Supporting disconnection operations through cooperative hoarding

Lai, Kwong; Tari, Zahir; Bertok, Peter

Lai, K., Tari, Z., & Bertok, P. (2005). Supporting disconnection operations through cooperative hoarding. Conference on Computer Communications and Networks (ICCCN), 35–42. https://doi.org/10.1109/ICCCN.2005.1523804

Published Version: https://doi.org/10.1109/ICCCN.2005.1523804

Supporting Disconnection Operations Through Cooperative Hoarding

Kwong Yuen Lai Zahir Tari Peter Bertok
School of Computer Science and Information Technology
RMIT University
Melbourne, Australia
Email: {kwonlai, zahirt, pbertok}@cs.rmit.edu.au

Abstract— **Mobile clients often need to operate while disconnected from the network due to limited battery life and network coverage. Hoarding supports this by fetching frequently accessed data into clients' local caches prior to disconnection. Existing work on hoarding have focused on improving data accessibility for individual mobile clients. However, due to storage limitations, mobile clients may not be able to hoard every data object they need. This leads to cache misses and disruption to clients' operations.**

In this paper, a new concept called *Cooperative Hoarding* is introduced to reduce the risks of cache misses for mobile clients. Cooperative hoarding takes advantage of group mobility behaviour, combined with peer cooperation in ad-hoc mode, to improve hoard performance. Two cooperative hoarding approaches are proposed that take into account access frequency, connection probability, and cache size of mobile clients so that hoarding can be performed cooperatively. Simulation results show that the proposed methods significantly improve cache hit ratio and provides better support for disconnected operations compared to existing schemes.

I. Introduction

Due to limited network coverage, signal interference and scarce battery power, mobile users are often disconnected from the rest of the network [1], [22]. Disconnected operation reduces the effect of intermittent connectivity by pre-load frequently accessed data objects into clients' caches, so that they can continue to access the data locally even when a connection is unavailable. However, as mobile devices have limited storage capacity, it is impossible for them to cache every data object available. This leads to the problem of critical cache misses during disconnection. To reduce the likelihood of cache misses and improve data accessibility for mobile clients, *Hoarding* and *Cooperative Caching* techniques have been proposed.

The goal of hoarding is to find the best set of objects to cache so that the probability of cache misses during disconnection is minimised. Existing hoarding techniques [9], [13], [20] rely on access order and frequency when making hoard selections. In these approaches, each client performs hoarding separately and only consider their own needs. As a result, when a cache miss occurs, a client must wait for a connection to be available before the required data can be accessed.

Cooperative caching reduces the chance of cache misses by allowing clients to contact their peers when a data object is not available locally. If a peer has cached the object, it can send the object back and prevent a cache miss. Existing work on cooperative caching [4], [18] have focused on reducing access delay and network traffic for wired networks. More recent studies ([3], [6], [15]) have addressed the issue of improving data accessibility, however, in these approaches, clients still perform hoarding separately and only consider cooperation after they have become disconnected. This reduces the effectiveness of their cooperative caches.

In this paper, we introduce a new concept called *Cooperative Hoarding* which combines the benefits of hoarding and cooperative caching to improve data accessibility for clients. Recent research have shown that mobile clients often move in groups [16], [21]. Examples of group mobility behaviour include soldiers travelling together on a battle field, rescue workers working at the scene of a disaster, students on a university campus etc. We refer to a group of clients travelling together as a *mobility group*. Cooperative hoarding takes advantage of the fact that even when disconnected from the rest of the network, clients in a mobility group may still be able to communicate with each other. By performing hoarding cooperatively, they can help each other during disconnected operation to improve the groups overall performance.

In this paper, two cooperative hoarding methods, Greedy Global Hoard (GGH) and Cooperative Access Probability-based hoarding (CAP), are proposed. In GGH, each client selects data objects to hoard based on their own access probabilities, the hoard content of its neighbour peers and the connection probability and hop distance from its neighbours. These parameters allow each client to utilise its hoard space efficiently by taking advantage of what its peers have already hoarded. In CAP, the best location to hoard each object is determined by a global cost function. This ensures access cost within the mobility group is minimised, while accounting for the possibility of partitioning within the group to reduce cache misses. Furthermore, by limiting the number of duplicate objects hoarded, more efficient use of the hoard space is achieved.

Extensive testing has been conducted to study the performance of the proposed methods. Test results confirm that cooperative hoarding is effective in supporting mobile clients during disconnected operations and significantly reduce the number of cache misses for disconnected clients.

The rest of this paper is organised as follows. Section II presents a survey of related work. The hoarding problem is formally described in Section III. Section IV presents the proposed methods, GGH and CAP. This is followed by the simu-

0-7803-9428-3/05/$20.00 ©2005 IEEE.

lation results in Section V. Lastly, conclusion and future work are discussed in Section VI.

II. Related Work

The Coda file system[11] is one of the earliest work to address the issue of hoarding. The hoarding mechanism relies on the user to specify the set of files to hoard. While this ensures accurate hoarding decisions are made, it is also very tedious for the user. To deal with this, automated hoarding techniques have been proposed [14], [19]. Automated hoarding techniques capture clients' access behaviour by recording the order and frequency in which data objects are accessed. Prior to disconnection, data mining techniques or graph analysis is used to predict which objects should be hoarded.

Although existing hoarding techniques improve data accessibility for individual clients, when an object cannot be found locally, a client still suffers the penalty of a cache miss. To overcome this, cooperative caching techniques (e.g. [8], [15], [23]) have been proposed to reduce the impact of local cache misses by allowing mobile clients to contact their peers to request an object when it cannot be found in its own cache.

Two cooperative caching schemes, CacheData and CachePath, designed for ad-hoc networks were proposed in [23]. In CacheData, mobile clients selectively cache copies of passing-by data objects when routing requests for other clients. This method reduces the overhead of future requests by caching frequently accessed objects near the requesting clients. The CachePath scheme addresses the issue of reducing object search overhead for cache cooperation. When a request is successfully routed, intermediate nodes will store the path used by the request. This ensure the correct path will be discovered quickly for future requests.

In [15], a peer-to-peer data sharing system called 7DS was proposed. When a client cannot find an object in its own cache, it will periodically broadcast requests to nearby peers until the needed object is found. It is assumed that mobile clients are willing to wait when a requested object cannot be found locally. But this assumption is often invalid because intuitively, clients would expect answers to their queries within a short time. 7DS also incurs high communication overhead due to periodic broadcasting of unanswered requests.

A difficult issue faced by cooperative caching in mobile networks is the problem of network partitioning. Network partitioning occurs because the movement of the clients often lead to division of the topology. As a result, clients in one partition cannot access information stored by clients in another partition. To deal with this, three replica allocation techniques, namely SAF, DAFN and DCG are proposed in [6]. In SAF, each client stores replicas of the data objects it most frequently access. While this incurs low replication overhead, multiple clients may cache duplicate objects, resulting in a waste in cache space. DAFN tries to solve this problem by requiring mobile clients to check with its neighbours before storing a replica. Although this eliminates duplication between neighbours, duplication can still existing between nodes that are more than one hop apart. Lastly, the DCG method groups mobile clients into highly stable groups, and performs replication based on this group structure. It is unclear how the stable/bi-connected groups are formed. Furthermore, all three methods in [6] only consider the access frequency when allocating replicas. The cost of retrieving the replicas and the connection probability between clients have not been accounted for in the models.

TABLE I

Reference to symbols

Symbol	Description
G	A mobility group
M	The number of clients in G
D	The set of data objects available at the server
N	The number of objects in D
m_i	A mobile client with ID=i
d_i	A data object with ID=i
$objSize$	The size of a data object
$\mathbf{F}$	Matrix representing the state clients' caches
$f_{i,j}$	Flag to indicate if m_i has cached d_j
$\mathbf{P}$	Access probability matrix
$p_{i,j}$	Probability of client i accessing object j on any given query
$\mathbf{L}$	Single hop connection probability matrix
$l_{i,j}$	Probability of m_i directly connected to m_j
$\xi_{i,j}$	A connection between m_i and m_j
$\mathbf{R}$	Multi-hop connection probability matrix
$r_{i,j}$	Probability of m_i connected to m_j over multiple hops
$\mathbf{H}$	Hop count matrix
$h_{i,j}$	Number of hops between m_i and m_j
$\pi_{i,j}$	All possible paths between m_i and m_j
$\rho_{i,j}$	The most reliable path between m_i and m_j
T	Set of processed nodes in Dijkstra's algorithm
c_{miss}	Cost of a cache miss
c_{hop}	Cost of transmitting an object over a single hop
$c_{i,j,k}$	Cost for m_i to access d_j from a peer m_k
Cost_i	Average access cost for m_i
S_i	Cache size of client m_i
$\mathbf{V}$	All possible hoard arrangements
τ	Combined connection probability from all clients to m_i

III. Problem Formulation

In this section, the cooperative hoarding problem and our system model are described. To help the reader follow our discussion, Table I lists all the symbols used in this section.

We consider a mobility group G that consists of M mobile clients. A client is denoted m_i where $i \in [1..M]$. Each client is equipped with a cache capable of storing S_i data objects. The set of data objects available for access from a central server is denoted D. There are N data objects in total, and a data object is denoted d_j where $j \in [1..N]$. For simplicity, we assume objects in D have the same size, $objSize$. Hoarding takes place prior to the clients disconnecting from the network and is performed with the help of the central server.

Let $\text{F}_i = \{\text{f}_{i,j} | \forall d_j \in D\}$ represent the state of client m_i's cache where:

$$f_{i,j} = \begin{cases} 1 & \text{if } d_j \text{ is cached by } m_i \\ 0 & \text{if } d_j \text{ is not found in } m_i \end{cases}$$

The access probability of each client is denoted:

$$\mathbf{P} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1..N} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ p_{M,1} & p_{M,2} & \cdots & p_{M,N} \end{bmatrix}$$

where $p_{i,j}$ is the probability of client m_i accessing objects d_j. The values in $\mathbf{P}$ can be obtained by keeping track of the number of previous accesses over time.

The connection probability matrix, $\mathbf{L} = \{l_{i,k} | \forall m_i, m_k \in G\}$, stores the probability of a connection existing between two clients. $l_{i,k}$ denotes the probability that m_i and m_k are connected over a single hop $\xi_{i,k}$. To obtain $\mathbf{L}$, each client periodically broadcasts a beacon signal containing its ID number. It also listens for beacons from other clients. By counting the number of beacons received from each peer over time, a client can estimate the probability that at any given time it will be connected to a particular peer.

Apart from single hop connections, connections over multiple hops are also considered. The multi-hop connectivity matrix is defined as $\mathbf{R} = \{r_{i,k} | \forall m_i, m_k \in G\}$, where $r_{i,k}$ denotes the probability that a connection of one or more hops exists between client m_i and m_k. Connection symmetry is assumed, therefore, $r_{i,k} = r_{k,i}$. The hop distance between two clients is represented by $\mathbf{H} = \{h_{i,k} | \forall m_i, m_k \in G\}$, where $h_{i,k}$ is the expected number of hops between m_i and m_k.

In order to obtain $\mathbf{R}$ and $\mathbf{H}$, it is necessary to traverse through all possible paths between every pair of clients within the mobility group. While this is possible for mobility groups with only a few clients, its computation overhead is very high for larger groups. To deal with this problem, we relax the definition of $\mathbf{R}$ and $\mathbf{H}$, and propose a modified version of Dijkstra's shortest path algorithm to obtain their values. The relaxed definitions are given as follows:

Given two client, m_i and m_k and the set of possible paths between them, $\pi_{i,k}$, the most reliable path between the two clients is denoted as $\rho_{i,k}$. The connection probability of $\rho_{i,k}$ is equal to:

$$Pr(\rho_{i,k}) = \prod_{\forall \xi_{x,y} \in \rho_{i,k}} (l_{x,y}) \tag{1}$$

such that,

$$Pr(\rho_{i,k}) = \max \left(\prod_{\forall \xi_{x,y} \in \pi} (l_{x,y}) \right) \quad \forall \pi \in \pi_{i,k} \tag{2}$$

Based on Equations 1 and 2, we redefine $r_{i,k} = Pr(\rho_{i,k})$. So now $\mathbf{R}$ stores the probability of two clients m_i and m_k being connected via $\rho_{i,k}$. We also redefine the hop count matrix, $\mathbf{H}$, so that $h_{i,k}$ is the number of hops in $\rho_{i,k}$.

With $\mathbf{R}$ and $\mathbf{H}$ redefined, we can use a modified version of Dijkstra's algorithm to obtain their values. The original Dijkstra's algorithm [5] was designed to find the shortest paths connecting every source-destination pair in a network. It is chosen for our model because it is simple to implement and has a low complexity. The modified algorithm is performed as follows. First, a graph is constructed from $\mathbf{L}$. For example, given:

$$\mathbf{L} = \begin{bmatrix} 1.0 & 0.9 & 0.8 & 0.1 & 0.0 \\ 0.9 & 1.0 & 0.15 & 0.0 & 0.0 \\ 0.8 & 0.15 & 1.0 & 0.8 & 0.0 \\ 0.1 & 0.0 & 0.8 & 1.0 & 0.2 \\ 0.0 & 0.0 & 0.0 & 0.2 & 1.0 \end{bmatrix}$$

Figure 1 shows the equivalent network graph.

The modified Dijkstra's algorithm as shown in Algorithm 1 is applied to each node in the graph to obtain $\mathbf{R}$ and $\mathbf{H}$. This

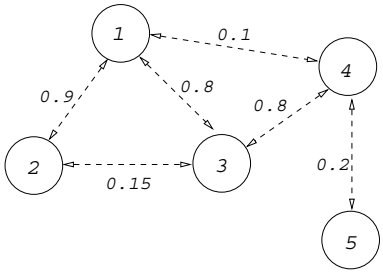

Fig. 1. *A connection probability graph*

algorithm is different to Dijkstra's algorithm because it does not maintain the hop count of the *shortest* path from the source node. Instead, in each iteration of this algorithm, $r_{i,j}$ and $h_{i,j}$ are used to maintain the connection probability and hop count of the *most reliable* path from the source node m_i to each of the unprocessed nodes ($m_x \notin T$). Table II shows the result of applying the algorithm on node 1.

Algorithm 1 Modified Dijkstra's Algorithm

Let :
$G = \{m_0, m_1, .., m_M\}$ denote the set of mobile clients
T denote the set of clients already processed
$l_{i,j}$ probability of direct connection between m_i to m_j
$r_{i,j}$ denotes probability of the most likely path from m_i to m_j
$h_{i,j}$ denotes number of hops of the most likely path from m_i to m_j

Repeat for each $m_i \in G$:
Let $T = \{m_i\}$
$r_{i,j} = \begin{cases} 0 & \text{if } i = j \\ l_{i,j} & \text{otherwise} \end{cases}$
$h_{i,j} = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \text{ and } l_{i,j} > 0 \\ \infty & \text{if } i \neq j \text{ and } l_{i,j} = 0 \end{cases}$
Repeat until $T = G$:
Find $m_x \notin T$ such that $r_{i,x} = \max(r_{i,k}) \quad \forall k \notin T$
$T = T \cup m_x$
$h_{i,j} = \begin{cases} h_{i,j} & \text{if } r_{i,j} \geq r_{i,x} * l_{x,j} \quad \forall j \notin T \\ h_{i,x} + 1 & \text{otherwise} \end{cases}$
$r_{i,j} = max[r_{i,j}, r_{i,x} * l_{x,j}] \quad \forall j \notin T$

Given $\mathbf{R}$ and $\mathbf{H}$, the cost for each client to access an object in D can be represented as, $\mathbf{C}$:

$$\mathbf{C} = \{\mathrm{c}_{i,j,k} | \forall m_i, m_k \in G, d_j \in D\} \tag{3}$$

where $c_{i,j,k}$ is the cost for client m_i to access an object d_j from the cache of a peer m_k. $c_{i,j,k}$ is calculated as follows:

$$\mathrm{c}_{i,j,k} = \begin{cases} c_{miss} & \text{if } f_{k,j} = 0 \\ 0 & \text{if } i = k \text{ and } f_{k,j} = 1 \\ c_{hop} \times h_{i,k} \times r_{i,k} + \\ c_{miss} \times (1 - r_{i,k}) & \text{if } i \neq k \text{ and } f_{k,j} = 1 \end{cases} \tag{4}$$

where c_{hop} represents the cost of transmitting an object over one hop, and c_{miss} represent the cost of a cache miss. It is assumed the cost of accessing an object from local cache is negligible and the cost of accessing an object from a peer is proportional to the hop distance to the peer ($h_{i,k}$) and the connection probability ($r_{i,k}$). Based on Equation 4, the mean access cost

TABLE II

APPLICATION OF ALGORITHM 1 FOR $i = 1$

Iteration	x	T	Path	$r_{i,2}$	$h_{i,2}$	Path	$r_{i,3}$	$h_{i,3}$	Path	$r_{i,4}$	$h_{i,4}$	Path	$r_{i,5}$	$h_{i,5}$
Iinitialisation	1	{1}	1-2	0.9	1	1-3	0.8	1	1-4	0.1	1	-	0	∞
1	2	{1,2}	1-2	0.9	1	1-3	0.8	1	1-4	0.1	1	-	0	∞
2	3	{1,2,3}	1-2	0.9	1	1-3	0.8	1	1-3-4	0.64	2	-	0	∞
3	4	{1,2,3,4}	1-2	0.9	1	1-3	0.8	1	1-3-4	0.64	2	1-3-4-5	0.128	3
4	5	{1,2,3,4,5}	1-2	0.9	1	1-3	0.8	1	1-3-4	0.64	2	1-3-4-5	0.128	3
				0.9	1		0.8	1		0.64	2		0.128	3

for a client without using cache cooperation is equal to:

$$\text{Cost}_i = \sum_{j=1}^{N} p_{i,j} \times \text{c}_{i,j,i} \tag{5}$$

If cache cooperation is used, the mean access cost is:

$$\text{Cost}_i = \sum_{j=1}^{N} p_{i,j} \times \min(\text{c}_{i,j,k} | \forall m_k \in G) \tag{6}$$

Finally, based on Equation 5 and 6, the goal of the cooperative hoarding algorithm is to find the best hoard arrangement in order to optimise the following objective function :

$$\min \left(\sum_{i=1}^{M} (\text{Cost}_i) \right) \tag{7}$$

subject to :

$$\sum_{j=1}^{N} (f_{i,j}) \leq S_i \qquad \forall m_i \in G \tag{8}$$

where S_i is the cache size of client m_i.

If cache cooperation is not applied, Equation 7 is minimised if each client caches the objects it most frequently accesses. That is, find the set of objects which maximises $\sum_{j=0}^{N}(p_{i,j})$ for $i \in [1..M]$.

For the case where cache cooperation is used, one can attempt to exhaustively search through all possible hoard arrangements. However, let $\mathbf{V}$ denote the set of all possible hoard arrangements, the size of $\mathbf{V}$ is equal to:

$$|\mathbf{V}| \quad = \quad \sum_{i=1}^{M} \binom{N}{S_i} \tag{9}$$

Clearly, even for very small values of N, M and S_i, the number of possible hoard arrangements is extremely large (as an example, given N=100, M=5, S_i = 10, we have $|V| = 1.73 \times 10^{13}$). As a result, a solution based on exhaustive search is infeasible.

IV. PROPOSED METHODS

This section describes two cooperative hoarding algorithms, namely GGH (The Greedy Global Hoard method) and CAP (Cooperative Access Probability-based method), to solve the cooperative hoarding problem presented in the previous section.

In GGH and CAP, hoarding is performed while clients are strongly connected to the network. A more powerful node (e.g. the information server or base station) performs the calculations in the algorithm and informs the clients on which objects to hoard. This is especially beneficial if the number of clients in the mobility group is large as it reduces the computational load on the clients. Another advantage of performing the hoarding algorithm with the help of server is that the server can record which objects have been hoarded by a mobility group. If a new client wishes to join the group later on, the server can use this information to help select a suitable hoard arrangements.

A. The Greedy Global Hoard method (GGH)

The GGH method improves the effectiveness of hoarding by providing each client with the knowledge of what their peers have already hoarded. The algorithm is outlined below:

1) Calculate the global connection probability τ_i of each client m_i, where τ_i is defined as :

$$\tau_i = \sum_{j=0}^{M} (r_{i,j}) \tag{10}$$

A client m_s is chosen as the starting node such that :

$$\tau_s = max(\tau_i) \qquad \forall m_i \in G \tag{11}$$

That is, the client with the highest connection probability with all other nodes in the group is chosen.

2) m_s calculates a cost value $cost_{s,j}$ for each object in D. This cost value represents the average penalty paid per query if d_j is not cached by m_s and is calculated as:

$$cost_{s,j} = p_{s,j} \times c_{miss} \tag{12}$$

3) Given $\{cost_{s,j} | \forall d_j \in D\}$, m_s fills its cache with objects starting from those with the highest $cost_{s,j}$ to the lowest $cost_{s,j}$ until its cache is full.
4) m_s then constructs an $M \times N$ matrix, $\mathbf{F} = \{f_{i,j} | \forall i \in [1..M], \forall j \in [1..N]\}$ to indicate which objects it has hoarded, where :

$$f_{s,j} = \begin{cases} 1 & \text{if } d_j \text{ is cached by } m_s \\ 0 & \text{if } d_j \text{ is not cached by } m_s \end{cases}$$

5) $\mathbf{F}$ is sent to the next client in a breadth first traversal of the mobility group based on the network graph constructed from $\mathbf{L}$ (similar to the one shown in Figure 1).
6) When a client (say m_i) receives $\mathbf{F}$, it recalculates $cost_{i,j}$ for all objects with the help of $\mathbf{F}$ as follows:

$$\text{cost}_{i,j} = \begin{cases} p_{i,j} \times c_{miss}, & \text{if } \nexists m_k \in G \text{ where } f_{k,j} = 1 \\ p_{i,j} \times \min\left(c_{i,j,k} | \forall m_k \in G\right) & \text{, otherwise} \end{cases} \tag{13}$$

7) The objects with the highest $cost_{i,j}$ are cached by m_i until its cache is full.
8) Once m_i has finished filling its cache, it updates $\mathbf{F}$ and passes it to the next client in the breadth first traversal order. The process is repeated from step 6 for the newly selected client. This continues until every client in the group has filled their caches.

The use of $\mathbf{F}$ allows clients to take into account which objects have already been cached by others in the group. The algorithm is greedy because each client attempts to maximises the effectiveness of its own hoard by caching objects which gives the best cost value for itself without considering the access probability of others. However, because each client takes advantage of objects already hoarded by its peers, the overall hoard performance of the group is improved.

B. The Cooperative Access Probability-based method (CAP)

Unlike the GGH method, CAP is non-greedy. The idea is to find the best location to cache each object so that the global access cost within the group is minimised. The CAP algorithm is outlined below:

1) First, the average access probability of each object within D is calculated. The average access probability of an object d_j is defined as :

$$g_j = \frac{\sum_{i=1}^{M}(p_{i,j})}{M} \tag{14}$$

We denote the set of objects which have an average access probability greater than 0, $\mathbf{B}$ (i.e. $\mathbf{B} = \{d_j | g_j > 0\}$).
2) Let $v_{k,j}$ denote the global access cost if object d_j is cached by a client m_k:

$$v_{k,j} = \sum_{i=1}^{M} p_{i,j} \times c_{i,j,k} \tag{15}$$

where $c_{i,j,k}$ is the access cost for client m_i to access object d_j from a peer m_k (as defined in Equation 4).
3) For each object $d_j \in \mathbf{B}$, starting from the one with the highest g_j value, a client m_i is identified such that :

$$v_{i,j} = \min(v_{x,j}) \quad \forall m_x \in G \tag{16}$$

4) The selected client (m_i) represents the best location to store d_j, which minimises the total cost to access d_j for the whole group. A copy of d_j is placed in the cache of m_i. If the cache of m_i is full, then d_j is given to the node with the next lowest $v_{i,j}$ value as defined in Equation 16.
5) Steps 3 and 4 are repeated for all $d_j \in \mathbf{B}$ in order of their average access probability until the caches of all the clients in the mobility group are filled.
6) If the number of objects in $\mathbf{B}$ is less than the total client cache size $|S|$ (where $|S| = \sum_{i=1}^{M} S_i$), then after every object has been cached once, the process starts again from the most frequently accessed to the least frequently accessed object. This time however, each object is stored at the best remaining location.

The CAP algorithms finds the best location to cache each object based on its global access probability. Since the function for selecting the location to hoard each object (i.e. Equation 16) also accounts for the access cost and connection probability between the clients in the group, CAP is able to reduce the overall distance objects need to be transferred when clients fetch objects from their peers. Furthermore, since no replication takes place until every object is cached at least once, the CAP method is likely to store more unique objects than the GGH method.

V. Simulation Results

In order to evaluate the performance of the proposed methods, extensive testing has been performed. The following schemes are tested in our simulation:

- $noCoop$ - Clients hoard random objects and do not share their caches during disconnected operations.
- $Coop$ - Clients hoard random objects, but cache cooperation is used once disconnected.
- Dynamic Access Frequency and Neighbourhood (DAFN)[6] - each client hoards the most frequently accessed objects while taking into account what its direct neighbours have cached.
- The proposed methods - GGH and CAP

The simulation model consists of N mobile clients travelling in a mobility group. At the start of the simulation, the group is connected to a central server. Clients perform hoarding according to the algorithm being tested. Once hoarding is completed, the group is disconnected from the server. During disconnected operations, a client m_i has a probability of $l_{i,j}$ of being directly connected to another client m_j. The value of $l_{i,j}$ is uniformly distributed between 0 and a parameter $P_{connect}$. $P_{connect}$ represents the level of connectivity in the group. AODV [17] is used as the routing protocol for communication between clients. Each simulation run is composed of 1000 queries randomly distributed among the clients. Queries are generated based on the Zipf distribution [2] distributed among data objects on the server. A random offset is applied to the Zipf distribution for each client to model variation in their access patterns. The simulation parameters and their default values are listed in Table III.

TABLE III
Simulation Parameters

Parameter	Value	Range
Number of simulation runs per data point	30	
Simulation duration	1000 queries	
Number of clients (M)	10	1 - 30
Client Cache size (S_i)	50 objects	10 - 100
Number of objects (N)	1000	
Cost of transmitting an object over a single hop	1	
Cost of a cache miss	100	
$P_{connect}$	0.7	0.05 - 1.0
Zipf parameter(α)	0.7	0.4 - 1.6

The schemes tested are compared in terms of the average cache hit ratio and the average time of operation before the

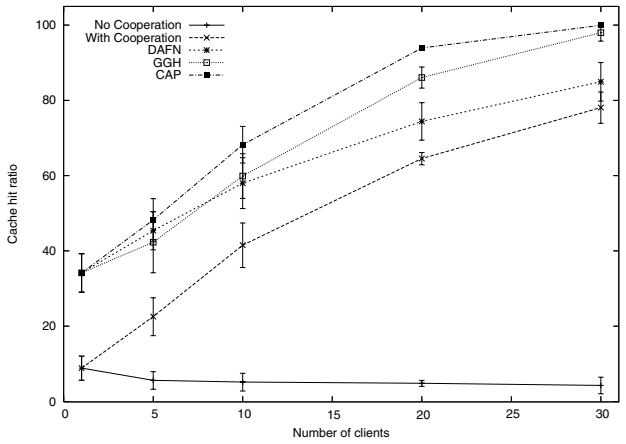

Fig. 2. Cache Hit Ratio vs. Number of clients

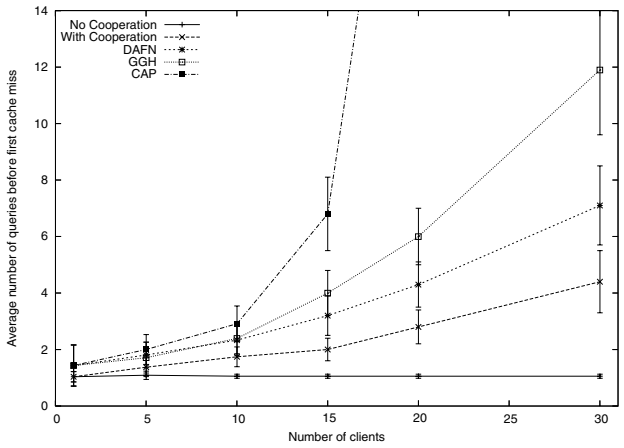

Fig. 3. Average number of queries before cache miss vs. Number of clients

occurrence of the first cache miss. These two metrics are important because the goal of hoarding is to reduce the impact of disconnection on mobile clients' operations. Therefore a good hoarding algorithm should achieving high cache hit ratio and ensure clients can operate for long period before the occurrence of any cache miss.

A. The effect of the number of clients

The average cache hit ratio is plotted against the number of clients in Figure 2. When the number of clients in the mobility group equals 1, DAFN, GGH and CAP have similar performance which is much higher compared to $Coop$ and $NoCoop$. This is because $Coop$ and $NoCoop$ hoard data objects randomly without considering access probability. As the number of clients increases, all the simulated approaches, except for $NoCoop$ achieve increasing cache hit ratio. The reason for this is that as the number of clients increases, clients are likely to be connected to more peers, which gives them better chances of finding the object they need from a peer. This reduces the number of cache misses experienced by the clients. We found that in nearly all cases, CAP and GGH perform much better compared to the other approaches.

Figure 3 shows the average number of queries before cache miss against the number of clients. As expected, the number of clients has no effect on the performance of $NoCoop$ because in this scheme, clients do not cooperate with each other. On the other hand, as the number of clients increases, the proposed schemes CAP and GGH are able to extend the time until first cache miss significantly. While DAFN and $Coop$ also perform well, their increase is not as great as CAP and GGH.

B. The effect of client cache size

Figure 4 shows the effect of different client cache size on average cache hit ratio. Client cache size is expressed as a percentage of the number of objects on the server in this graph. As expected, cache hit ratio increases as client cache size increases. The increase is not linear however, because clients' access patterns are skewed due to the Zipf distribution. The graph shows that even with simple cache cooperation ($Coop$), cache hit ratio can be improved significantly compared to the case where clients do not cooperate ($NoCoop$). This is because in the event of a cache miss, clients are able to contact

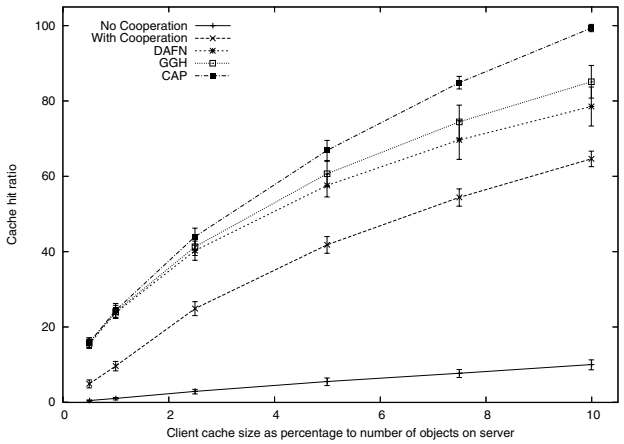

Fig. 4. Cache Hit Ratio vs. Cache Size

their peers to request the needed object when cooperation is applied. Furthermore, by using hoarding cooperatively, the average cache hit ratio achieved by the group is improved as hoard spaced are shared by the clients to maximise the number of objects hoarded. The proposed methods, GGH and CAP, both performed better than existing methods. This is especially the case for CAP where the improvement ranges from 9% to 22% compared to DAFN.

In Figure 5, the average number of queries before clients experience their first cache miss is plotted against client cache size. This is an important performance measure because if a client can process more queries before the first cache miss occurs, it is likely to be able to operate for longer before a critical cache miss. It can be seen from the graph that while most approaches have similar performances, CAP's performance improves the most as client cache size increases. This is because in the CAP scheme, duplicate copies of data objects are only cached when there is at least one copy of every object available. As a result, it maximises the number of unique objects hoarded in the limited hoard space. Furthermore, by allowing clients to contact their peers when a cache miss occurs, the cooperative methods extend the time clients can operate before cache misses occur compared to the non-cooperative methods.

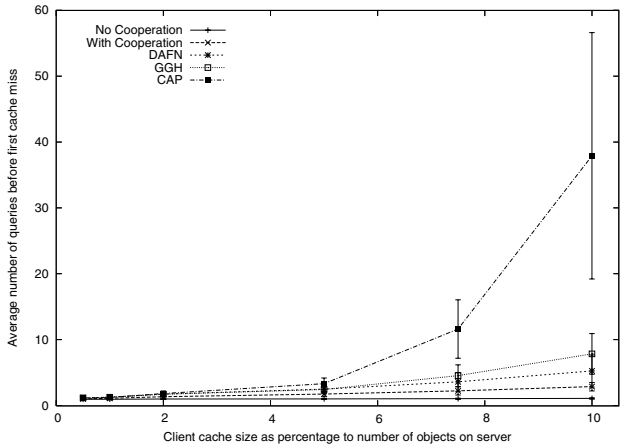

Fig. 5. Average number of queries before cache miss vs. Cache Size

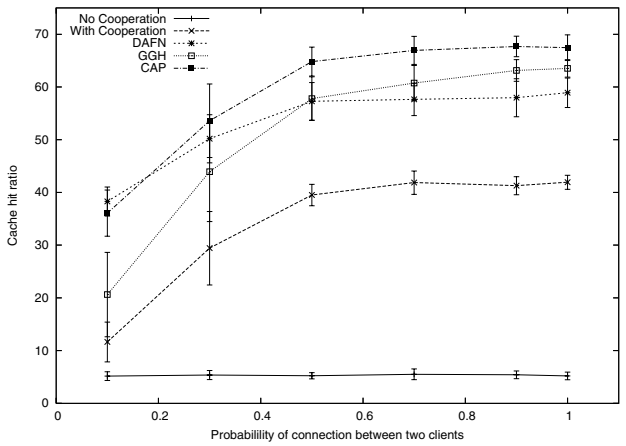

Fig. 6. Cache Hit Ratio vs. Connection probability

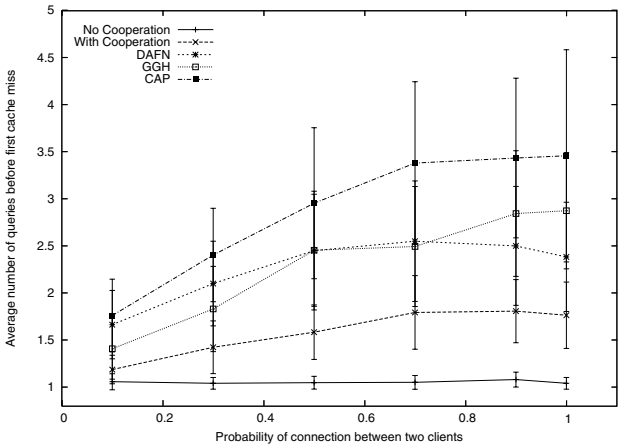

Fig. 7. Average queries before cache miss vs. Connection probability

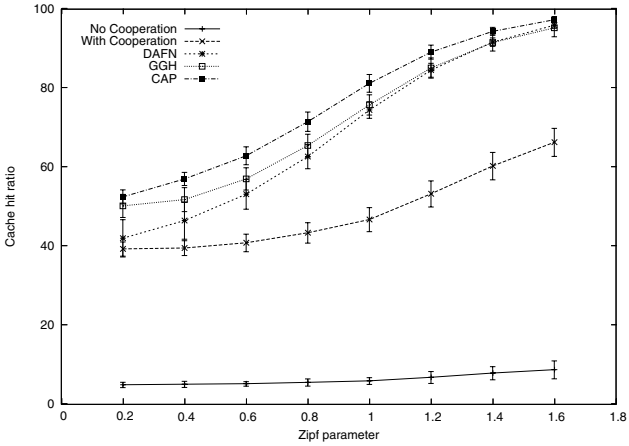

Fig. 8. Cache hit ratio vs. Zipf parameter

C. The effect of connection probability

Next, the effect of different connection probability on hoard performance is studied. Higher connection probability represents scenarios where clients are strongly connected and are travelling within close proximity of each other. While, low connection probability may represent cases where connections are poor due to interference or where clients operate in "doze" mode frequently to preserve energy.

As can be seen from Figure 6, cache hit ratio is much lower when connection probability is low. This is because with a low connection probability, clients are often disconnected from each other. This results in network partitioning which leads to a drop in the effectiveness of cooperative caching. While CAP provides the best performance overall, in the case where connection probability is very low (less than 0.15 in our experiment), the performance of CAP drops slightly. This is because the CAP approach relies heavily on mobile clients helping each other. When connection probability is low, clients are unable to assist each other, resulting in a drop in cache hit ratio.

Figure 7 shows the average number of queries before cache miss against the connection probability. Again, a drop in connection probability lead to earlier cache misses because with weak connectivity between clients, they are unable to help each other in the event of local cache misses. In all cases, CAP is able to provide the best performance.

D. The effect of access skewness

The graphs in this section illustrate how well the proposed methods adapt to access patterns with different degree of locality. It is important because depending on the application, the proposed methods may need to work in scenarios with high access skewness or a uniformly spread access pattern. We model access locality using the Zipf distribution which has an α parameter which determines the skewness of the distribution. A high α value (i.e. $\alpha = 1.6$) means that client queries are concentrated on a small number of data objects. On the other hand, a small α ($\alpha < 0.5$) means client queries are distributed over a wide range of objects.

Figure 8 shows the average client cache hit ratio against α. It can be seen that at lower α values, the cache hit ratio of all schemes is significantly lower than when α is large. this is because when α is small, client queries are distributed over many different objects, making caching less effective. For large α values, very high cache hit ratio can be achieved as most queries concentrated on a few objects cached locally by each client.

The average number of queries before cache miss is plotted against α in Figure 9. The graphs shows that because the proposed methods takes into account access probability when performing hoarding, an increase in α leads to a better cache hit ratio, which in turn defers the occurrence of the first cache miss for clients.

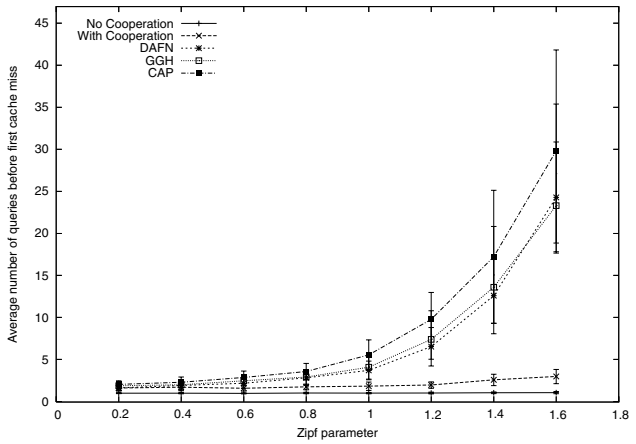

Fig. 9. Average number of queries before cache miss vs. Zipf Parameter

VI. Conclusion and Future work

In this paper, we have introduced the concept of cooperative hoarding to improve data accessibility for group of mobile clients travelling together. Two cooperative hoarding schemes, GGH and CAP, are proposed. In GGH, each client performs hoarding based on its own access probabilities and the knowledge of which objects have already been hoarded by its peers. Depending on whether it is more beneficial to cache an object locally or to access it remotely from a peer, GGH allows each client to make hoarding decisions which improves its own hoard performance. CAP is a non-greedy method where clients attempt to perform hoarding as a group by maximising a global access probability measure. Replicas of objects are only cached when every object is cached at least once. This maximises the number of unique objects hoarded within the group, thus improving the groups cache hit ratio.

Extensive simulation has been performed to compare GGH and CAP to existing schemes. It is found that while both proposed schemes are able to improve cache hit ratio and delay the occurrence of the first cache miss, CAP performs better than GGH by also providing lower average access cost.

As part of our future work, we intend to collect real work traces and test the proposed schemes under a more realistic environment instead of using workloads generated from mathematical distributions.

Acknowledgement

This project is supported by the ARC (Australian Research Council - under the Linkage-Project scheme, no. LP0455234) and SUN Microsystems grant no.7832-030217-AUS.

References

[1] D. Barbara and T. Imielinski. Sleepers and workaholics: caching strategies for mobile environments. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 1–12, 1994.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the INFOCOM*, pages 126–134, 1999.

[3] G. Cao, L. Yin, and C. R. Das. Cooperative cache-based data access in ad hoc networks. *IEEE Computer*, pages 32–39, 2004.

[4] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pages 267–280, 1994.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1, 1959.

[6] T. Hara. Effective replica allocation in ad hoc networks for improving data accessibility. In *Proceedings of the IEEE INFOCOM*, pages 1568–1576, April 2001.

[7] T. Hara. Replica allocation methods in ad hoc networks with data update. *Mobile Networks and Applications*, 8:343–354, 2003.

[8] T. Hara, N. Murakami, and S. Nishio. Replica allocation for coreelated data items in ad hoc sensor networks. *SIGMOD Record*, 33(1):38–43, March 2004.

[9] A. S. Helal, A. Khushraj, and J. Zhang. Incremental hoarding and reintegration in mobile environment. In *Proceedings of the 2002 Symposium on Applications and the Internet*, pages 8–11, January 2002.

[10] J. Huang, M. Chen, and W. Peng. Exploring group mobility for replica data allocation in a mobile environment. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM'03)*, pages 161–168, November 2003.

[11] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

[12] U. Kubach and K. Rothermel. Exploiting location information for infostation-based hoarding. In *Proceedings of the 7th ACM SIGMOBILE Annual International Conference on Mobile Computing and Networking (MobiCom 2001)*, pages 15–27, 2001.

[13] G. Kuenning. The design of the SEER predictive caching systems. In *Proceedings of Mobile Computing Systems and Applications*, 1994.

[14] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. In *Proceedings of the 16th Symposium on Operating Systems Principles*, pages 264–275, 1997.

[15] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *Proceedings of the ACM Symposium on Mobile Ad-hoc Networking and Computing*, pages 117–127, October 2001.

[16] W. Peng and M. Chen. Allocation of shared data based on mobile user movement. In *Proceedings of the IEEE International Conference on Mobile Data Management*, pages 105–112, 2002.

[17] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc on demand distance vector (aodv) routing. In *IETF RFC 3561*, 2003.

[18] P. Sarkar and J. Hartman. Efficient cooperative caching using hints. In *Proceedings of the ACM Symposium on Operating Systems Design and Implementation*, pages 35–46, 1996.

[19] Y. Saygin, O. Ulusoy, and A. Elmagarmid. Association rules for supporting hoarding in mobile computing environments. In *Association Rules for Supporting Hoarding in Mobile computing environments*, 2000.

[20] C. D. Tait, H. Lei, S. Acharya, and H. Chang. Intelligent file hoarding for mobile computers. In *Mobile Computing and Networking*, pages 119–125, 1995.

[21] K. H. Wang and B. Li. Group mobility and partition prediction in wireless ad-hoc networks. In *Proceedings of the IEEE International Conference on Communication (ICC'02)*, pages 1017–1021, April 2002.

[22] K. Yasuda. Cache cooperation for clustered disconnected computers. In *Proceedings of the IEEE International Conference on Parallel and Distributd Systems*, pages 457–464, 2002.

[23] L. Yin and G. Cao. Supporting cooperative caching in ad hoc networks. In *Proceedings of the IEEE INFOCOM*, March 2004.