Gavish, O.R.L. Sheng, Dynamic file migration in distributed computer systems, Communications of the ACM 33 (2) (1990 February) 177–189.

[5] A. Hac, A distributed algorithm for performance improvement through file replication, file migration, and process migration, IEEE Transactions on Software Engineering 15 (11) (1989 November) 1459–1470.

[6] G. Herman, G. Gopal, K. Lee, A. Weinrib, The datacycle architecture for very high throughput database systems, Proceedings ACM SIGMOD, 1987, pp. 97–103.

[7] T. Hara, K. Harumoto, M. Tsukamoto, S. Nishio, 'Location Management Methods of Migratory Data Resources in ATM Networks', Transactions of the Institute of Electronics, Information and Communication Engineers, D-I, vol. J80-D-I, no. 2, pp. 137–145, Feb. 1997, also in Proceedings of the ACM Symposium on Applied Computing (ACM SAC'97), pp. 123–130, Feb. 1997, and in Systems and Computers in Japan, vol. 28, no. 9, pp. 35–45, Aug. 1997.

[8] T. Hara, K. Harumoto, M. Tsukamoto, S. Nishio, DB-MAN: a distributed database system based on database migration in ATM networks, Proceedings of the 14th International Conference on Data Engineering (ICDE'98), Florida, 1998 February, pp. 522–531.

[9] T. Hara, K. Harumoto, M. Tsukamoto, S. Nishio, Database migration: a new architecture for transaction processing in broadband networks, IEEE Transactions on Knowledge and Data Engineering 10 (5) (1998 September/October) 839–854.

[10] T. Hara, K. Harumoto, M. Tsukamoto, S. Nishio, Dynamic replica allocation using database migration in broadband networks, Proceedings of the International Conference on Distributed Computing Systems (ICDCS' 2000), Taipei.

[11] T. Hara, K. Harumoto, M. Tsukamoto, S. Nishio, J. Okui, Main memory database for supporting database migration, Proceedings IEEE Rim Conference on Communications, Computers and Signal Processing (IEEE PACRIM'97), Victoria, Canada, vol. 1, 1997 August, pp. 231–234.

[12] H. Hagino, T. Hara, M. Tsukamoto, S. Nishio, J. Okui, A location management method using network hierarchies, Proceedings IEEE Rim Conference on Communications, Computers and Signal Processing (IEEE PACRIM'97), Victoria, Canada, vol. 1, 1997 August, pp. 243–246.

[13] T. Hara, M. Tsukamoto, S. Nishio, A scheduling method of database migration for WAN environments, Proceedings of Brazilian Symposium on Databases (SBBD'99), Santa Catarina, Brazil, 1999 October, pp. 125–136.

[14] R.T. Hurley, S.A. Yeap, File migration and file replication: a symbiotic relationship, IEEE Transactions on Parallel and Distributed Systems 7 (6) (1996 June) 578–586.

[15] T. Johnson, P. Krishna, Lazy updates for distributed search structure, Proceedings of ACM SIGMOD, vol. 93, 1993, pp. 337–346.

[16] W. Litwin, M.A. Neimat, D.A. Schneider, LH*-Linear Hashing for Distributed Files, Proceedings of ACM SIGMOD, vol. 93, 1993, pp. 327–336.

[17] R. Kadobayashi, M. Tsukamoto, Performance comparison of mobile support strategies, Proceedings of the 1st International Conference on Mobile Computing and Networking, Mobicom'95, 1995, pp. 218–225.

[18] R. Meyer, PARSEC User Manual, Release 1.1, Computer Science Department, UCLA, Los Angeles, CA, 1998 August, 90024.

[19] G. Matsliach, O. Shmueli, An efficient method for distributing search structures, Proceedings of the 1st International Conference on Parallel and Distributed Information Systems (PDIS), 1991.

[20] S. Nishio, M. Tsukamoto, Towards new multimedia information base in broadband networks, IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences 79-D-II (no. 4) (1996 April) 460–467.

[21] P.C. Saxena, S. Gupta, G. Gabrani, A location chasing algorithm for migratory data resources in ATM networks, International Journal of Information and Computing Science 3 (2) (2000 December) 1–28.

[22] C. Severance, S. Pramanik, P. Wolberg, Distributed linear hashing and parallel projection in main memory databases, Proceedings of the 16th International Conference on Very Large Databases, 1990, pp. 674–682.

[23] R. Vingralek, Y. Breitbart, G. Weikum, Distributed file organization with scalable cost/performance, Proceedings of ACM SIGMOD, vol. 94, 1994, pp. 253–264.

[24] C.T. Yu, K.C. Guh, D. Brill, A.L.P. Chen, Partitioning relation for parallel processing strategy in fast local networks, Proceedings of the International Conference on Parallel Processing, 1986.

[25] C.T. Yu, K.C. Guh, D. Brill, A.L.P. Chen, Partition strategy for distributed query processing in fast local networks, IEEE Transactions on Software Engineering 15 (6) (1989 June) 780–793.

G. Gabrani is an Assistant Professor in Department of Computer Engineering at Delhi College of Engineering, University of Delhi, New Delhi, India. She received Bachelors, Masters and Ph.D in Engineering in the years 1984, 1990 and 2003 respectively. She has guided several B.E. projects and around 25 M.E. Dissertations. She has published several research papers in International and National Journals. Her research interests include Distributed Systems, Networks and Digital Systems Design.