An Evolution-Based Cache Scheme for Scalable Mobile Data Access

Fan Ye[1, 2, 3], Qing Li[2, 3], and Enhong Chen[1, 2]
[1] Department of Computer Science and Technology,
University of Science & Technology of China, Hefei, China.
[2] Joint Research Lab of Excellence,
CityU-USTC Advanced Research Institute, Suzhou, China.
[3] Department of Computer Science,
City University of Hong Kong, Hong Kong, China.
yfan@mail.ustc.edu.cn, itqli@cityu.edu.hk, cheneh@ustc.edu.cn

ABSTRACT
Streaming media data access has been a problem for several years, and the problem becomes tougher in the mobile environment in which mobile users use mobile devices that are of rather limited storage space, preventing the clients from having a large cache. In this paper, we design a novel evolutionary caching algorithm for base stations to adapt to the user requests, so as to make the scheme more adaptive to the changing environment while maintaining good Byte Hit Ratio (BHR) or Number Hit Ratio (NHR) for the real world requests. We evaluate the effectiveness of our evolutionary caching algorithm through simulation studies, the results of which demonstrate that our scheme can obtain good performance on buffering streaming media data for user requests as far as the BHR and NHR metrics are concerned.

Categories and Subject Descriptors
Scalable Mobile Systems

General Terms
Algorithms, Design

Keywords
Stream Media Caching, Hybrid Scalable Caching Scheme, Evolutionary Algorithm

1. INTRODUCTION
With the continued growth of the Internet and WWW in the last decade, many interesting applications including multimedia streaming are flourished. Streaming media (e.g., music or video) data access has been a research problem over the past few years.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Infoscale 2007, June 6–8, 2007, Suzhou, Jiangsu, China.
Copyright 2007 ACM 978-1-59593-757-5…$5.00.

Much work has been done on consolidating the WWW with the wireless networks [1], [2]. Such an integration is sometimes also referred to as W4 − World Wide Web for Wireless.

Among the numerous studies for enhancing the mobile Internet performance, caching popular media data in base stations to facilitate the usage of the mobile users is emerging as one of the popular approaches. Indeed, setting up caching mechanisms in mobile base stations can reduce the connection time between the mobile hosts and the base stations, and ease the network traffic between base stations and the back-end media servers [2].

However, it is a challenging task to design a scalable and adaptive caching scheme suitable for the base stations. While some previous work has concentrated on such aspects as distributed, cooperative [2] or proxy [5] cache mechanisms, our focus in this paper is on devising a scalable caching scheme, In particular, we advocate an evolution approach which has been widely used in many fields and obtained many good results. In designing our evolution algorithm, we consider a number of factors which are related to caching optimization. Through simulation studies, we demonstrate that the scalable caching scheme using our evolution algorithm can get much better result than existing popular schemes such as least frequently used (LFU) and least recently used(LRU) in the same environment.

The rest of this paper is organized as follows. In section 2 we review some existing research work closely related to our research. We provide our modeling framework in section 3. In section 4, we present an evolution-based caching scheme, and describe a number of algorithms employed by the scheme. A simulation study is conducted in section 5, and the extensibility issues discussed as well. Section 6 concludes the paper and offers a number of further research issues.

2. RELATED WORK
Research on data caching has been conducted for many years. Most of the works have concentrated on the cooperative and distributed environments. More recently, multimedia data caching is getting more attentions. A caching and streaming framework for multimedia has been proposed in [1], where the authors have considered such factors as the frequency, recentness, and the size of the media objects in order to decide which objects should be replaced from the cache. In their study, the authors found that the cooperative method can get a better Byte Hit Ratio (BHR) or

Number Hit Ratio (NHR) than other existing schemes. In [2], an SAA* based search and optimization approach for caching the media data has been proposed, in which the cache optimization is turned into a binary-tree search problem. In [3], segmentation based Multimedia Streams Caching Scheme has been described. However, little work has concentrated on the scalability issue of the caching scheme. In fact, a lot of parameters should be considered if a caching scheme is to become practically useful, and many of the parameters can only be obtained through trial-and-error. Also, some traditional caching mechanisms and algorithms may be too simple to be effective in addressing the inner complexity of mobile multimedia data. In this paper, based on an evolution algorithm we devise a scalable caching mechanism for the base stations, so as to obtain better performance for user requests as far as the BHR and NHR metrics are concerned.

3. MODELING FRAMEWORK

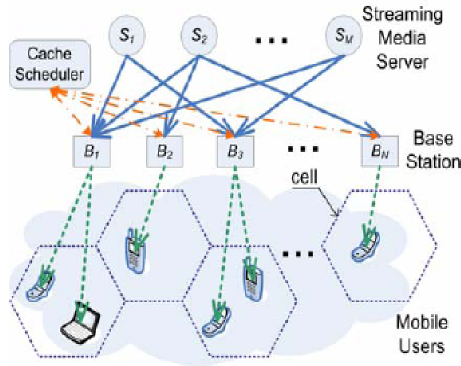

Figure 1. Mobile Environment

As in [2], our system exhibits a three-tier hierarchy model which is depicted in Figure 1. Streaming media servers (denoted as S_t) provide multimedia services over the Internet, which may handle a lot of requests simultaneously. Mobile devices (denoted as M_i) connect to the Internet through the *N* base stations, numbered as B_j *(j= 1, 2… N)*. A mobile device joins the network to get services via a base station B_i. A base station can give services in two ways: if the base station[1] has cached the media object requested by a mobile client, then it let the mobile client access the requested media object directly; else the base station has to broadcast the request to the neighboring base stations which may have cached the required object or, in case none of the neighbors has it, send the request to a server S_t to ask for the media object before sending the requested media object to the mobile client. Clearly, the second case is much more time consuming as more traffic on the (mobile) network occurs.

In our model, each base station maintains two types of data, namely, “cached request list” and “unsatisfied request list”. “cached request list” denoted as C_{i1} ,C_{i2} ,…, C_{iK_i} , where each C_{ip} (1≤p≤Ki) has five field for BHR or NHR calculation: Cid (the id of the media object), ReqNum (the number of times the mobile clients have accessed the block), Count (timer: each new request will cause the Count of all cached request to be incremented by one; if the cache block has been accessed, Count is reset to zero), Size (the size of the media), and Data (real data of the media). As the second data structure is the “unsatisfied request list” denoted by U_{i1}, U_{i2}… U_{iL_i} , where each U_{iq} (1≤q≤L_i) has two fields, namely, Uid(the id of the media object which can not be found from the base station B_i), and UReqNum (the number of times of the unsatisfied requests).

We employ two widely used metrics: Number Hit Ratio (NHR) and Byte Hit Ratio (BHR), to evaluate the performance of our mechanism. NHR is defined by the ratio of total media from cached objects over the total number of objects requested by all the clients, and BHR is defined by the ratio of total bytes from cache over the total bytes of user request. Suppose there are *R* mobile clients numbered as M_1, M_2 ,… ,M_R. Formula (1) shows how to calculate NHR within a time window T=[t_1 , t_2], and Formula (2) shows how to calculate BHR within the time window:

In this paper, we use the two ratios as the most important criteria to design and to evaluate the performance of our algorithm under different conditions and situations.

$$NHR=\frac{\sum_{i=1}^{N}\sum_{p=1}^{K_i} C_{ip}.ReqNum}{\sum_{j=1}^{R} M_j.ReqNum} \quad \ldots\ldots(1)$$

$$BHR=\frac{\sum_{i=1}^{N}\sum_{p=1}^{K_i} C_{ip}.ReqNum * C_{ip}.Size}{\sum_{j=1}^{R}\sum_{i=1}^{K_j} M_j.Req_i.ReqNum * M_j.Req_i.Size} \quad (2)$$

4. AN EVOLUTION-BASED CACHING SCHEME

In this section, we describe a scalable caching scheme based on an evolution approach. First, some preliminary introduction is given, followed by a detailed description of our evolution algorithm which serves as the core of our scalable cache mechanism.

4.1 Preliminaries

In our approach, an evolution algorithm is used to make our caching scheme scalable. Table 1 lists most of the parameters used by our scheme.

In Table 1, the parameter f can be regarded as the power of C_i.ReqNum, r is the power of C_i.Count and s is the power of C_i.Size; f, r and s are all real numbers. In addition, W is the weight of the cache block, whereas MinW is the minimum weight of the cache block. ReqNum is the total number of the requests to the cache block, whereas Count is the number of the client requests to the block: each new request to the block will cause its Count to be incremented by one; if the cache block has been

[1] Strictly speaking, each service area should have a proxy responsible for providing the caching and book keeping services, and each proxy may correspond to several base stations within the same service area.

accessed, Count is reset to zero. Using this scheme, we can update the cache block with a larger Count, since a cache block may have been not accessed for a long time.

Table 1. The parameters of our scheme

Notation	Definition (Default values)
Zipf Factor	0.47
Total number	500 media objects
Cache space	10% -30% of the total media
Mean interval	12.5s
GENERATION	the era of evolution (10)
POP_SIZE	the number of individual (40)
GENELENGTH	3-4
SIMULATION TIME	50000s(about 4000 request)
Media Size	Uniform distribution in 1Mb – 10Mb
ReqNum	Request times to a cache block
Size	The size of the media block
w	The weight of the cache block
minW	The minimum weight of the cache block
Count	A timer for the LRU algorithm
cid	Id of the media object to be updated
f	The weight of ReqNum
s	The weight of Size
r	The weight of Count
T	Parameter used for EBHA-PT.
β, χ	Parameter used for EBHA-P
flag	Decide the update strategy to use

For our scheme, the following three algorithms are devised and experimentally compared.

1. An *Evolution-Based Hybrid Algorithm with probability=1* (EBHA-1): In this scheme, the block which has the minimum weight MinW should be updated with a probability equal to 1; other media blocks with a larger W will not be changed.
2. An *Evolution-Based Hybrid Algorithm with probability p dependent on T* (EBHA-PT): Here the *T* can be regarded as the temperature parameter which is denoted as:

$$p(j) = \frac{\exp(T / w_j)}{\sum_{j}^{CacheSize} \exp(T / w_j)}$$

If the temperature is large, a larger W means a small probability for the cache block to be replaced. If *T* is small, the probability for a media block of a larger W will have a slightly higher probability to be replaced than the previous case. This scheme is derived from a simulated annealing algorithm, using which we can get a global optimization of the Number Hit Ratio and/or Byte Hit Ratio.

3. The third one is called *Evolution-Based Hybrid Algorithm* with probability=P (EBHA-P), in which two parameters β and are χ used as determinant constants to help determine if an update operation to the cache list should be done or not. In fact, we use them to compare with the parameter minW which is calculated as follows:

$$\min W = \min\left(\frac{(C_j.ReqNum)^f}{(C_j.Count)^r * (C_j.Size)^s}\right) \quad (1 \le j \le N)$$

where min() gets the minimum of the function with $1 \le j \le N$. EBHA-P works as follows: if $\min W < \beta / \chi$, then we replace the cache; else we do not replace the cache, but satisfy the client by sending the request to a server. This is because if the minW is large (i.e., $\ge \beta / \chi$), then the cache block may still have a high probability to be accessed by the clients in the near future, so we should not replace it.

4.2 Algorithm Descriptions

When a mobile client M_i enters the service area of a base station B_j and issues a request, the data structures "Cached request list" and "Unsatisfied request list" are updated, and if needed, the cache list must be updated by using a cache replacement algorithm. Figures 2 - 7 give our main caching scheme, in which Figure 1 illustrates the algorithm for checking if M_i's request should be satisfied by allocating a cache block and/or replacing an existing block, Figures 3 - 5 describe the aforementioned three evolution algorithms respectively, and Figures 6-7 depict the overall caching scheme based on evolution.

```
Algorithm_Request_Satisfy (individual ind, int flag)
1. for every block Cp in the cache list do
2.        Cp.Count++;
3. end for
4.  if ( Bj.Cp = = Mi.Req)
5.        then Return Bj.Cp to the mobile client
6.        and Bj.Cp.ReqNum++, Bj.Cp.Count=0;
7.  else if(MAX_CACHE_SIZE - TOTAL_IN_CACHE) >
          sum_{p=1}^{Kj} Bj.Cp.Length
8.        then create a cache block for Mi.
9.  else
10.     update cache List using by updating
        algorithm(EBHA-1, EBHA-P or EBHA-PT)
```

Figure 2. Deciding the satisfiability of a client request

For our main caching scheme (cf. Figure 6), we use an evolutionary algorithm to get the parameters f, s, r, and *T*. We put these parameters into a list called Individual, and by putting the Individual list as the parameter to our main caching update algorithm, we use a random function to first initialize f, s, and r before running the remaining algorithm. In the evolution algorithm, Crossover() is the function to get a new individual from two parents and it has no difference with the common real

code crossover operation. Besides, `Mutation()` is the function to get a new individual by changing f, s, r, and T to new real numbers within the range limit of the parameters. We then run the algorithm by using the individuals (parameters) from the Individual list for many times, and calculate the average Number Hit Ratio (NHR) as the fitness of the individuals. After we get the best individual, we put the individual as the parameter to one of the hybrid caching replacement algorithms and cache the appropriate media block.

Evolution_Based_Hybrid_Algorithm_1 (M_i, B_j, $Media_k$)

1. $\min W = \frac{(C_0.ReqNum)^f}{(C_0.Count)^r * (C_0.Size)^s}$
2. cid = 0;
3. for every media j in B_j' s cache List do
4. $if(\frac{(C_{j+1}.ReqNum)^f}{(C_{j+1}.Count)^r * (C_{j+1}.Size)^s} < \frac{(C_j.ReqNum)^f}{(C_j.Count)^r (C_j.Size)^s})$
5. then $\min W = \frac{(C_{j+1}.ReqNum)^f}{(C_{j+1}.Count)^r * (C_{j+1}.Size)^s}$
6. cid = j+1;
7. end for
8. update the id of cid cache block with $Media_k$;

Figure 3. The algorithm of EBHA-1

Evolution_Based_Hybrid_Algorithm_PT (M_i, B_j, $Media_k$)

1. for every media j in M_i' s cache list, do
2. $W_j = \frac{(C_j.ReqNum)^f}{(C_j.Count)^r * (C_j.Size)^s}$;
3. choose (cid=j) with the probability $p(j) = \frac{\exp(T / w_j)}{\sum_j^{CacheSize} \exp(T / w_j)}$
4. end for
5. update the cid cache block with $Media_k$;

Figure 4. The algorithm of EBHA-PT

As listed in Table 1, the parameter GENERATION is the total era of our evolution-based algorithm, POPSIZE is the total number of individual of the evolutionary algorithm, and the GENELENGTH is the number of the real number in the individual. To generate an individual, we put the needed parameters such as f, s, r, T, β, χ in the object list; each object in the objects list can be regarded as an individual.

In the scheme described in Figure 6, the evolution process is very much like a genetic algorithm. In particular, it gets POP_SIZE individuals and returns the best one which contains the best parameters to be used by one of the evolutionary cache replacement algorithms.

Evolution_Based_Hybrid_Algorithm_P (M_i, B_j, $Media_k$)

1. $\min W = \frac{(C_0.ReqNum)^f}{(C_0.Count)^r}$
2. cid = 0;
3. for every media j in M_i' s cache list
4. if $(\frac{(C_{j+1}.ReqNum)^f}{(C_{j+1}.Count)^r (C_{j+1}.Size)^s} < \frac{(C_j.ReqNum)^f}{(C_j.Count)^r (C_j.Size)^s})$
5. $\min W = \frac{(C_{j+1}.ReqNum)^f}{(C_{j+1}.Count)^r}$
6. cid = j+1;
7. end for
8. If $\min W < \beta / \chi$
9. update the cid cache block with $Media_k$;

Figure 5. The algorithm of EBHA-P

The function evaluate($individual_i$,flag) uses the parameters in $individual_i$ and decides which evolutionary cache replacement algorithms to apply based on the value of "flag". The result "value" returned by evaluate is the BHR or NHR value obtained after the chosen cache replacement algorithm is conducted upon the user request. If the value is big (>bestValue), then we regard the $individual_i$ as a desirable candidate with good parameters (f, s, r, T, β, χ) to cater for the specific problem.

Algorithm_Evolution (Individual list)

1. Randomly generate POP_SIZE individuals;
2. While (Gen<GENERATION)
3. for every $individual_i$ in the Individual list
4. value = Evaluate ($individual_i$, flag);
5. $individual_i$.fitness = value
6. if (value>bestValue)
7. then bestValue = value;
8. $individual_{best}$ = $individual_i$;
9. end for
10. for every individual in the Individual list
11. Crossover();
12. Mutation();
13. end for
14. Gen++;
15. end While
16. Return $individual_{best}$;

Figure 6. The main evolution-based caching scheme

```
Subroutine Evaluate(individual_i, flag)
1. for every request in request list
2.    Algorithm_Request_Satisfy(individual_i, flag);
3. end for
4. return value;
```

Figure 7. The Evaluate subroutine

5. EMPIRICAL STUDY

In order to evaluate the performance of our scheme vis-à-vis the other ones, we compare it with the Least Recently Used (LRU) and Least Frequently Used (LFU) algorithms which are widely used in mobile data caching. Also, we give a comparison with the FSR scheme proposed in [1], in which the weight is defined as:

$$w = F^f S^s R^r$$

where F is the number of times the block is accessed (i.e., the frequency), S is the size of the block (size) and R is the times since the last access for the block (i.e., the recentness). The three exponents f, r and s are chosen by trial-and-error, and in our experiment we adopt f=2, r=0, and s=-1.5 as suggested by the authors. A comparison study among all these schemes against our three hybrid cache algorithms is conducted with respect to the BHR and NHR ratios.

5.1 Simulation Model

In our simulation, each media object is randomly assigned with a value representing its popularity, and mobile clients choose media objects randomly based on such values. The larger the value is, the more possible that mobile clients may access that object. Formula (3) defines the relationship between the ranking and the popularity of a media object, where α is assumed to be around 0.5(or more precisely, 0.47).

$$popular_i = \frac{1}{rank^{\alpha}} (1 \le rank \le mediaNumber) ....(3)$$

Every mobile client is independent to each other. If the cache of a base station B_j can not satisfy a request of its mobile client in B_j's service area, B_j will get the requested media object from a media server which is the nearest to it. For the program to generate a reliable result, we keep the cache size to be much smaller than that of the actual requests in our experiment.

5.2 Simulation Result

In the first study, we compare the performance of our Evolution-Based Hybrid Algorithms with LFU, LRU and FSR from the perspectives of NHR, our media size obey the uniform distribution (it is also the default distribution of our simulation). Totally 500 media objects are involved, with the cache space being about 10% to 30% of the total media data space, and an average arrival time of new request being 12.5 seconds (cf. Table 1). Figure 8 illustrates the result of our study, in which it is shown that our three EBHA algorithms have a better Number Hit Ratio than that of LFU (Least Frequently Used), LRU (Least Recently Used) and FSR. In fact, our three evolutionary hybrid cache algorithms perform very similarly to each other, whereas the FSR performance is better than LRU and LFU but poorer than ours.

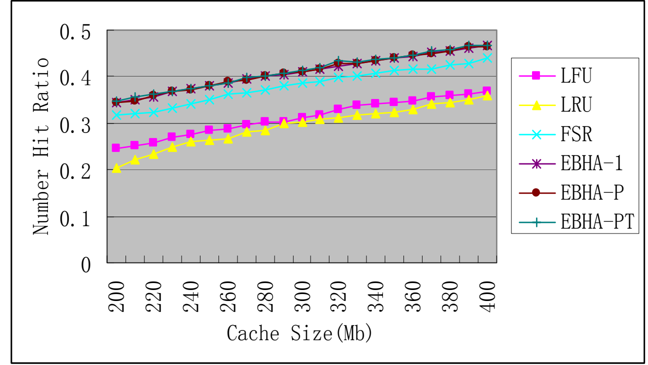

Figure 8. Comparison of NHR against Cache Size

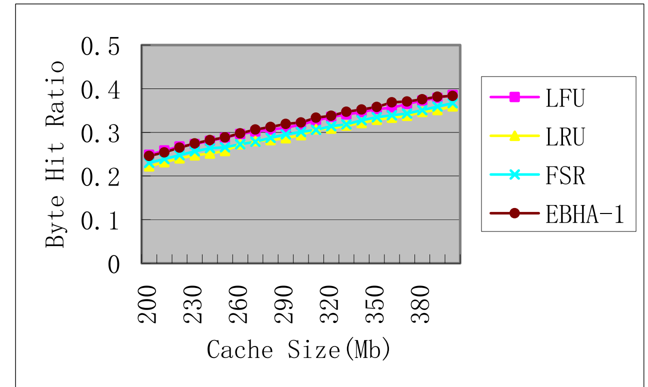

Figure 9. Comparison of BHR against Cache Size

Our second study compares, from the perspectives of BHR against cache size, the performance of our Evolution-Based Hybrid Algorithms with LFU, LRU, and FSR. As shown in Figure 9, the EBHA-1 is of only a slightly better BHR in comparison with LRU and FSR (EBHA-P and EBHA-PT get quite similar results with EBHA-1 here). For the size of the media

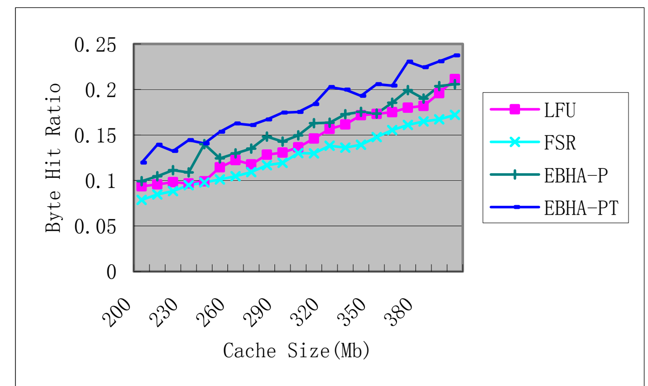

Figure 10. Comparison of BHR against Cache Size

objects uniformly distributed within 1Mb - 10Mb, the size distribution has no big impact on caching, so our scheme's performance has little difference with that of LFU in this case.

Next, we compare our three algorithms with LFU, LRU and FSR based on NHR against request number, with the cache size being

300Mb and media data size obeying the uniform distribution. As shown in Figure 11, we can see that EBHA-1, EBHA-P, and EBHA-PT perform similarly, having the best NHR than other three algorithms; in addition, FSR outperforms LFU and LRU, with LRU being the worst.

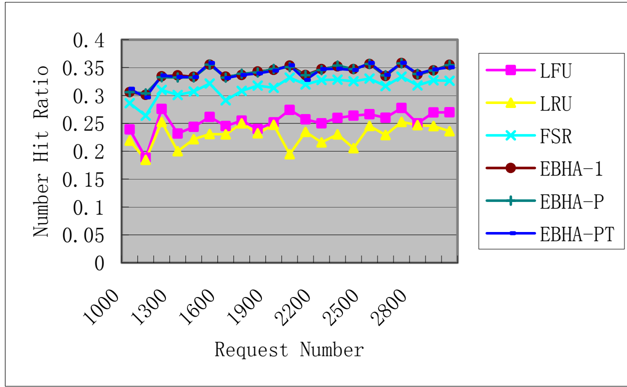

Figure 11. Comparison of NHR against Request Number

From the above simulation studies, we see that our caching scheme can always get the best result in all the cases, whereas LRU does not perform well in most cases and FSR or LFU may get a good result in some case but not in all the cases. Also, from Figure 11, we can see that the user request pattern have a big impact on the performance of the cache scheme. As the fluctuations of the curves appear to be similar, we can also conjecture that the cache schemes share some inner similarity.

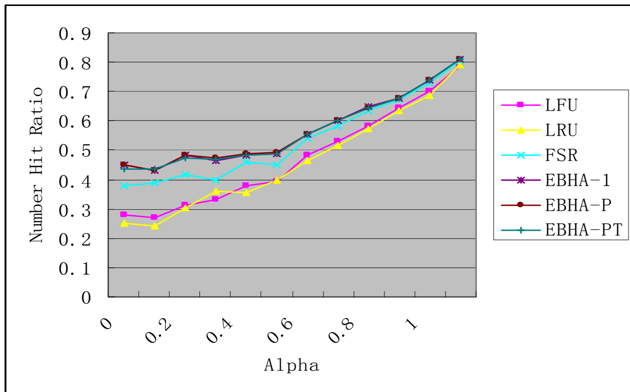

Figure 12. Comparison of NHR against ALPHA (α)

In Figure 12, we compare the algorithms of LFU, LRU, FSR and EBHA against α with uniform distribution of media size; here we keep the cache size as 300Mb. We can see that α of formula (3) also has an impact on the performance: when α is small (i.e. in reasonable value range), the algorithms exhibit quite different results, whereas when α is large, the algorithms perform closer to each other. The reason is that if some media object has a too high popularity, the algorithms based on popularity can always keep it in the cache, so all algorithms such as LFU, LRU, FSR and our EBHA get a good result. However, in most real cases α is of a medium value (e.g., around 0.5), in which case our EBHA based algorithms always have the best NHR values, reflecting therefore the desired scalability and adapting.

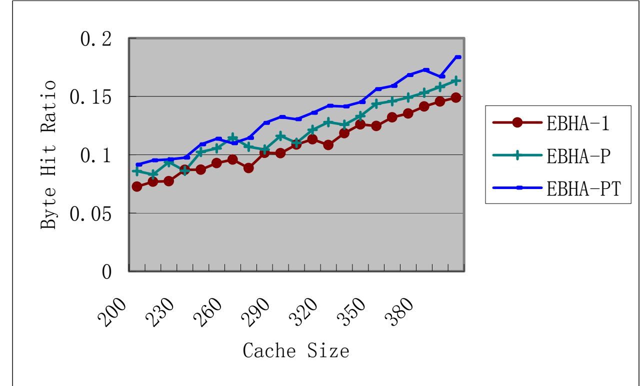

Figure 13. Comparison of BHR against Cache Size

As our last study, we compare the difference among our three algorithms so as to illustrate their different scalability in terms of cache size. In particular, we compare the three algorithms under the situation that the user requests are not very regular. As in the real-world situations, some media objects may suddenly become popular whereas some others quickly become unpopular during a particular period of time. As in the previous studies, we assume that user requests for the media objects follow the Zipf distribution. From Figure 13, we can see that our three algorithms perform quite differently against the Byte Hit Ratio (BHR): EBHA-PT has the best result; EBHA-P takes the second place, whereas EBHA-1 is the worst among the three.

5.3 Extensibility of our work

Though our work has been primarily conducted within the mobile environment, our approach of devising a scalable cache scheme can be applied, with simple adaptation, to different environments including for example the traditional multimedia databases and document caching. As media objects may have different sizes and popularity, the scalability of the caching scheme is very important in coping with the changing patterns of different popularity and different cache conditions. Our proposed caching scheme can be adapted to various dynamic environments in which scalability is the must.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an evolution-based hybrid cache scheme for mobile media data access. The evolutionary aspect of our scheme can be perceived as a process of learning the pattern of the user requests, the interest of the mobile clients, and the characteristics of the cache environment. As user interests may keep changing, the popularity of the media data may change with time and location. The problem is further complicated when the media objects may have different sizes and request patterns. All of these factors make it unpractical to design the cache scheme based on a fixed approach by simply adopting the traditional algorithms such as LFU or LRU. Based on the learning capability, our evolution-based hybrid cache scheme is able to adjust to different conditions and different criteria. Simulation studies suggest that our cache scheme is also scalable in that it can be enlarged to a large number of clients and base stations, and applied to predict the user request patterns.

Our future work will investigate the impact of such different conditions as “cooperative” vs. “selfish” caching among of the mobile clients and base stations. In combination with our evolution-based hybrid algorithm, we also plan to incorporate other machine learning techniques into the cache mechanism, so as to be able to predict the user request pattern and environmental characteristics, thereby optimizing the cache performance and providing better quality of service.

7. ACKNOWLEDGEMENTS

The work described here is supported by the National Basic Research Fund of China (“973” Program) under Grant No.2003CB317006 and Program for New Century Excellent Talents in University under Grant No.60573077. 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.

8. REFERENCES

[1] S. Paknikar, M. Kankanhalli, K.R. Ramakrishnan, S.H.Srinivasan, L.H. Ngoh, “A Caching and Streaming Framework for Multimedia”, *ACM Multimedia 2000*, Los Angeles, USA.

[2] J. Zhai, X.Li, and Q. Li, ”Statistical Buffering for Streaming Media Data Access in a Mobile Environment”, *ACM SAC’06*.

[3] K. Wu, P. S. Yu, J. L. Wolf, “Segmentation of Multimedia Streams for Porxy Caching”, IEEE TRANSACTIONS ON MULTIMEDIA. VOL. 6, NO. 5, OCTOBER 2004.

[4] Daniel Barbara, ”Mobile Computing and Database – A survey”, IEEE TKDE 11(1), 1999.

[5] B. Wang, S. Sen, M. Adler, and D. Towsley, “Optimal Proxy Cache Allocation for Efficient Streaming Media Distribution”, *Proceedings of INFOCOM conference*, 2002.

[6] D.L. Lee, J. Xu, B. Zheng and W.-C. Lee, “Data Manage in Location-Dependent Information Services”, IEEE Pervasive Computing 1(3), 2002.

[7] X. Tang, F. Zhang, and S.T. Chanson, “Streaming Media Caching Algorithms for Transcoding Proxies”, *Proceedings of the international Conference on Parallel Processing (ICPP’02)*.

[8] M. Chesire, A. Wolman, G. M. Voelkert, and H.M. Levy, “Measurement and Analysis of a Streaming-Media Workload”, *The USENIX Symp. Internet Technologies and Systems*, Boston, Massachusetts, 2001.

[9] Y. Huang, P. Sistla, and O Wolfson, “Data Replication for Mobile Computer”, *ACM-SIGMOD’94*, Minneapolis, Minnesota, May 1994.

[10] S. Jin, A. Bestavros, and A. Iyengar, “Accelerating Internet Streaming Media Delivery using Network-Aware Partial Caching”, *IEEE ICDCS’02*.

[11] X. Li and Q. Li,”User Pattern Analysis in Cellular System”, *Proc. the 7th International Conference on Mobile Data Management (MDM’06)*.

[12] U. Bodenhofer, “Genetic Algorithms: Theory and Applications”, Lecture Notes (3rd edition), Johannes Kepler University Linz, Winter 2003/2004.

[13] X. Zhu, Y. Huang and J. Doyle, “Genetic algorithms and simulated annealing for robustness analysis”, *Proceedings of American Control Conference*, 1997.

[14] L. Davis, “Job shop Scheduling with Genetic Algorithms”, *Proceedings of the 1st International Conference on Genetic Algorithms*, pp 136-140, Hillsdale, NJ, Lawrence Erlbaum Associates, 1985.