World Wide Web (2011) 14:243–259
DOI 10.1007/s11280-010-0103-3

Benefit based cache data placement and update for mobile peer to peer networks

Fan Ye · Qing Li · Enhong Chen

Received: 31 August 2010 / Revised: 22 November 2010
Accepted: 23 November 2010 / Published online: 8 December 2010

Abstract Mobile Peer to Peer (MP2P) networks provide decentralization, self-organization, scalability characters, but suffer from high latency and link break problems. In this paper, we study the cache/replication placement and cache update problems arising in such kind of networks. While researchers have proposed various replication placement algorithms to place data across the network to address the problem, it was proven as NP-hard. As a result, many heuristic algorithms have been brought forward for solving the problem. In this article, we propose an effective and low cost cache placement strategy combined with an update scheme which can be easily implemented in a decentralized way. The contribution of this paper is the adaptive and flexible cache placement and update algorithms designed for real MP2P network usage. The combination of MP2P cache placement and update is the novelty of this article. Extensive experiments are conducted to demonstrate the efficiency of the cache placement and update scheme.

Keywords benefit based · mobile data cache · cache placement · cache update

F. Ye · E. Chen
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

E. Chen
e-mail: cheneh@ustc.edu.cn

F. Ye · Q. Li · E. Chen
Web Service, Joint Research Lab of Excellence, CityU-USTC Advanced Research Institute, Suzhou, China

Q. Li
e-mail: itqli@cityu.edu.hk

F. Ye (✉) · Q. Li
Department of Computer Science, City University of Hong Kong, Hong Kong, China
e-mail: yfan@mail.ustc.edu.cn

1 Introduction

The recent years have witnessed a rapid development of mobile computing and wireless communication technologies. Mobile Peer to Peer (MP2P) network is a very popular kind of network composed of small computing devices with wireless interfaces; the devices can be notepad or even embedded processors such as sensors. From the physical layer's aspect, a MP2P network can be regarded as an ad hoc network supporting multi-hop routing. Every node can conduct package forwarding, package cache and so on. Our research emphasis is different from those on conventional ad hoc networks, as we are more concerned with such aspects as efficient data access, data sharing which are more related to the application layer's service provision. More specifically, in this paper, we concentrate on the issues of optimally placing cache and effective update support, aiming to demonstrate the relationship and mutual promotion of these two techniques in a MP2P network.

There are two reasons to study optimal placement of caches in a MP2P network: Firstly, a MP2P network is a multi-hop wireless network for sharing data among remote nodes, and multi-hop access of information (e.g., multi-data items) typically occurs via multi-hop routing; obviously, the overall cost of data access can be greatly reduced through caching relevant data on the relay nodes. Secondly, a MP2P network is generally resource constrained in terms of channel bandwidth and battery power of the nodes. Caching can help reduce communication cost, resulting in possible bandwidth and battery energy savings. This paper thus takes the stance of cooperative cache of multi-data items in a MP2P network.

Let's consider a popular application of multimedia data sharing as an example. Suppose there are multiple multimedia files (each multimedia file is segmented when it is too large) to be shared among a number of mobile nodes in a MP2P network, and one node can only keep certain number of segments in it. Each node can, however, access all the segments from the entire cache space in the network. The problem of how and where to cache the segments in the network becomes a relevant and challenging issue for efficient data sharing. In such a MP2P network, each mobile node can act as both a server and client. When acting as a client, the mobile node wishes to access the required items at a given frequency. When acting as a server, the mobile node carefully chooses multimedia segments/file to cache in its limited memory so as to help minimize the overall access cost. In contrast to the mobile proxy cache problem on base stations, where proxy cache stations are statically linked together via optical fiber/copper wire and have adequate space to cooperatively store huge amount of data, mobile devices in a MP2P network can, however, only temporarily form an adaptive network to cooperatively share a limited quantity of data. Heuristic strategies to select suitable data items to cache at each mobile node are thus needed.

In this article, we present an effective and low cost cache placement and update scheme for MP2P networks, which can be easily implemented in a distributed manner. The rest of the paper organization is as follows. In Section 2 we review the relevant work on data replication placement and cache placement. Section 3 formulates the cache placement problem formally. In Section 4, a number of cache placement algorithms are introduced. Section 5 discusses the relationship between, and necessity of supporting cache placement and cache update in a single scheme. In Section 6, several simulation studies are conducted to evaluate the efficiency of our proposed cache replacement and update scheme. Finally, Section 7 concludes the paper with a few further research directions.

2 Related work

Cache placement in a MP2P network is similar to the well-known facility location problem and k-median problem, and quite a few papers in the literature have tackled these two problems. In this section, we review the existing work on addressing the more general cache placement problem from two different aspects.

2.1 Single data item placement

2.1.1 In general graph

Placement of single data items in a general graph can be described by the facility location problem and the k-median problem; the latter two problems have been widely studied in graph theory. In the facility location problem, the total cost is the sum of the total access cost and the setting up cost of a cache on a node, without the constraint of the number of nodes being selected. In the k-median problem, at most k nodes can be selected as cache nodes. Both problems are NP-hard [12] and many constant-factor approximation algorithms have been proposed for these two problems [1, 4, 5, 8]. A detailed account of many different kinds of topologies such as line, ring network to place cache has been given in [2].

2.1.2 In tree graph

Several papers have focused on devising effective algorithms for the location problem on a tree topology. All these works consider single data item placement on a tree network topology. In such a kind of topology, the problem can be solved in polynomial time. In [14], optimal polynomial algorithms are developed based on dynamic programming for the k-median problem in both undirected and directed trees. [7] devised a cost model involving reads, writes and storage. In [17], a distributed placement algorithm for sensor networks to reduce the total power consumption is developed.

2.1.3 Greedy placement algorithm for single data

The Reverse Greedy algorithm (RGREEDY) for the k-median problem is a typical single data cache placement problem. It works as follows: the algorithm starts by placing facilities on all nodes. At each step, it removes a facility to minimize the total distance to the remaining facilities. It stops when k facilities remain. Our Global Benefit Based Cache Placement is quite the same as such greedy method.

2.2 Multiple data items placement

Baev and Rajaraman [2] have studied the hardness of approximate solution for multiple data placement in arbitrary network. It is shown that the data placement problem with uniform length multiple data object is MAXSNP-hard, and they have come up with a 20.5-approximation algorithm for the cache placement problem. Tang et al. [15] proposed a benefit based cache placement algorithm and gave a distributed algorithm for the cache placement in ad hoc networks. Petrank and Rawitz [13] also described the hardness of multiple cache placement/update. Most of the current works on multiple cache data placement focus on adaptive/distributed algorithm without many rigorous mathematic analyses due to the hardness of the problem.

2.3 Cache update

Meanwhile, cache update/replacement has also been widely studied with multi-data items. Hara [6] gave three adaptive replica allocation methods based on access frequency from mobile hosts to each data item and the status of the network connection. Another cooperative cache algorithm named as Hybrid Cache was proposed for ad hoc networks based on the notions of Cache Data and Cache Path [17]. Cache update for heterogeneous nodes is investigated in [16]. Cache update with energy metric is studied in [9]. However, few works have considered combining cache update and cache placement together. In this article, we advocate to use the two methods as an integral scheme to conduct cache in a MP2P network. There are also related problems including prediction, synchronization and multi-server support that need to be considered [3, 10, 11].

3 Cache placement: problem formulation

3.1 MP2P network scenario

A typical scenario of a MP2P network is shown as Figure 1. In particular, we assume that there are multiple nodes (mobile hosts) in the network *N*, and each mobile host, m_i, has a unique identifier, i.e., $M=\{m_1, m_2, ..., m_{NumClient}\}$, where *NumClient* is the total number of mobile devices in the network environment. Each mobile device is equipped with two wireless network interface cards: one is dedicated to communicate with the center server (e.g. Mobile Support Station), and the other one is devoted to communicate with the neighbor nodes to form the MP2P network.

3.2 Cache placement scenario

The scenario of a special placement on a mobile network is shown in Figure 2. The network is composed by 6 nodes and the total number of cached data placed is 6. Data block D_i stand for

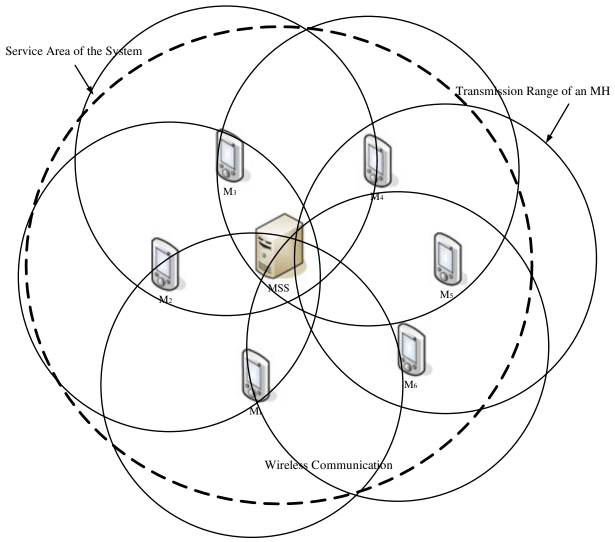


Figure 1 Typical MP2P network environment.

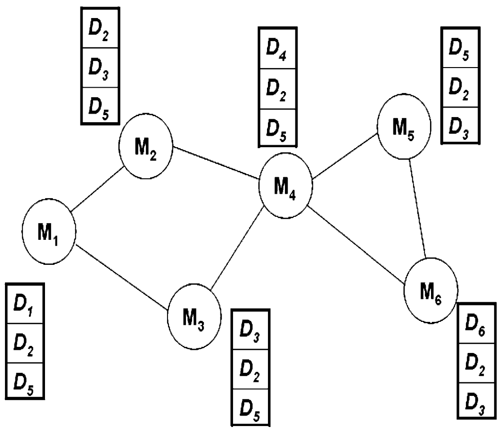


Figure 2 Cache placement scenario.

data blocks which are cached on mobile nodes, and M_j stands for the mobile nodes in the distributed MP2P network. Every node has a maximum cache space is 3 data blocks. In this placement scenario, the total cache space is 3 times of the total requested data, for efficiency purpose, we can see every data block can be found in the cache (from D_1 to D_6)

3.3 Problem formulation

We suppose $\sum$ to be a collection of multimedia segments (assuming each multimedia file is divided into several segments) for all the nodes to cache. For each pair of nodes *i* and *j*, let *cost(i,j)* denote the cost of transmitting a unit-length message between these two nodes. We assume that the cost function defines a metric space (that is, it is nonnegative, symmetric and satisfies the triangle inequality). Each node *i*, which may act as both a client and a server, has capacity *csize(i)* of space available for storing some parts of the multimedia files in $\sum$. The mobile nodes in the network *N* may periodically issue access requests for the multimedia files, the rate of which is given by the demand function *d*. For each node *i* and file segment *f*, *d(i,f)* represents the frequency of node *i* accessing *f* in a special period of time.

A placement P is a function $N \rightarrow 2^{\Sigma}$ that yields, for each node, the set of multimedia segments to be stored in that node. For a placement to be valid, the sum of the lengths of the multimedia segments stored at any node must not exceed the capacity of the node.

For each node *i* and multimedia segment *f*, the demand-weighted cost of accessing *f* equals to *d(i,f)*c(i,j)*length(f)*, where *j* is the node nearest to *i* which has a copy of *f*, *length(f)* is the length of the multimedia segment *f*, and *c(i,j)* is defined as the total time delay between nodes *i* and *j* for transmitting a unit data. Then, the total cost of a placement can be given by the sum, taken over all the nodes and all the data, of *d(i,f)*c(i,j)*length(f)* in a special time range. Then the multimedia data placement problem can be expressed as an integer program problem below:

$$\begin{aligned}
&\min \sum_f \sum_i d(i,f)c(i,j)length(f)y_j^f x_{ij} \\
&s.t. \sum_j x_{ij} \geq 1 \qquad \forall i \\
&\sum_f y_j^f length(f) \leq u_j \\
&x_{ij} \in \{0,1\}, y_j^f \in \{0,1\} \\
&length(f) > 0
\end{aligned} \tag{1}$$

Here x_{ij} indicates if the request of f by client i is mapped to a cache node j; the sum of x_{ij} indicates that at least a cache node j will be able to satisfy the request from node i. The sum of variable y_j^f tests if all the file segments f stored in client j have reached to the limit of j's capacity u_j.

4 Placement algorithms

4.1 Global benefit based cache placement (GBCP)

Motivated by [2], we first present the Global Benefit based Cache Placement (GBCP) as an alternative solution. The GBCP algorithm is given below. The relevant parameters used include: Γ, which denotes the set of nodes that have been selected for cache placement, and τ(G,Γ) representing the total access cost. Then the benefit $\beta(A_{ij}, \Gamma)$ is defined as follows:

$$\beta(A_{ij}, \Gamma) = \tau(G, \Gamma) - \tau(G, \Gamma \cup A_{ij}) \tag{2}$$

where A_{ij} indicates that cache j is placed in node i. The detailed placement process is given as follows:

```
Algorithm GLOBAL-BENEFIT-BASED MULTIDATA-CACHE-PLACEMENT
Input: Network G=(V,E), nonnegative edge costs c, nodes' data demands h: V→∑, where ∑ is the
       collection of multimedia file segments (i.e., for each node i in V, the function h puts U_i data
       files into the cache space of node i, where U_i denotes the capacity of node i).
Output: $\Gamma$
1 $\Gamma \leftarrow \Phi$;
2 while (there are still nodes with empty cache)
3         let $A_{ij}$ be the variable which leads to the maximum benefit $\beta(A_{ij},\Gamma)$, $\Gamma \leftarrow \Gamma \cup A_{ij}$;
4    put the data block j into node i's cache
```

```
Algorithm $\beta(A_{ij},\Gamma)$-CALCULATION
1 for a multimedia segment j put in the cache of node i
2    for each data request, let HN be the total Hop Count to transmit a required multimedia file,
     RN be the total Request Number, and the cost of each data request be: $\sum_{i=1}^{HN} w_i$, where $w_i$ is
     each hop's delay
3        calculate the total access cost as: $\tau(G,\Gamma \cup A_{ij}) \leftarrow \sum_{j=1}^{RN}\sum_{i=1}^{HN} w_i$;
4 calculate benefit $\beta(A_{ij},\Gamma)$ according to formula (2)
```

4.2 Random placement (RAND)

As a baseline case, the random placement (RAND) strategy places the multimedia segments to cache randomly, and only ensures that each node does not contain the same data segments already in it.

4.3 Distributed cache placement scheme

In addition to the Global Benefit base Cache Placement (GBCP) scheme using the global information, we further introduce a Local Benefit based Cache Placement (LBCP) scheme for "benefit calculation" that only uses the placement information of the nearby neighbors in the MP2P network, along with a Clustering Based Cache Placement (CBCP) scheme.

4.3.1 Local benefit based cache placement (LBCP)

LBCP can be viewed as the distributed version for implementing GBCP. Different from GBCP, we calculate the benefit not through global nodes, but only with t hop neighbors' information. The parameter t is calculated based on the neighbor nodes in the network. By flooding a message to t hop neighbors, the (local) benefit of accessing the cached data is calculated by just involving these neighbors, as shown in formula (3) below:

$$conflict = \sum_{i=1}^{num} r^* \frac{1}{hop_i} \tag{3}$$

Then, the benefit can simply be calculated as:

$$benefit = 1/conflict \tag{4}$$

The parameter *num* in formula (3) is the number of the nodes which contain the same data as the current node, and hop_i is the minimum hop count between nodes i and the current node; r is a coefficient, and *1/hop* means that if two nodes have a larger hop count, the conflict of their cached data (i.e., with identical data files) is smaller. The process is described as below:

```
Algorithm    LOCAL-BENEFIT-BASED MULTIDATA-CACHE-PLACEMENT
Input: Network $G≡(V,E)$, nonnegative edge costs c, nodes' data demands $h: V→\sum$, where $\sum$ is the
        collection of  file segments.
Output: a placement $\Gamma$
1 for each node who has empty space for placement
2 while (there are still nodes with empty cache)
3     send message to its neighbor to exchange the data information in other nodes,
      and the nodes should no more than t hop
4 use formula (3) and (4) to place cache data
```

4.3.2 Cluster based cache placement (CBCP)

Another distributed cache placement approach is based on nodes clustering. The clustering strategy is described as follows: each node first declares itself as a Cluster Head (CH). Each node i broadcasts to the list of nodes that it can hear, that is, the set of nodes that are within the communication range of node i. If a node k hears from a node j with a lower ID[1] than itself, node k sends to j a message requesting j to join. If j has already resigned from being a CH itself, j returns a rejection; otherwise j returns a confirmation. When k receives the confirmation and accepts j to be a member, j resigns itself from being a CH in its original cluster. If the cluster with node k as the CH has already reached the maximum size, all future requests for joining this cluster are automatically rejected.

When the above process is completed, the entire network is divided into a number of clusters. Every node belongs to a cluster, and it is either a CH or directly connected to a CH. The next step is for every cluster to broadcast its size to all the neighboring nodes. If a node k receives a message from a cluster C' which has a larger size than the cluster C currently containing node k, then k joins the new cluster C'. (This assumes that the larger cluster C' has not reached its maximum size yet.) Node k then sends notifications to both the new and old clusters to update its new membership status. The notifications first go to the CHs, and are then propagated to the relevant clusters. This process can be repeated as needed, depending on what the maximum cluster diameter is in this case. Each node keeps track of the ID of its CH, the time the node has been a member of its current cluster, as well as the number of nodes in the cluster. A CH also keeps track of the time for each node when the latter has become a member of that cluster.

It is possible for a cluster to grow too large. Consider a situation when a cluster is just below the maximally allowed size, and several nodes want to join simultaneously. Eventually, the CH will be notified of all the new nodes. When the size of the cluster exceeds the maximally allowed size, then one or more nodes need to leave the cluster. Note that a node can leave a cluster either due to such a situation, or because it is (physically) moving away from the cluster (e.g., more than d hops away from its CH). When a node j leaves a cluster, it tries to find another (new) cluster to join. That new cluster must not have reached the maximally allowed size, and the node j cannot be more than d hops away from the CH of the cluster. If several such clusters are found, node j joins the largest one. If no such cluster is found, node j forms a cluster containing j as the single member, with the CH being itself. After all the nodes have been attached to some clusters, the cache placement is then conducted based on the conflict function defined in formula (3).

4.3.3 Approximate ratio analysis

We now conduct some analysis on the approximate ratios of the cache placement schemes introduced above. Assuming the access cost of the best placement is C_o, we have the following lemmas. For the sake of easier analysis, we suppose the delay of each hop is the same.

Lemma 1 If a placement algorithm can ensure every node to get the needed data from no more then d hops in the network, then the cost of this placement algorithm is no more than dC_o

Proof When every node can get the needed data from its local cache, the cost is 0. When the data is in a neighbor's cache, each node should be able to get the data with at least the

[1] Each ID is a random number generated with the current time as the "seed" for each node.

minimum 1 hop. As each data access cost of the placement algorithm is at most d hops, the total cost is thus no more than $dC_{o.}$

Assertion 1 Suppose the maximum hops for each node to obtain all the segments via the placement algorithm GBCP is g, and the cost of GBCP is denoted as C_g,, then we have $C_g \leq gC_0$.

Proof When a segment is not in the local cache, to obtain the segment with the optimal placement incurs at least 1 hop, However, the maximum hops for any node to obtain a cached file with GBCP is at most g hops. Based on Lemma 1, the assertion holds. Actually, the above assertion is rough, as GBCP will make all the segments to be "evenly" placed on each node. As a result, the maximum hop count g will be small.

Assertion 2 If we use m hops neighbors for conflict calculation where m is the maximum number of hops for any of the nodes to obtain any of the required data files, then the access cost of algorithm LBCP is no more than mC_o.

Proof If there are data segments which have not been added to the sub-graph composed by the m hop neighbors, then we claim that each placement generated by LBCP will involve a new segment. Otherwise, suppose a segment/file already existing in cache is added again, the benefit will then be smaller, which conflicts with our placement rule. So the segment/file can not be added. As a result, after LBCP is invoked, every node can get its needed data with no more than m hops, and thus this assertion holds based on Lemma 1.

Assertion 3 If every cluster can contain all the segments, then algorithm CBCP can have a placement with its access cost being no more than kC_o, where k is the maximum diameter which is the maximum distance (denoted as minimum path) of any pair of nodes in the graph corresponding to the largest cluster.

Proof As each cluster generated by the CBCP scheme contains all the segments, and the maximum distance of the largest cluster is k, then every node can get the required data with no more than k hops. When the data segment is not in the local cache, the optimal placement still needs at least 1 hop to obtain the required data; if the data segment is obtained from local cache, it is counted as of 0 hop. Based on Lemma 1, this assertion holds.

5 Cache update

In the set up stage of a MP2P network when caches are first time placed or when the MP2P network is relatively static, the Global Benefit base Cache Placement (GBCP) algorithm can be used for cache placement if a central server exists; otherwise, the Local Benefit based Cache Placement (LBCP) scheme can be used. As the nodes in a MP2P network can be quite dynamic, the initial placement may be non-optimal subsequently, hence cache update/replacement is required. In this section, we present two types of cache update schemes for this purpose.

5.1 Global cache update

The first scheme assumes that there exists a central server. When a multimedia segment/file can not be found from the MP2P network as per a new data request, the server chooses a node p with a globally minimum benefit block for possible cache update. In particular, the new segment/file is placed on p's cache by replacing an old segment/file there. In other words, this scheme uses the global information from the server to update an individual node's cache.

5.2 Local cache update

While the global cache update scheme is simple to devise, it assumes a central server with a high calculation cost. In contrast, the local cache update scheme assumes that every mobile node has a Cache Request List recording the data requests from itself and its neighbors. When a mobile node receives a new data request from its neighbor, it undergoes a possible cache update to keep itself adaptive to the dynamic network conditions, based on formula (5) for calculating the weight of any of its cached blocks:

$$W = \frac{F^f}{S^s * D^d} \tag{5}$$

The variable F is the number of times the cached block (holding a segment/file) is requested (by itself and its neighbors), f, s, d are the ratio parameters for calculation, the parameter S is the size of the cached segment/file, and D is the duplication number of the cached segment/file in the network the node ever knows. For precise exposition, the calculation of the parameters D and F can be illustrated by formula (6) and (7) below:

$$D = \sum_{i=1}^{t} \frac{num_i^n}{hop_i^h} \tag{6}$$

$$F = \sum_{i=1}^{t} \frac{freq_i^f}{hop_i^h} \tag{7}$$

The parameters hop_i and num_i stand for the i_{th} hop and the repetition number of a segment at hop i in the network; The exponents n, f, h are the relative ratios of the corresponding parameters. $freq_i$ stands for the request times of a cache block with all the nodes at hop i. All the variables should be normalized by:

$$\begin{aligned} hop_i &= (hop_i - min\_hop)/(max\_hop - min\_hop), \\ num_i &= (hop_i - min\_num)/(max\_num - min\_num), \\ freq_i &= (freq_i - min\_freq)/(max\_freq - min\_freq). \end{aligned}$$

The segment/file with the minimum W will be kicked out of the cache and replaced by the newly required data item (viz, a multimedia file segment).

5.3 Further analysis: relationship between update and placement

Actually, cache placement is similar to multiple cache update to a great extent. If cache update is achieved globally, it can be regarded as "placement". On the other hand, though the benefit for cache update is calculated dynamically and locally, so as to accommodate dynamic data requests, the operation is conducted when each node in the MP2P network becomes "placed" which can be denoted as "fully cached".

However, they still have some differences: when we “place” a cache block, there still exists enough space for holding the new segment/file, so no replacement will occur. On the other hand, when doing cache update, the predicted small-weight-ones should be kicked out.

6 Simulation experiments

In order to evaluate the efficiency of our proposed cache placement and update schemes, this section conducts comparison study on four cache placement schemes as: Random Cache Placement (RAND), Global Benefit based Cache Placement (GBCP), Localized Benefit based Cache Placement (LBCP) and Cluster Based Cache Placement (CBCP). Table 1 show the relative parameters for the simulation study.

Without loss of generality, in our simulation study, the compared measure is the Average Hop Count (AHC) for every node. AHC is defined as the average number of hops for a required mobile data to be transmitted. The length of each file is based on a uniform distribution. The comparisons are conducted in a simulated MP2P network with 40 nodes on a 600 m*600 m square place. The user requests are generated based on Poisson distribution for each node. The nodes are moving based on a random walk model. The Hop Bound is the neighbors of the maximum hop distance for duplication calculation.

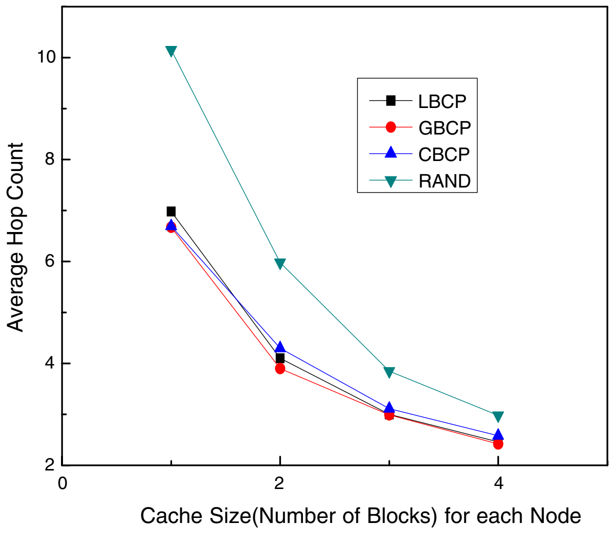
The first comparison in Figure 3 assumes each node have a relatively small cache space (number of unit segments). With the changes of the cache size of each node, we compare the statistic variable AHC to evaluate the system performance. Note that if a mobile host can not get the required multimedia segment from its local cache and its neighbors’ cache, it is assumed that its base station/access point can supply the multimedia data with a maximum hop count.

Figure 3 is the scenario with each mobile node having about 5000 requests generated, there are totally 10 segments to cache, the total node number is 20 and the cache space for each node is 2. From the comparison, we find GBCP, LBCP and CBCP have little difference while GBCP has the minimum AHC. Note that if a node can obtain the requested data segment from its local cache, the hop count is regarded as 0. In the simulation, LBCP uses 3 hop neighbors for the cache benefit calculation. As the total data number is small, all three heuristic placement algorithms (other than the RAND) can

Table 1 The simulation parameters.

Notation	Default value	Definition (range of value)
Cache Size	2	2–10% of the total file obtained
Total File Number	40	20–100(total file can be obtained)
Node No.	40	20–100
Simulation Range	Square place	600 m*600 m matrix bound
Node Signal Transmission Range	150 m	50 m–200 m
Request number for each node	2000	500–5000
Maximum Hop Count(MHC)	10	When cache miss, the MHC value will be used for calculation
Hop Bound for LBCP	4	2–8

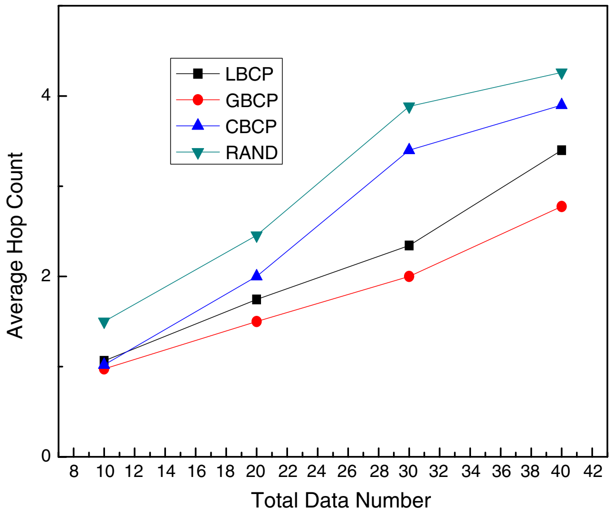
Figure 3 AHC against different cache size.



place the data effectively. In fact, when the data number is small, most of the mobile nodes can obtain their requested multimedia segments from their 1 or 2 hops' neighbors.

Figure 4 investigates another comparison of the four cache placement schemes against different total number of multimedia segments to be accessed. Each node in the network can keep only 2 segments in its cache, this time the total node number is 40. When the total data number is small, all the cache placement schemes have little difference because most mobile nodes can obtain their needed data from nearby neighbors. As the total file number becomes larger, the GBCP algorithm wins out with respect to the hop count. While LBCP performs a bit poorer than GBCP, it does not need any global information of the network, and is very adaptive to the MP2P network conditions. When the number of data segments is not too large, GBCP and LBCP have little difference compared with the CBCP scheme. Although clustering is a quite

Figure 4 AHC against different total data number.



effective method for MP2P network organization and management, it is obvious that CBCP does not perform as well as LBCP and GBCP when the total data number becomes large.

Figure 5 illustrates some detailed difference of the compared cache placement algorithms, the total data number is fixed with 80 data files and the cache space of each node can contain at most 2 data file. As revealed in the figure, when the total nodes number in the experiment increase; the AHC value of each algorithm becomes smaller for the available total cache space become bigger. By using 4 hops neighbor for duplication calculation for LBCP, compared with GBCP, it is only a bit worse than GBCP. We can find when nodes number is small, the CBCP scheme is even wore than random placement (RAND), that is because when nodes number is smaller, the nodes in each cluster is even smaller by dividing the cluster number on average. For CBCP scheme, the nodes should get the cache data only from the special cluster which they belong to, obviously CBCP scheme will cause many cache miss of the most request and the punish cost for cache miss is large. However, when the total node number get larger, each cluster can contain at least 1 copy of each data file, then the efficiency of it will easily exceed RAND.

A comparison of the running time of GBCP and LBCP mainly from the aspect of calculating time the placement benefit is illustrated in Figure 6. Suppose *n, d, c, e* denote the node number, total multimedia segments number, cache blocks number and edge number of the simulation network, respectively. We further assume that one segment has the same size as one cache block and each segment has the same size. The running time complexity of GBCP is calculated as follows: every data placement needs to try, on average, $0.5*n*d$ choices to place cache, so the calculation cost for a placement is $0.5*n*d*O(n^2)$, where $O(n^2)$ is the cost for calculating the minimum path for each pair of nodes by Dijkstra algorithm. The total number of cache blocks to place is $n*c$, so the cost for GBCP is $0.5*n*c*n*d*O(n^2)$. The time complexity for GBCP is thus equal to $O(cdn^4)$. As for LBCP, the total number of cache blocks to place is also $n*c$. For each placement, suppose t $(t<n)$ neighbors have the same data segment in k hops, then the cost is $t*O(s^2)$ for conflict calculation, where $O(s^2)$ is the minimum cost calculated for each pair of nodes within hop k, and s $(s<n)$ is the maximum number of nodes of each sub-graph formed by

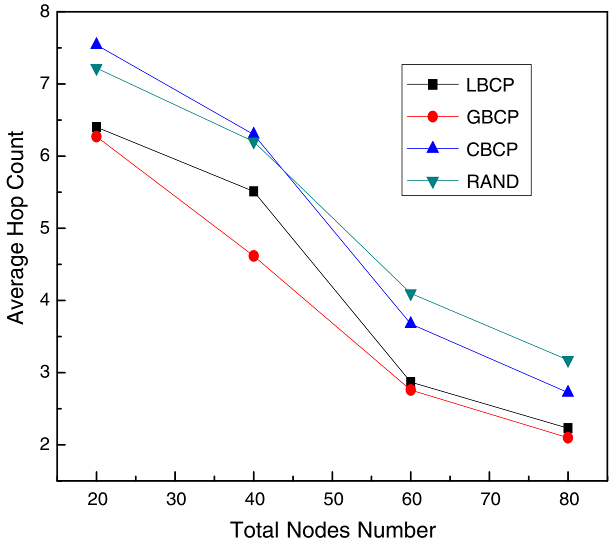


Figure 5 AHC comparison against total nodes number.

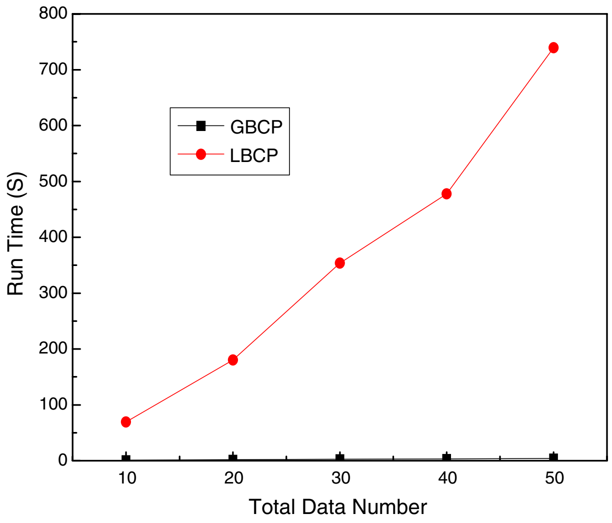


Figure 6 Run time comparison.

the nodes within *h* hops of the current node. Each time *O*(*d/2)* of multimedia segments should be accessed on average to select the segment with the maximum benefit. So the total calculation cost is:

$$n^*c^*d/2^*t^*O\left(s^2\right) = O\left(cdts^2n\right) << O\left(cdn^4\right) \tag{8}$$

We can calculate in advance all the minimum paths between every pair of nodes in the simulation network, but the inequality (8) remains the same. Figure 6 depicts the complexity (running time) comparison between GBCP and LBCP.

For algorithm LBCP, Figure 7 shows a comparison against different hop neighbors with totally 40 files. We compare AHC against the average cache size of each mobile node. Neighbors with different hop counts (marked as 2,4,6,8 hops) are considered for benefit calculation. Notably, when a node's cache space is small, taking further neighbors into the

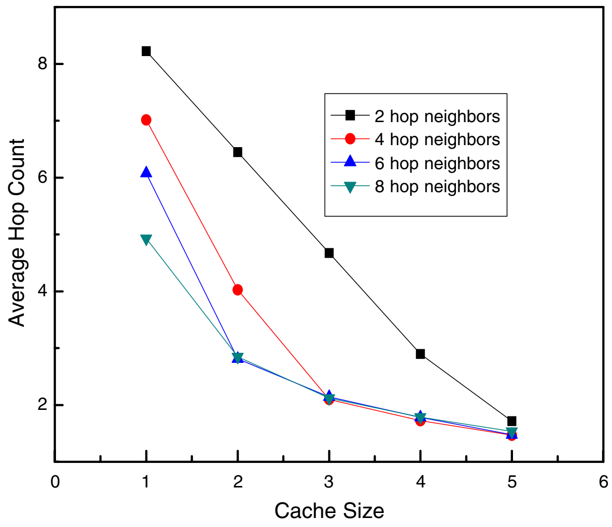


Figure 7 Comparisons against different hop neighbors with LBCP.

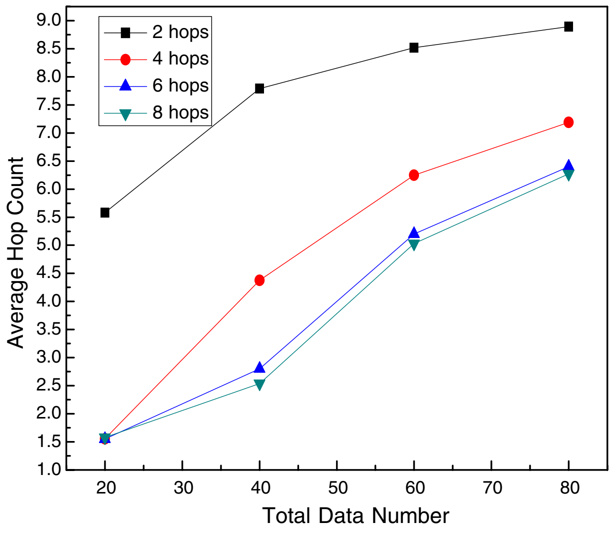


Figure 8 Comparisons against total data number with different hops.

conflict calculation is useful; however, when the average cache size of each mobile node is large, it becomes meaningless to calculate the global benefit for cache placement.

Similar with Figure 7, the experiment in Figure 8 takes comparison against various total data number with different hop neighbors for duplication calculation, in this scenario, totally 20 nodes combined with 20,40,60 and 80 data files are measured. Each node's cache size is at most 2 data file. The simulation illustrate that 4 or 6 hop neighbors used for duplication calculation is best in such scenario as the AHC value improvement is bigger. When the total hops used for duplication calculation become 8 from 6 hops, the improvement is small. However, in a distributed manner, farther neighbors used for duplication calculation means more information exchange and energy consumption.

The final experiment examines the advantage of combining placement and update operation in the network. As shown in Figure 9, with the placement and cache update support the

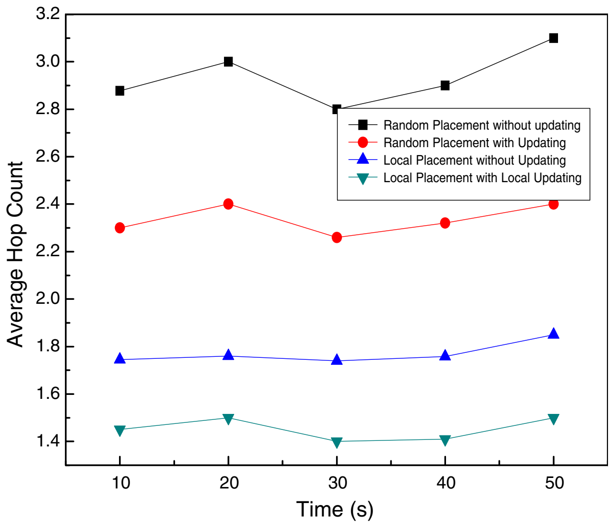


Figure 9 The advantage of placement and update.

network can greatly decrease the AHC value for data access, compared with those supporting none (i.e., no operation) or just one type of these operations. Indeed, an optimized placement and demand-adaptive cache update can make data requests to be satisfied mostly from local or only a few hops' neighbors. In addition, proper update support can handle the problems of dynamic user movement and data popularity change effectively.

7 Conclusion and future work

MP2P networks provide decentralization, self-organization, scalability characters, but suffer from high latency and link break problems. In this article, we have proposed several cache placement and update schemes for MP2P networks. Various simulation experiments show that: (1): the Global Benefit based Cache Placement (GBCP) has the best placement cost but is hard to be implemented in a distributed manner; (2): the Cluster Based Cache Placement (CBCP) is a useful method for MP2P network management, yet the efficiency is not as good; (3): both CBCP and the LBCP schemes are easier for adoption in a real MP2P network, and they have nearly the same cost as the GBCP scheme when the local cache size of each node is not too small (hence the chance of having segment duplication is relatively high); (4): LBCP can get much the same cost as GBCP if the conflict is calculated with enough neighbors and when the level of data redundancy is not too small; (5): the support and provision of both placement and cache update can greatly decrease the AHC for data access, thereby improving the overall efficiency of the entire MP2P network.

As future work, energy consumption as a new parameter will be considered, and more subtle statistic values (e.g., the total bytes of messages sent and received) should be calculated for cache placement and update. Furthermore, as mobile devices are very often different in their inner characters such as their bandwidth, cache size, process ability and even security level, mobile caching and update with heterogeneous nodes in heterogeneous wireless networks (with possibly different service providers) will be a challenging and practically important direction for our future work.

Acknowledgements The research described in this paper has been supported primarily by a grant from City University of Hong Kong (Project No.7008043), and partially by a grant from the Research Grants Council of the Hong Kong SAR, China (Project No. CityU 117608). It has also been supported by the Key Program of Natural Science Foundation of China (No.60933013) and the National Natural Science Foundation of China (Grant No.60775037, 61073110). The research has been benefited from various discussions among the group members of the Joint Research Lab between CityU (Hong Kong) and USTC (China) in their advanced research institute in Suzhou, China.

References

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. Society for Industrial and Applied Mathematics. (2004)
2. Baev, I., Rajaraman, R.: Approximation algorithms for data placement in arbitrary networks, Proc. 12th ACM-SIAM Symposium on Discrete Algorithms, pp. 661–670, USA, (2001)
3. Chan, A., Lau, R., Ng, B.: A hybrid motion prediction method for caching and prefetching in distributed virtual environments. ACM VRST 2001, pp. 135–142 (2001)
4. Charikar, M., Guha, S.: Improved combinatorial algorithms for the facility location and k-median problems, Proc. 40th IEEE Symposium on Foundations of Computer Science. 378–388 (1999)
5. Gupta, A., Kumar, A., Roughgarden, T.: Simpler and better approximation algorithms for network design. STOC'03, June 9–11, 2003, San Diego, California, USA

6. Hara, T.: Effective replica allocation in mobile P2P networks for improving data accessibility. IEEE INFOCOM (2001)
7. Kalpakis, K., Dasgupta, K., Wolfson, O.: Sterner-optimal data replication in tree networks with storage costs. Proc. IDEAS, (2001)
8. Krishnan, P., Raz, D., Shavitt, Y.: The cache location problem. IEEE/ACM Trans Networking **8**(5), 568–582 (2000)
9. Li, W., Chan, E., Chen, D.: Energy-effective cache replacement policies for cooperative caching in mobile Ad Hoc network. Wireless Communications and Networking Conference (WCNC'07). (2007)
10. Li, F., Lau, R., Ng, F.: Vsculpt: A distributed virtual sculpting environment for collaborative design. IEEE Trans Multimedia **5**(4), 570–580 (2003)
11. Ng, B., Si, A., Lau, R., Li, F.: A multi-server architecture for distributed virtual walkthrough, ACM VRST 2002. pp. 163–170, (2002)
12. Nuggehalli, P., Srinivasan, V., Chiasserini, C., Rao, R.R.: Efficient cache placement in multi-hop wireless networks. IEEE Transactions on Networking. **14**(5): (2006)
13. Petrank, E., Rawitz, D.: The hardness of cache conscious data placement. Nord. J. Comput
14. Tamir, A.: An O(pn2) algorithms for the p-median and related problems on tree graphs. Oper. Res. Lett. **19**, (1996)
15. Tang, B., Gupta, H., Das, S.R.: Benefit-based data caching in mobile P2P networks. IEEE Trans Mob Comput **7**(3), 289–304 (2008)
16. Ye, F., Li, Q., Chen, E.: Adaptive caching with heterogeneous devices in mobile peer to peer network. ACM SAC'08
17. Yin, L., Cao, G.: Support cooperative caching in mobile P2P networks. IEEE Trans Mob Comput **5**(1), 77–89 (2006)