1-1-2006

CADRE: A Collaborative Replica Allocation and Deallocation Approach for Mobile-P2P Networks

Anirban Mondal

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

Masaru Kitsuregawa

Recommended Citation

A. Mondal et al., "CADRE: A Collaborative Replica Allocation and Deallocation Approach for Mobile-P2P Networks," *Proceedings of the 2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2006.
The definitive version is available at https://doi.org/10.1109/IDEAS.2006.13

CADRE: A Collaborative replica allocation and deallocation approach for Mobile-P2P networks

Anirban Mondal[1] Sanjay Kumar Madria[2] Masaru Kitsuregawa[1]

[1] Institute of Industrial Science
University of Tokyo, JAPAN
{anirban, kitsure}@tkl.iis.u-tokyo.ac.jp

[2] Department of Computer Science
University of Missouri-Rolla, USA
madrias@umr.edu

Abstract

This paper proposes CADRE (Collaborative Allocation and Deallocation of Replicas with Efficiency), a dynamic replication scheme for improving the typically low data availability in mobile ad-hoc peer-to-peer (M-P2P) networks. The main contributions of CADRE are two-fold. First, it collaboratively performs both replica allocation and deallocation in tandem to facilitate optimal replication and to avoid 'thrashing' conditions. Second, it addresses fair replica allocation across the MHs. CADRE deploys a hybrid super-peer architecture in which some of the MHs act as the 'gateway nodes' (GNs) in a given region. GNs facilitate both search and replication. Our performance study indicates that CADRE indeed improves query response times and data availability in M-P2P networks as compared to some recent existing schemes.

Keywords: Mobile peer-to-peer networks, dynamic replication, fair replication, thrashing prevention

1 Introduction

In a Mobile ad-hoc P2P (M-P2P) network, mobile hosts (MHs) interact with each other in a decentralized P2P fashion without using base stations. The proliferation of mobile computing technology (e.g., laptops, PDAs, mobile phones) coupled with the ever-increasing popularity of the Peer-to-Peer (P2P) paradigm (e.g., Kazaa) strongly motivate M-P2P network applications. Let us consider an M-P2P application involving *inter-vehicular communication*. Suppose groups of tourists in different sightseeing buses are visiting several touristic places albeit at different times. Incidentally, each sightseeing bus almost always has its own *schedule* concerning the places that it should visit at different points of time. In practice, every sight-seeing bus generally has a tour guide for facilitating tourists, and tour guides in different buses are typically aware of each other's schedules (since several tour buses are usually owned by the same organization). Thus, tourists in bus A, which is on its way to a particular museum, could request their tour guide for current snapshots (pictures or video-clips) of different parts of the museum to determine which part of the museum they should visit first. The tour guide in bus A would forward such requests to the tour guides in those buses, which are currently visiting the museum, to obtain information from tourists in those buses, thereby supporting *remote querying*. Tourists in bus A could also query their tour-guide *locally* for data items that are stored at the MHs in bus A.

Now suppose tourist X in bus A requests an image, and obtains a replica of the image (from tourists of other buses). After a while, X deallocates (deletes) this replica, but now another tourist Y in bus A requests the same image, hence the replica has to be allocated again to bus A. Since images are relatively large in size, multiple allocations and deallocations of the same replica at the same bus tax the generally limited bandwidths and energy resources of MHs. This may lead to undesirable *'thrashing conditions'*, where MHs spend more bandwidth and energy on allocating and deallocating replicas than on answering queries. Notably, absolute consistency is not a requirement in such scenarios [11].

Data availability is typically low in M-P2P networks due to frequent network partitioning arising from user movement and/or users switching 'on' or 'off' their mobile devices. (Data availability is less than 20% even in a wired environment.) Hence, several dynamic replication schemes [14, 5, 9] have been proposed for improving data availability in M-P2P networks. However, while these existing schemes perform *collaborative* replica allocation, replica deallocation is done only *locally* by each MH. Suppose a data item d has a high access frequency at MHs M_i and M_j, while being rarely accessed at an MH M_k. Under the existing schemes, M_k would deallocate d. But, since d is actually a 'hot' data item, d may be once again allocated to M_k

after sometime, which may lead to undesirable *'thrashing'* conditions. Hence, both replica *allocation* and *deallocation* should be performed *collaboratively* in tandem with each other to prevent thrashing. Furthermore, existing schemes do not consider *fairness in replication* since they allocate replicas solely based on the read/write access probability of any given data item d without considering the origin of queries for d. Thus, these schemes would regard d as 'hot' even if only a single MH issues a large number of (read) queries for d, thereby possibly creating several replicas of d. This runs contrary to the principle of *fairness* in serving multiple peers' requests, which is an important requirement in all P2P systems.

This paper proposes CADRE (Collaborative Allocation and Deallocation of Replicas with Effi ciency), a dynamic replication scheme for improving data availability in M-P2P networks. CADRE considers a hybrid super-peer architecture, in which some of the MHs act as the 'Gateway Nodes' (GNs). The functionality of a GN is analogous to that of a super-peer. In the inter-vehicular M-P2P application, a tour-guide in a given bus would act as the GN for the MHs in his bus. GNs have high processing capacity, high available bandwidth and high energy. Observe that our architecture does *not* have any single 'root' GN for supervising the working of all the GNs. Neighbouring GNs periodically exchange their regional information with each other. Hence, in case of GN failures, neighbouring GNs could take over the responsibility of the failed GN.

GNs facilitate both replication and search. Intuitively, storing replicas arbitrarily at any MH could adversely impact many MHs due to high communication overheads between MHs, unnecessary delays and querying failures. Thus, replication should be performed carefully based on MH characteristics (e.g., load, energy) as well as network topology, thereby implying that some *regional knowledge* becomes a necessity. As we shall see later, GNs have such regional knowledge due to MHs periodically sending the necessary information to GNs, hence GNs can better manage replication. GNs can also collaborate for replication across different regions. In contrast, for an architecture without any GN, each MH would have to broadcast its status to all other MHs to make each other aware of the regional status, thereby creating an undesirable broadcast storm during replica allocation. Our architecture avoids such broadcast storm due to the presence of GNs.

Recall that our M-P2P applications support both *local* and *remote* querying. To support effi cient querying, each MH *periodically* sends its list of data items and replicas to its corresponding GN. Thus, GN is able to periodically broadcast the list of available items within its region to the MHs, thereby enabling a query issuing MH M to distinguish whether its query is *local* or *global*. Moreover, when an MH enters a region R, it registers with the GN G in R, and G provides the MH with the list of data items currently available in R. (We assume that an MH will have to register with one GN.) For local queries, a broadcast mechanism is used. For a remote query Q, MH M sends Q to its corresponding GN, which forwards Q to its neighbouring GNs. If any of the neighbouring GNs contains the item queried by Q, they ask the MH, which stores that item, to send the item to M. Otherwise, the neighbouring GN will forward it to its neighbouring GN and so on.

The main contributions of CADRE are two-fold:

1. It collaboratively performs both replica allocation and deallocation in tandem to facilitate optimal replication and to avoid 'thrashing' conditions.

2. It addresses fair replica allocation across the MHs.

CADRE also considers replication of images at different levels of granularity to optimize MH memory space. Our performance evaluation demonstrates that CADRE indeed improves data availability in M-P2P networks with signifi - cant reduction in query response times and low communication traffi c during replication as compared to some recent existing schemes. Notably, our previous work in [9] has also addressed replication in M-P2P networks. However, it does not address replica deallocation, fairness in replication and prevention of thrashing.

2 Related Work

In [8], a suite of replication protocols for maintaining data consistency and transactional semantics of centralized systems have been proposed. The protocols in [7] exploit the rich semantics of group communication primitives and the relaxed isolation guarantees provided by most databases. Existing systems in this area include ROAM [12], Clique [13] and Rumor [4], while a scalable P2P framework for distributed data management applications and query routing has been presented in [10]. An update strategy, based on a hybrid push/pull Rumor spreading algorithm, for truly decentralized and self-organizing systems (e.g., pure P2P systems) has been examined in [3], the aim being to provide probabilistic guarantees.

In [5], the E-DCG+ approach for replica allocation in mobile ad-hoc networks (MANETs) has been proposed. By creating groups of MHs that are biconnected components in a network, E-DCG+ shares replicas in larger groups of MHs to provide high stability. In E-DCG+, an RWR (read-write ratio) value in the group of each data item is calculated as a summation of RWR of those data items at each MH in that group. In the order of the RWR values of the group, replicas of data items are allocated until memory space of all MHs in the group becomes full. Each replica is allocated at an MH whose RWR value to the data item is the highest among MHs that have free memory space to create it. However, E-DCG+ [5] does not consider collaborative replica deallocation, fairness in replication and load sharing.

3 Key components of CADRE

This section discusses the key components of CADRE.

User-specified size of the query result image

When a user issues a query for an image, he knows his available memory space status. Hence, he knows the *maximum* amount of memory space, which he can expend for storing the query result image. Thus, when querying, users specify the *maximum size*, designated as $maxsize$, for the query result image and CADRE answers queries while considering this $maxsize$ constraint. This is especially important for M-P2P networks due to the generally limited memory space at individual MHs and the significant differences in available memory space across the MHs. Since image size increases with increasingly finer granularity, a user specifying larger value of $maxsize$ would obtain finer image granularity (i.e., better image quality). Notably, this is in contrast with static P2P systems, where memory space constraints of the query issuing peer are not considered when answering queries due to static peers generally having significant available memory space.

Suppose MH M_I issues a query Q for an image img, and MH M_S serves the query request. M_S could either be the owner of img or it could store a replica of img. Let the size of img at M_S be $size_{img}$. The following cases arise:

1. Query Result $maxsize < size_{img}$
2. Query Result $maxsize \geq size_{img}$

In Case 1 above, img should be *compressed* to satisfy the $maxsize$ query constraint. Such compression should be performed by M_S (and not M_I) due to two reasons. First, it ensures smaller-sized images being transmitted across the network, thereby optimizing bandwidth consumption. Second, a one-time compression of img by M_S is likely to enable M_S to serve multiple user requests, which optimizes energy consumption. Furthermore, performing the image compression at M_I would require every query issuing MH to compress img individually, which would increase individual MH energy consumption significantly. Notably, image compression algorithms [2, 6] can be used in conjunction with CADRE. For Case 2, image decompression is not necessary since users specifying larger $maxsize$ values can be directed to either the original owner of img or any MH that stores a relatively larger-sized replica of img.

Given that different users can specify different $maxsize$ values for their queries on img, M_S needs to determine the size of the replica (of img) that it should store at itself to reduce its image compression-related energy consumption. For this purpose, M_S keeps track of queries issued to itself by maintaining a list $RepSize$ of the form (img_{id}, MH_{id}, $maxsize$), where img_{id} is the unique identifier of the queried image img, MH_{id} is the identifier of the MH that issued the query and $maxsize$ is the maximum query result size specified by MH_{id}. If an MH M_I accesses img multiple times, the value of $maxsize$ for the most *recent* access is used to populate $RepSize$ since recent $maxsize$ value specified by M_I better reflects M_I's current memory space status. Thus, given img, M_S determines img's replica size by providing equal weight to accesses made by each query issuing MH since it considers one entry of $maxsize$ for each of these MHs, thereby ensuring *fairness* across multiple user requests for deciding the replica size.

The owner of an image stores the original image. Given the original image size S_o, we consider n different ranges of granularity for replica size based on the extent of image compression relative to S_o. Results of our preliminary performance study revealed that $n = 4$ is a reasonable value for our application scenarios. Hence, we consider the following four ranges of granularity: *low*, *medium*, *high*, *original*. Let the replica size be S_r. For *low*, $(0.25 \times S_o) \leq S_r < (0.5 \times S_o)$. In case of *medium*, $(0.5 \times S_o) \leq S_r < (0.75 \times S_o)$. Similarly, for *high* $(0.75 \times S_o) \leq S_r < S_o$. Finally, for *original*, $S_r = S_o$. Thus, when different MHs issue queries to MH M_S with different $maxsize$ values, M_S maps each query to any one of these four mutually exclusive ranges and keeps a count of the number of queries for each range. M_S determines the replica size to be in the range that corresponds to the maximum number of queries. Finally, M_S decides the exact size of the replica by averaging the $maxsize$ values of the queries within the selected range.

Fairness in replication

To ensure fairness in replication, each MH M assigns a score σ to each data item d. σ essentially quantifies the *importance* of d to the network as a whole. Hence, σ should increase as d serves more MHs. Given d, M computes σ as follows. First, M sorts sorts the MHs, which recently requested d, in *descending* order of their access frequencies for d i.e., the first MH in this order made the most accesses to d. Given this order, M computes σ of d as follows:

$$\sigma = \left(\sum_{i=1}^{N_{MH}} (w_i \times n_i) \right) \times \rho \times \mu \qquad (1)$$

where N_{MH} is the number of MHs that recently accessed d, and n_i is the number of accesses to d by the i^{th} MH in the order specified. w_i is a weight coefficient, which equals (i/N_{MH}), thereby ensuring that more the number of MHs served by d, the more its score will be. Thus, given two data items with equal access frequencies, the score of the data item that serves a larger number of MHs would be higher. This is in contrast with existing works [5], which do not consider the origin of queries.

In Equation 1, ρ is the spatial density of the region from where the query originated. We consider ρ since we want to allocate a data item to spatially denser neighbourhoods, in order to serve more MHs. ρ = (Num_{MH}/$Area$), where Num_{MH} is the number of MHs in the region from which the query was issued and $Area$ is the area of the region. Notably, GN G knows Num_{MH} since every MH entering G's region needs to register with G. $Area$ is pre-defined w.r.t. the application and G knows the value of $Area$. For the tourist bus application, it could be either the area of the bus, or the area of some castle, museum or garden which the tourists of a bus are currently visiting. Each GN periodically computes its ρ and piggybacks this information in its periodic broadcast to all the MHs in its region.

μ is a weight factor for normalizing the data score w.r.t. image size. In M-P2P networks, users often issue queries for small-sized data items due to limited memory space in their mobile devices, hence more replicas for small-sized data items are likely to be allocated. Consequently, replicas may seldom be allocated for large-sized images and this would be unfair to users who issue queries for these images. μ ensures that larger-sized images would have a fair opportunity of being replicated. We consider three different ranges of image sizes, namely *small*, *medium* and *big*, for which we assign the values of μ to be 0.25, 0.5 and 0.75 respectively. These size ranges are application-dependent.

The score σ_G of a data item d (or replica) w.r.t. a given GN G is the sum of the scores of d at each MH within G's region. Hence, σ_G equals $(\sum_{i=1}^{\eta} \sigma_i)$, where η is the number of MHs in G's region, and σ_i is d's score at the i^{th} MH.

Prevention of thrashing conditions

To address prevention of thrashing, each MH keeps track of the number of deallocations of each replica at itself over a period of time. We define a metric designated as the Flip-Flop Ratio (FFR). The FFR of a replica r at a given MH M is computed as follows:

$$FFR = (N_{dealloc} \div T_{dealloc}) \times (size_r \div T_{size}) \qquad (2)$$

where $N_{dealloc}$ is the number of times that r has been deallocated at M, and $T_{dealloc}$ is the total number of deallocations of all the replicas at M during recent time period. $size_r$ is the size of the replica, while T_{size} is the sum of the sizes of all the replicas at M. Thus, the value of FFR is always between 0 and 1. We normalize FFR w.r.t. replica size to minimize the probability of thrashing of large data items since the effect of thrashing is more pronounced for such items due to bandwidth and energy considerations. Periodically, every MH computes the FFR for each replica r stored at itself. As the FFR value of r increases, the probability of thrashing of r also increases. Hence, r should not be deallocated if its FFR value exceeds a certain threshold ω, which is the average value of FFR across all the replicas in the network.

4 CADRE: A dynamic replication scheme for M-P2P networks

This section discusses the details of the CADRE replication scheme for M-P2P networks. In CADRE, each data item is owned by only *one* MH. Available memory space at each MH, bandwidth and data item sizes may vary. We define the**load** L_i of an MH M_i as its job queue size normalized by bandwidth to account for bandwidth heterogeneity.

Table 1 summarizes the notations used in this paper. Recall that query issuing MHs can distinguish between local and remote queries. An MH M_I issuing a local query stamps the query with the identifier of its corresponding GN G. On the other hand, M_I sends remote queries to G, which stamps such queries with the GN identifier. Using the notations in Table 1, queries in CADRE are of the form (img_{id}, M_I, GN_I, $maxsize, t$). Let us now examine how MH M_S, which serves the query request, maintains access statistics of queries issued to itself for facilitating replication.

Maintenance of access statistics at each MH

M_S distinguishes between accesses made to its own data items from within the region of its corresponding GN (i.e., *internal accesses*) and accesses to its own data items from MHs that are moving within the region of other GNs (i.e., *external accesses*). Hence, in Table 1, D_{int} and D_{ext} are lists in which M_S summarizes the internal accesses and external accesses respectively to its own data items. D_{int} guides M_S in selecting its own data items that should be replicated within the region of its corresponding GN, while D_{ext} facilitates M_S in determining its own data items that should be replicated at regions covered by other GNs. Using Table 1, each entry in D_{int} is of the form (img_{id}, M_I), while entries in D_{ext} are of the form (img_{id}, GN_I, M_I). M_S uses the entry of GN_I in D_{ext} to decide the GN, within whose region the given data item should be replicated.

M_S also differentiates between its own data items and the replicas that are stored at itself. Thus, in Table 1, R_{int} and R_{ext} are lists in which M_S summarizes the internal accesses and external accesses respectively to the replicas stored at itself. Using Table 1, each entry in R_{int} is of the form (img_{id}, M_I), while entries in R_{ext} are of the form (img_{id}, GN_I, M_I). Notably, here img_{id} refers to the identifier of the*replica* stored at M_S, while for the lists D_{int} and

Parameter	Signifi cance
img	A given queried image
img_{id}	Identifi er of img
M_I	Identifi er of the query issuing MH
M_S	Identifi er of the MH serving the query request
GN_I	Identifi er of the GN in whose region M_I is currently moving
$maxsize$	User-specifi ed maximum query result size
t	Time of query issue
D_{int}	List summarizing internal accesses to M_S's own data items at M_S
D_{ext}	List summarizing external access to M_S's own data items at M_S
R_{int}	List summarizing internal accesses to replicas at M_S
R_{ext}	List summarizing external access to replicas at M_S

Table 1. Summary of Notations

D_{ext}, img_{id} represented the identifi er of M_S's own data item. Besides this difference, the data structures of R_{int} and R_{ext} are essentially similar to that of D_{int} and D_{ext} respectively. R_{int} and R_{ext} guide M_S in computing replica scores w.r.t. different GNs, thereby facilitating M_S in deallocating replicas that have low scores w.r.t. a particular GN. Notably, the lists D_{int}, D_{ext}, R_{int} and R_{ext} are *periodically* refreshed to reflect *recent* access statistics. This is performed by periodically deleting all the existing entries from these lists and then re-populating them with fresh information from the recent queries. Such refreshing is especially important due to the dynamic changes in access patterns in M-P2P networks.

Selection of candidate data items for replication

Using its D_{int} and R_{int} respectively, each MH computes the score of each of its items (i.e., its own data items and replicas stored at itself), which were accessed by MHs from within the region of its corresponding GN G. Since D_{int} and R_{int} summarize the *internal accesses*, these scores are w.r.t. G. Similarly, from its D_{ext} and R_{ext} respectively, each MH calculates the score of each of its items, which were accessed by MHs that are outside the region of G. In this case, MHs from the respective regions corresponding to multiple GNs may have accessed a particular item, hence the scores of data items and replicas are computed w.r.t. each GN *separately*. *Periodically*, each MH sends all these scores to G. Upon receiving these scores from all the MHs in its region, G sums up the score of each item (w.r.t. each GN) from each MH within its region, thereby computing the total score of each item w.r.t. each GN.

Intuitively, internally accessed items should be replicated at MHs within G's region, while the externally accessed items need to be replicated at MHs in the regions of other GNs. Hence, when selecting candidate items for replica allocation, G distinguishes between internally accessed and externally accessed data items. For the internally accessed items, G sorts these items in descending order of their scores. G considers those items, whose scores exceed the average score ψ, as candidates for replication. ψ equals ($(1/N_d)\sum_{k=1}^{N_d}\sigma_j$), where N_d is the total number of items and σ_j is the score of the j^{th} item. Observe how G prefers items with relatively higher scores for replica allocation due to the higher importance of these items.

For the externally accessed items, G computes the score of each data item d w.r.t. every (external) GN from whose region at least one access was made for d. Then G creates a list $L_{Suggest}$ of these items, each entry of which is of the form (img_{id}, σ, GN_{id}), where img_{id} is the identifi er of the item, and σ is the score of the item w.r.t. GN_{id}, which is the identifi er of a given external GN. Then G sorts the items in $L_{Suggest}$ in descending order of σ. G considers items (of $L_{Suggest}$), whose scores exceed the threshold λ, as candidates for replication. (The remaining items are deleted from $L_{Suggest}$.) λ equals ($(1/N_d)\sum_{k=1}^{N_d}\sigma_j$), where N_d is the total number of items accessed by external GNs, and σ_j is the score of the j^{th} data item w.r.t a given external GN.

G does not participate in allocating replicas for the selected candidate items in $L_{Suggest}$. Instead, for each candidate item, G just sends a message to the *relevant* external GN, which will perform the actual replica allocation at some MH within its region. Given img_{id}, the relevant external GN is the corresponding GN_{id} in $L_{Suggest}$. Notably, just as G suggests external GNs to replicate items, the external GNs also suggest G to replicate items that have been accessed at these external GNs by the MHs of G's region. We shall henceforth refer to the list of items, for which G needs to allocate replicas, as I_{Rep}. Thus, I_{Rep} comprises (a) items that are stored at the MHs within its own region R (i.e., *internal items*) (b) items stored at MHs outside R (i.e., *external items*), which are recommended to G by the other (external) GNs.

The CADRE replication algorithm

In CADRE, each GN executes replica allocation and deallocation within the region that it covers. In addition to the scores of items, each MH also sends its load status, energy status, available memory space status and the FFR values of the replicas stored at itself to the corresponding GN in its region. Figure 1 depicts the CADRE replication algorithm, which is executed by a given GN G for allocating replicas at MHs within its own region. The list I_{Rep} in Figure 1 comprises items that are candidates for replica allocation by G. Line 1 of Figure 1 indicates that CADRE allocates replicas starting from the data item with the highest score, thus preferring data items with higher scores.

Lines 3-4 indicate that CADRE tries to replicate a given data item d either at the MH M_{max}, which made the maximum number of accesses to d or at one of M_{max}'s 1-hop

Algorithm *CADRE*

```
$I_{Rep}$: List of data items that are candidates for replica allocation
(1) Sort data items in $I_{Rep}$ in descending order of $\sigma$
(2) for each data item $d$ in $I_{Rep}$
(3)     Find the MH $M_{max}$ which has made maximum accesses to $d$
(4)     Add $M_{max}$ and its 1-hop neighbours to a set $Dest$
(5)     From $Dest$, delete overloaded MHs
(6)     From $Dest$, delete MHs with low remaining energy
(7)     From $Dest$, delete MHs with low available memory space
(8)     if $Dest$ is an empty list
(9)         break
(10)    else
(11)        Sort the MHs in $Dest$ in ascending order of load
(12)        for each MH $M$ in $Dest$
(13)            Create a list $L_R$ of the replicas stored at $M$
(14)            From $L_R$, delete replicas with FFR above threshold $\omega$
(15)            if $L_R$ is an empty list
(16)                break
(17)            else
(18)                Sort $L_R$ in ascending order of $\sigma$
(19)                $L_{dealloc}$ = empty /* List of deallocations */
(20)                $\sigma_{cnt}$ = 0, $size_{cnt}$ = 0
(21)                for each replica $r$ in $L_R$
(22)                    /* $size_d$ is $d$'s size and $size_r$ is $r$'s size */
(23)                    /* $\sigma_d$ is $d$'s score and $\sigma_r$ is $r$'s score */
(24)                    $\sigma_{cnt} = \sigma_{cnt} + \sigma_r$
(25)                    $size_{cnt} = size_{cnt} + size_r$
(26)                    if $\sigma_d < \sigma_{cnt}$
(27)                        break
(28)                    else
(29)                        if $size_d < size_{cnt}$
(30)                            Deallocate all entries in $L_{dealloc}$ from $M$
(31)                            Allocate $d$ at $M$
(32)                        else
(33)                            Add $r$ to $L_{dealloc}$
end
```

Figure 1. CADRE replica allocation algorithm

neighbours. This facilitates bringing d nearer to the origin of most of the requests for d. We also consider the 1-hop neighbours of M_{max} since M_{max} may not have adequate available memory space and/or remaining energy. Line 5 suggests that CADRE avoids replica allocation at overloaded MHs primarily because such MHs would not be able to provide good service due to their large job queues, which would force queries to incur long waiting times and consequently, higher response times. For similar reasons, CADRE allocates replicas of data items with relatively high scores to underloaded MHs (see Line 11). From Lines 6-7, observe how CADRE takes available memory space and energy of the MHs into consideration.

Lines 12-33 depict how the score of the item d to be replicated at the MH M is compared with the scores of the existing replicas. In essence, we are first *simulating* the eviction of replicas, and if d can be replicated at M by removing a set of existing replicas, whose combined score is less than that of d, we deallocate those existing replicas in favour of d. In particular, Line 14 indicates that we do not deallocate replicas whose FFR values exceed the pre-defined threshold ω, the primary reason being to avoid thrashing conditions. Incidentally, if there is still some available memory space at some MHs after the CADRE algorithm has been executed for all the candidate data items, the algorithm is executed multiple times until none of the MHs have adequate memory space for storing replicas.

5 Performance Evaluation

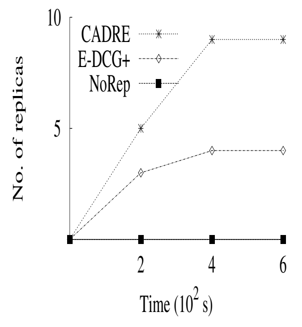
Our performance evaluation considers *five* different regions. Each region has 50 MHs and 1 GN. MHs in each region move according to the *Random waypoint model* [1] within the region, the area of the region being 1000 metre $\times$1000 metre. GNs move within their respective regions and are aware of each other's schedules, hence they are able to communicate with each other. Each region contains 200 data items that are uniformly distributed among 50 MHs i.e., each MH owns 4 data items. Each query is a request for either a local data item or a remote one. In all the experiments presented here, 60% of the queries were remote ones, while the other 40% were local queries.

Periodically, every TP seconds, each GN decides whether to perform replica allocation. Network topology does *not* change significantly during replica allocation since it requires only a few seconds [5]. 20 queries/second are issued within each region, the number of queries directed to each MH being determined by the Zipf distribution. Communication range of all MHs (except the GNs) is a circle of 100 metre radius. Table 2 summarizes the performance study parameters, which are the same for all five regions.

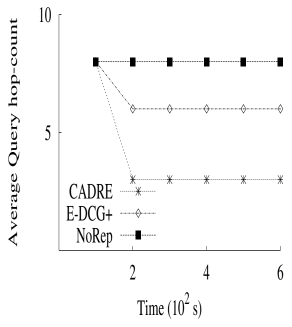
Parameter	Default value	Variations
No. of MHs (N_{MH})	50	
Zipf factor (ZF)	0.9	
Allocation period TP (10^2 s)	2	
Queries/second	20	
Bandwidth between MHs	28 Kbps to 100 Kbps	
Probability of MH availability	50% to 85%	
Size of a data item	1 MB to 10 MB	
Available Memory at an MH	10 MB to 20 MB	
Speed of an MH	1 metre/s to 10 metres/s	
Size of message headers	220 bytes	

Table 2. Performance Study Parameters

Performance metrics are **average response time** (**ART**) of a query, **data availability** (**DA**) and communication **traf-**

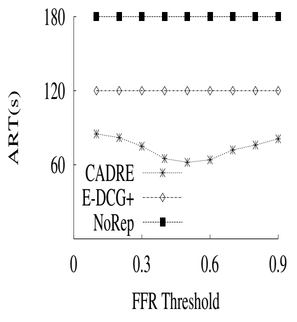


(a) No. of replicas

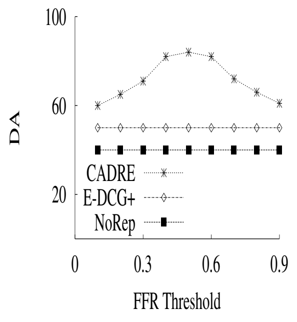


(b) Average query hop-count

Figure 2. Effect of fair replica allocation



(a) Average Response Time



(b) Data Availability

Figure 3. Effect of thrashing prevention

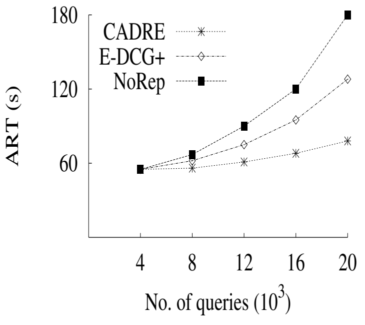
fi c(TR) for replica allocation. ART = $(1/N_Q)\sum_{i=1}^{N_Q}(T_f - T_i)$, where T_i is the time of query issuing, T_f is time of the query result reaching the query issuing MH, and N_Q is the total number of queries. DA = (N_S/N_Q)*100, N_S being the number of queries that were answered successfully. Each query has a 'hops-to-live' i.e., queries that are not answered within n hops are dropped. All our experiments use n = 4 since preliminary experiments indicated that it is a reasonable value for our application scenarios. We defi ne TR as the total hop-count for replica allocation during the experiment. As reference, we adapt the **E-DCG+** approach [5] (discussed in Section 2) to our scenario. E-DCG+ is executed at every replica allocation period. As a baseline, we also compare CADRE with an approach **NoRep**, which does not perform replica allocation.

Effect of fair replica allocation: We conducted an experiment to observe the number of replicas created by CADRE and E-DCG+ for a single 'hot' data item d over a period of time. This data item was selected randomly from the top 10% hottest data items. Figure 2a depicts the results. For both CADRE and E-DCG+, the number of replicas increases over time in response to conditions necessitating replica allocation. However, the number of replicas does not increase indefi nitely over time and eventually plateaus after some time due to competition among replicas for MH memory space. Observe that CADRE creates more replicas than E-DCG+. This is because CADRE would create a replica for a data item d, which is accessed by a large number of MHs, even if d's total access frequency is low, in which case E-DCG+ would not create any replica. Thus, CADRE creates replicas for more data items than E-DCG+ since CADRE selects candidate items for replication based on scores as opposed to total access frequencies.

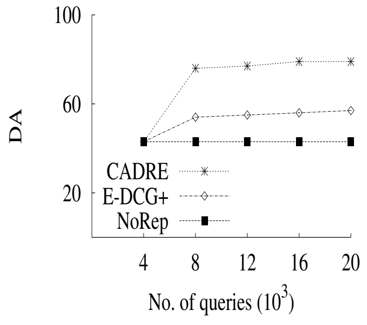
Figure 2b indicates the average number of hop-counts required for querying the same data item d during different periods of time. These results were averaged over a total of 1200 queries. After replica allocation has been performed, CADRE requires lower number of hops than E-DCG+ to answer queries on d since CADRE creates more replicas for d, as discussed for Figure 2a. More replicas generally decrease the querying hop-count since it increases the likelihood of queries being answered within lower number of hops and provide multiple paths to locate a queried data item. E-DCG+ requires lower number of querying hop-counts than NoRep essentially due to replication.

Effect of thrashing prevention: Recall that CADRE deallocates only those replicas, whose FFR values are less than that of the FFR threshold ω. Figures 3a and 3b depict the effect of variations in ω on the ART and DA of CADRE. E-DCG+ and NoRep show relatively constant ART and DA as these approaches are independent of ω. For high values of ω, the FFR of more replicas fall below ω, thereby making the occurrence of a large number of deallocations more likely. This is likely to lead to thrashing conditions, and consequently increased ART and decreased DA. However, when the value of $omega$ is low, the FFR of few replicas fall below ω. This makes deallocation too 'conservative' in the sense that replicas, which should have been deallocated to create memory space for 'hot' items, will not be deallocated, Thus, items with high scores cannot be allocated due to lack of space, thereby adversely affecting the performance of CADRE. As the results in Figures 3a and 3b indicate, CADRE performs best at intermediate values of ω i.e., 0.4 $\leq \omega \leq$ 0.6.

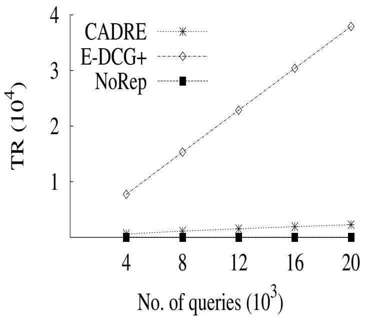
Performance of CADRE: We conducted a simulation experiment using default values of the parameters in Table 2. Figure 4 depicts the results. Figure 4a indicates that the performance gap between CADRE and E-DCG+ keeps increasing over time due to several reasons. First, in case of CADRE, GNs with good regional knowledge perform replica allocation and deallocaton, while for E-DCG+, replica allocation is performed in a distributed manner by individual MHs that lack good regional knowledge. Second, CADRE allocates replicas *only* to underloaded MHs,



(a) Average Query Response Time



(b) Data Availability



(c) Replica Allocation Traffi c

Figure 4. Performance of CADRE

thereby ensuring relatively short waiting times for queries at the job queues of these MHs, and consequently, reduced query response times. However, since E-DCG+ does not consider MH load, it may allocate replicas to overloaded MHs, thereby resulting in high query response times at these MHs due to their large job queues. Third, CADRE uses the FFR threshold to facilitate the prevention of thrashing conditions, which E-DCG+ does not address. Notably, preventing thrashing can improve performance signifi cantly when large-sized items (e.g., images) are present, as in our application scenarios.

The results in Figure 4b suggest that CADRE provides higher data availability than E-DCG+ essentially due to the reasons discussed for Figure 4a. Incidentally, during replica allocation, E-DCG+ requires every MH to broadcast its RWR values to every MH, thereby incurring O(N_{MH}^2) messages, where N_{MH} is the number of MHs. In contrast, CADRE requires each MH to send only one message to its corresponding GN, and the GN broadcasts a message to each MH, thus incurring O(N_{MH}) messages, which explains the results in Figure 4c.

6 Conclusion

We have proposed CADRE, a dynamic replication scheme for improving the typically low data availability in M-P2P networks. CADRE collaboratively performs both replica allocation and deallocation in tandem to facilitate optimal replication and to avoid 'thrashing' conditions, while addressing fair replica allocation across the MHs. CADRE deploys a hybrid super-peer architecture in which the GNs facilitate effi cient search and replication. Results of our performance study demonstrate that CADRE indeed improves query response times and data availability in M-P2P networks as compared to some recent existing schemes. In the near future, we plan to create an economy-based model for dynamic replication in M-P2P networks.

References

[1] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocol. *Proc. MOBICOM*, 1998.

[2] B. Carpentieri, M. Weinberger, and G. Seroussi. Lossless compression of continuous-tone images. *Proc. IEEE*, 88(11), 2000.

[3] A. Datta, M. Hauswirth, and K. Aberer. Updates in highly unreliable replicated peer-to-peer systems. *Proc. ICDCS*, 2003.

[4] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile data access through optimistic peer-to-peer replication. *Proc. ER Workshops*, 1998.

[5] T. Hara and S. Madria. Data replication for improving data accessibility in ad hoc networks. *To appear in IEEE Transaction on Mobile Computing*, 2006.

[6] http://compression.ca/act/.

[7] B. Kemme. Implementing database replication based on group communication. *Proc. Future Directions in Distributed Computing*, 2002.

[8] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *Proc. ACM TODS*, 25(3), 2000.

[9] A. Mondal, S. Madria, and M. Kitsuregawa. Clear: An effi cient context and location-based dynamic replication scheme for mobile-P2P networks. *Proc. DEXA*, 2006.

[10] V. Papadimos, D. Maier, and K. Tufte. Distributed query processing and catalogs for peer-to-peer systems. *Proc. CIDR*, 2003.

[11] E. Pitoura and B. Bhargava. Maintaining consistency of data in mobile distributed environments. *Proc. ICDCS*, 1995.

[12] D. Ratner, P. Reiher, G. Popek, and G. Kuenning. Replication requirements in mobile environments. *Proc. Mobile Networks and Applications*, 6(6), 2001.

[13] B. Richard, D. Nioclais, and D. Chalon. Clique: A transparent, peer-to-peer replicated fi le system. *Proc. MDM*, 2003.

[14] O. Wolfson, S. Jajodia, and Y. Huang. An adaptive data replication algorithm. *Proc. ACM TODS*, 22(4):255–314, June 1997.