An Efficient Cache Replacement Strategy for the Hybrid Cache Consistency Approach

Aline Zeitunlian and Ramzi A. Haraty

***Abstract*—**Caching was suggested as a solution for reducing bandwidth utilization and minimizing query latency in mobile environments. Over the years, different caching approaches have been proposed, some relying on the server to broadcast reports periodically informing of the updated data while others allowed the clients to request for the data whenever needed. Until recently a hybrid cache consistency scheme Scalable Asynchronous Cache Consistency Scheme SACCS was proposed, which combined the two different approaches benefits' and is proved to be more efficient and scalable. Nevertheless, caching has its limitations too, due to the limited cache size and the limited bandwidth, which makes the implementation of cache replacement strategy an important aspect for improving the cache consistency algorithms. In this thesis, we proposed a new cache replacement strategy, the Least Unified Value strategy (LUV) to replace the Least Recently Used (LRU) that SACCS was based on. This paper studies the advantages and the drawbacks of the new proposed strategy, comparing it with different categories of cache replacement strategies.

***Keywords*—**Cache consistency, hybrid algorithm, and mobile environments

I. INTRODUCTION

IN mobile computing environments, where low powered devices are used to access and query databases over relatively low-bandwidth wireless channels, caching frequently accessed data objects will reduce bandwidth usage and delays perceived by users.

In mobile environments, caching is more challenging due to the mobility of the users and the disconnected modes, which arise due to the battery power saving measures or the unpredictable disconnection of wireless networks. However, having a copy of the data in the cache is not sufficient; the cache should also provide the users a fresh data on each hit.

Broadcasting was assumed to be an effective method for data dissemination, which consumes little bandwidth. Several methods for data distribution had been suggested to guarantee the cache consistency in mobile environments. Some used stateless servers to maintain the mobile environment [1][2][3], others stateful servers [4]. Combining each approaches positive features, the Scalable Asynchronous Cache Consistency Scheme (SACCS) maintenance scheme was proposed. It was based on the Least-Recently-Used (LRU) cache replacement strategy [5][6].

Aline Zeitunlian is with the Department of Computer Science and Mathematics at the Lebanese American University, P.O. Box 13-5053 Chouran, Beirut, Lebanon 1102 2801. Email: aline.zeitunlian@lau.edu.lb

Ramzi A. Haraty is with the Department of Computer Science and Mathematics at the Lebanese American University, P.O. Box 13-5053 Chouran, Beirut, Lebanon 1102 2801. Phone: 961 1 867620, Fax: 961 1 867098, Email: rharaty@lau.edu.lb.

To support the cache consistency maintenance algorithms, it is important to have an efficient cache replacement policy, for after all mobile units have limited disk storage and not all data objects can be cached. In this work, we propose the least-unified value algorithm (LUV) [7] to be used with the SACCS cache consistency maintenance scheme and compare it with the other four cache replacement strategy categories. LUV is a cache replacement technique that associates a value to each object in the cache and when needed replaces it with the object with the smallest value. This policy considers the reference potential and the retrieval cost of the data object per unit size.

This work is organized into six sections. Section 2 provides a literature review of the approaches proposed for cache consistency, invalidation strategies, and replacement policies of mobile environments. Section 3 describes the SACCS maintenance approach. In section 4 we present SACCS and LUV and the other cache replacement techniques. Section 5 presents the experimental results of the LUV cache strategy with SACCS as compared to the four different class strategies. Finally, in section 6 we provide a conclusion and discuss the future work.

II. LITERATURE REVIEW

With the development of wireless communications a new model of distributed computing was introduced. It is more challenging and difficult than the other client/server based environments, since users can connect from different access points and may stay connected while on the move, at the same time its performance relies on the wireless bandwidth communication and the battery power.

A mobile unit (MU) communicates via an MSS (Mobile Support Stations) over a wireless channel. The wireless channel has upload channel and download channel. MUs use the upload channel to submit queries to the server, while the MSSs disseminate information or respond to the MU via the download channel. Each MSS is responsible for the MUs within a given geographical or logical area, known as a cell. Therefore, when an MU leaves a cell serviced by an MSS, a handoff protocol transfers the responsibility to the MSS of the new cell. This is shown in Figure 1.

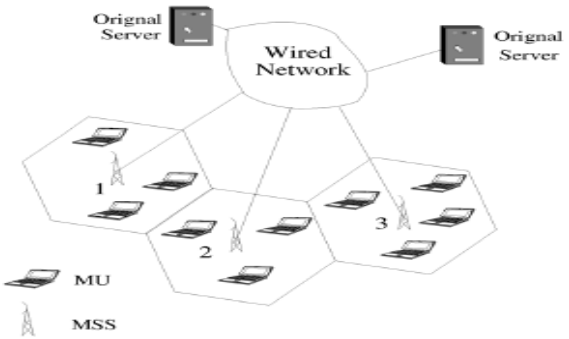

Fig. 1 Wireless data communication system architecture [6].

A mobile unit may move through the cells and may disconnect from the network. After an unknown time of disconnection an MU can reconnect to a different MSS.

For delivering data there exist two different systems; push-based and pull-based [8]. In push-based systems, the server decides to send information either periodically or sporadically to the clients without waiting for their requests. In pull-based systems, clients send messages to the server to request for data. Broadcasting minimizes the number of uplink requests. By broadcasting invalidation reports (IR)s clients are notified about the cached items changes'. Yet a client may miss IRs when disconnected during broadcast and this has its drawbacks.

To maintain cache consistency three different types of algorithms were suggested. In stateless approaches, an MSS has no knowledge of MUs cache contents. The MSS periodically sends invalidation reports to the MUs. While at an MU, a data object request cannot be served until the next IR. The advantage for stateless approaches is that they are easy to manage. Their drawbacks are: they are not scalable to large database; their access latency on average is always longer than half of the broadcast period and finally at reconnection after a long disconnection all cache entries are deleted, even the valid data objects. The stateful approaches were suggested by Barbara and Imielinski [1][3][9][10][11][12][13][14]. In stateful approaches, an MSS keeps the state of each object for every MU cache and broadcasts their IRs only. Kahol et al. proposed a scheme that minimizes the overhead for MUs to validate their caches when reconnected, using stateless servers and asynchronous invalidation messages [4].

As for the hybrid approach Scalable Asynchronous Cache Consistency Scheme (SACCS), the MSS identifies only the data objects that might be valid in MU caches. It does not broadcast IRs periodically. The uncertain and ID-only states of an MU allow handling of sleep-wakeup patterns and mobility. All these improve the broadcast channel efficiency [5][6][15].

A cache replacement strategy decides which object to evict from the cache when no space is available to store additional objects. It is based on several factors: recency, frequency, cost for fetching and size. To determine the effectiveness of a replacement strategy, certain metrics are measured such as the *cache hit ratio*, *byte hit ratio*, *delays.* Xu and Hu's proposed the (Min_SAUD) [16], Yin et al. presented a generalized target-driven cache replacement policy for mobile environments [17]. [15][18][19] suggested cache consistency algorithms that integrated cache replacement and prefetching algorithms to efficiently maintain the read-only transactions data requirements for mobile hybrid data delivery environments. The first presented the Greedy Dual Utility cache replacement policy and the second Multi-version integrated caching and prefetching policy. A different replacement strategy than the conventional ones was suggested by Santhosh et al. which was based on semantic [20].

Web caching, like mobile data caching, aims to reduce network traffic, server load, access delays and is again impacted by the replacement strategy. Rabinovich and Spatscheck presented an overview of web caching and replications [21]. The first classification of replacement strategies for web caching was given by Aggarwal et al. who proposed three categories: direct extensions of traditional strategies, key-based and function-based [22]. Later, Podlipnig and Boszormenyi classified them as follows: recency-based, frequency-based, recency/frequency-based, function-based and randomized strategies [23]. Certainly, each class of strategies has its own advantages and disadvantages.

III. SCALABLE ASYNCHORONOUS CACHE CONSISTENCY SCHEME (SACCS)

In SACCS, the MSS is only responsible for identifying the data objects of the database that might be valid in the MU caches. To save downlink bandwidth usage, SACCS also reduces the periodic IR messages broadcasted. In addition to these two features, SACCS added two different states for data objects in MU caches, they are *uncertain* and *ID-only*, that allow handling of random sleep-wakeup patterns and mobility.

In SACCS, they used the LRU replacement algorithm [5][6] and it was for systems with read-only transactions.

A. The SACCS Cache Management

In the server, each data object has a flag bit, which changes when the data is retrieved to indicate that a valid copy is available in the cache. Consequently, when this data object is updated, the server immediately broadcasts its IR and resets the flag bit to indicate that the cached data object is not valid anymore. Therefore, until the flag is reset no update requires broadcast of IR.

At an IR broadcast, an MU is either in an awake or in a sleep state. If the MU is awake then the state is changed from *valid* to *ID-only*. If the MU is disconnected, then the IRs are ignored and data are unaffected. However, when an MU wakes up after a disconnection, all *valid* state cached data objects are changed to *uncertain* state.

In SACCS using the LRU, every time a data is cached or is already found in the cache, it is moved to the head of the cache list. In case the cache is full and a new data needs to be cached, to accommodate it and make enough space data entries are deleted from the tail of the cache list. In case the cache needs to be refreshed, to validate the data of the cache, all data with *uncertain* or *ID-only* state are allocated their original places and if there is not enough space, then data entries found at the tail are removed.

IV. SACCS AND THE CACHE REPLACEMENT POLICIES

In general, cache replacement strategies affect hit rates; however, they are not the limiting factor for caching. Each strategy has certain targets and defines its metric.

While SACCS is based on (LRU), in this work; we examine the SACCS using a value-based function, the LUV, which emphasizes the reference information of the object as well as considering the fetch cost of the object and its size.

A. A Value-based Function: Least-Unified Value

The LUV replacement algorithm is based on time of all past references and the number of references [7]. It uses the complete reference history. The only weakness of this strategy is in how to consider the parameter tuning. The LUV replacement strategy is being a value-based strategy, calculates for every data object *i* a value *V(i)*, defined by the following formula

$$V(i) = W(i).p(i)$$

where *W(i)* is the relative cost to fetch the object from its original server, calculated as the ratio of fetching cost of object *i* from the server (*c(i)*) and its size *(s(i))*.

$$W(i) = c(i)/s(i)$$

while *p(i)* is the "probability" that object *i* is referenced in the future and is calculated as

$$p(i) = \sum_{k=1}^{fi} (t_c - t_k)$$

t_c being the current time and t_k the oldest request time in a window of *k* request times. To give more weight to more recent references *F(x)* should be a decreasing factor. A possibility for the function is $F(x) = \frac{1}{2}^{\lambda x}$ ($0 \leq \lambda \leq 1$). Note that λ converging to 1 reduces it to LRU where only the last reference time is considered, while λ converging to 0 reduces it to a weighted LFU, counting the number of previous references.

B. The New Algorithm for SACCS and its Description

The algorithm first checks if the data is available in the cache and in valid state then the time of the last reference is updated and the

new value calculated. Adding this new value to the heap, it restores the heap property. If the data is available in cache with uncertain state, it calculates the new value at the new referenced time, and then sends a message to check the validity of the data. In case the data is in ID-only state or not available in the cache it fetches the data and its value from the server, adds the time of the last reference and calculates the new value, inserting it to the heap, it adjusts the heap. If there is not enough space to download the data then it finds the data object with the lowest value and replaces it with the new data object, adds the value of the newly introduced data object and restores the heap. The algorithm for the new proposed cache replacement strategy is shown in Figure 2.

d_x = data object,
M_x = message for the data object,
d_y = data object that will be replaced, LUV= value calculated for the data object,
L is the minimum value.
Case 1: d_x is in cache and valid
then calculate the LUV value
return d_x to the application.
Case 2: d_x is in cache and uncertain
then calculate the new LUV value
send uncertain message to the server.
Case 3: d_x is not cached or ID-only
Send cache missing message to the server.
Wait for message M_x to appear at downlink channel
If M_x is confirmation then set the state of d_x as valid
Return d_x to the application,
If M_x is the data item d_x then
While there is not enough space for d_x
Find min value L = Minimum LUV value for data object y belonging to the cache
Evict the d_y such that LUV Value of y = L;
Keep value of the evicted data object
End while
Bring M_x into cache
Calculate its LUV value
Return M_x to the application.

Fig. 2 New SACCS Algorithm

LUV uses the heap structure to maintain the ordering of data according to their LUV values. The root of the heap has the lowest value. Therefore, the object found at the root of the heap is the object to be deleted. This is repeated until we get enough space for the incoming data object. Every deletion and insertion operation has a complexity of $O(\log_2 N)$. For every object's updated value, the heap is adjusted. The time complexity for every adjustment operation is $O(\log_2 N)$.

C. The Other Cache Replacement Strategies

To examine the efficiency of the proposed strategy LUV, we compared it with four different cache replacement algorithms, each belonging to a different classification of cache replacement strategies.

A Recency-Based Strategy: Least Recently Used (LRU)

This class' strategies in general replace objects that were used least recently. Their implementation is fairly easy. The LRU was already used for SACCS.

A Frequency-Based Strategy: Least Frequently Used (LFU)

This class' strategies replace data objects that were used least frequently. They are popular and easy to be implemented. In the LFU cache replacement policy, the frequency of references for each data entry in the mobile user cache list is counted. The tail of the list contains the data with the minimum number of accesses.

A Recency/Frequency Based Strategy: Least Recently/Frequently Used (LRFU)

The LRFU policy combines the two policies (LRU and LFU) and results in a policy that is better than both. To each data object it assigns a value. Every time the data object is referenced a weighing function F(x) is calculated which considers the data objects reference time span from the past to the current [24].

A Randomized Strategy: Random

It is different from the previous strategies, a nondeterministic approach. The random strategy uses randomized decisions to remove and replace an object from the cache [23]. It does not need special data structure for inserting or deleting object and is simple to implement. Its disadvantage is that it cannot be evaluated and different simulation runs will give different results.

V. PERFORMANCE EVALUATION

A. Environment

We tested the performance of our model by means of a simulated environment in C++. For our simulation we considered a single cell environment with 100 MUs as clients and each MU with identical cache size 300. We had also considered 1000 data objects of five types of access of random object sizes (bytes) and variable average update interval (sec).

The sleep wakeup process is modeled as two-state Markov chain with MUs alternating between sleep and awake states. Each MU has a sleep-wakeup period randomly picked from the set of values (500, 1000, 1500, 2000, 2500) sec. The sleep ratio is picked from (0.1, 0.3, 0.5, 0.7, 0.9) and the request arrival rate from (1/10, 1/60, 1/110, 1/160, 1/210). When an MU is in the sleep state, all requests are ignored. The query delay is counted as 0, when a requested data object is available at the MU. Otherwise, the query delay is counted as the time interval between the query response and query initiation. An uplink is counted when a query is retrieved from the original station through an uplink channel. A zipf-like distribution for MU access pattern is used in the simulation with z equal to 1. The update process for a data object and the arrival requests follow a Poisson distribution. The channel is used for downlink and uplink data transmission with a bandwidth 1250 bps. Uplink message size is assumed 64 bytes and downlink message size as 64 bytes. As for the function parameters used for the LUV, we considered λ to be equal to 0.5 and considered variable and random fetching cost and size ratios.

B. Simulation Results

The performance of SACCs based on LUV value-based cache replacement policy is evaluated and compared to SACCS based on LRU, LFU, LRFU and Random representative cache replacement policies of the remaining other four categories of strategies.

When an MU receives a query, if the queried data object is valid in the cache, a cache hit is counted, and no uplink is needed for the query. The higher the hit ratio is the fewer the uplink per query.

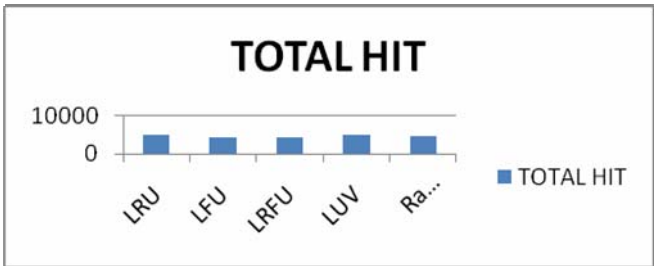

Fig. 3 Total Hit for Simulation

TABLE I TOTAL QUERY, TOTAL HIT, TOTAL MISS

	TOTAL QUERY	TOTAL HIT	TOTAL MISS
LRU	**16751**	**5086**	**11666**
LFU	**16688**	**4172**	**12519**
LRFU	**16695**	**4491**	**12206**
LUV	**16726**	**5126**	**11599**
Random	**16567**	**4695**	**11871**

As it is shown in Table 1, the total hit for LUV is the highest. Figure 4. shows that the LUV cache replacement policy improves performance since the data requested is available in the cache, it will reduce the IR message broadcasts, avoiding the unnecessary traffic and retaining the valid data objects of the MU.

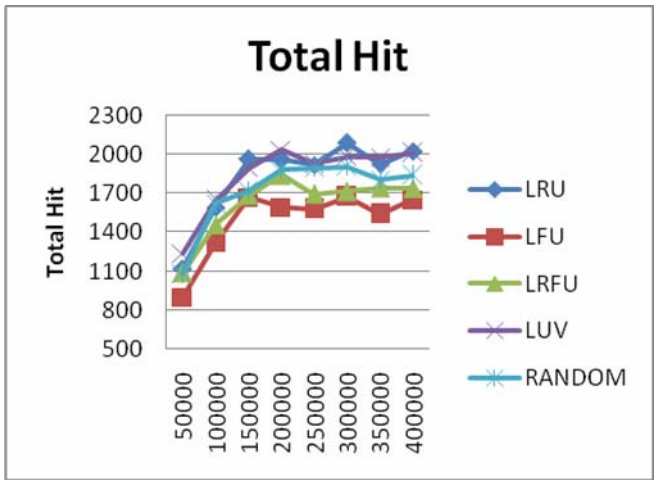

Fig. 4 Number of Hits vs. Time

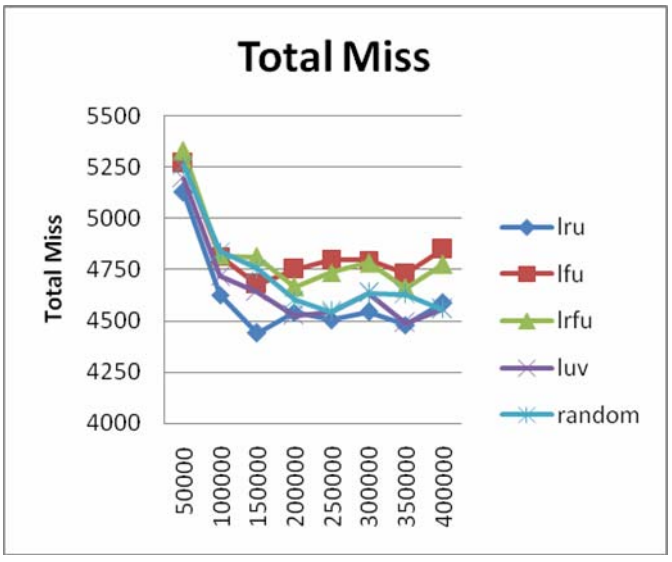

Fig. 5 Number of Misses vs. Time

The results of the number of misses and number of hits (Figure 4. and Figure 5) and the miss and hit ratios (Figure 6. and Figure 7) are depicted for the five cache replacement strategies, over eight simulation time units with an interval of 50000 of simulation time. The miss (hit) ratio is the ratio of the number of unfound (found) data items in the cache over the number of all requested data. The worst hit ratios performance is for LFU, while LRU and LUV have the best hit ratio performances interchangeably. However, on an average the LUV outperforms the LRU.

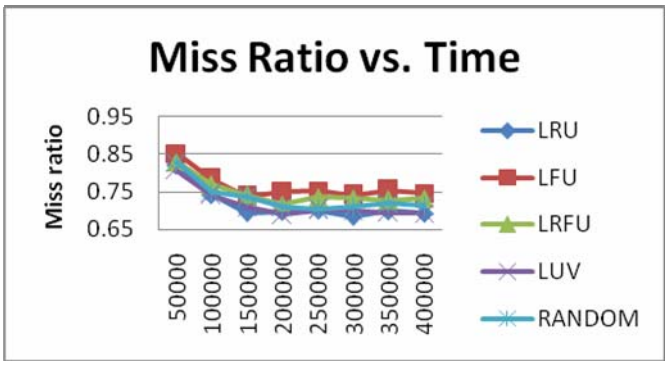

Fig. 6 Miss Ratio vs. Time

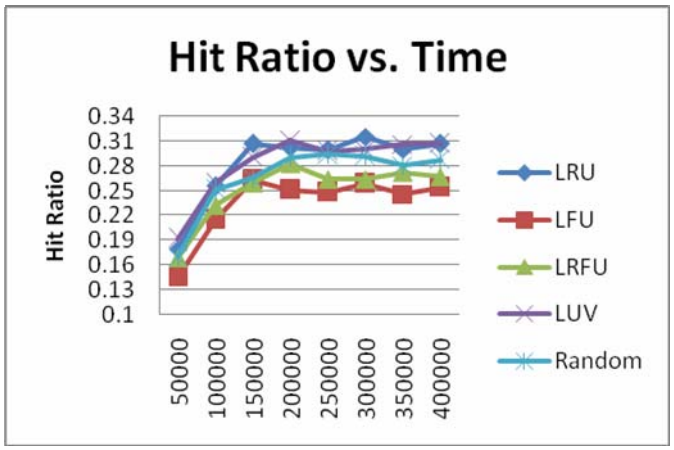

Fig. 7 Hit Ratio vs. Time

A delay is the period of time between the time a request is issued and the time the result is received by the mobile application user. The average access delay is an important measurement of system performance. A shorter delay implies better performance. The total delay results of our simulation are presented in Figure 8. and the average delay results in Figure 9.

It is obvious that the tradeoff between energy cost and access latency is a hard one, we can decrease the uplink and download messages or improve the access latency to decrease the energy cost.

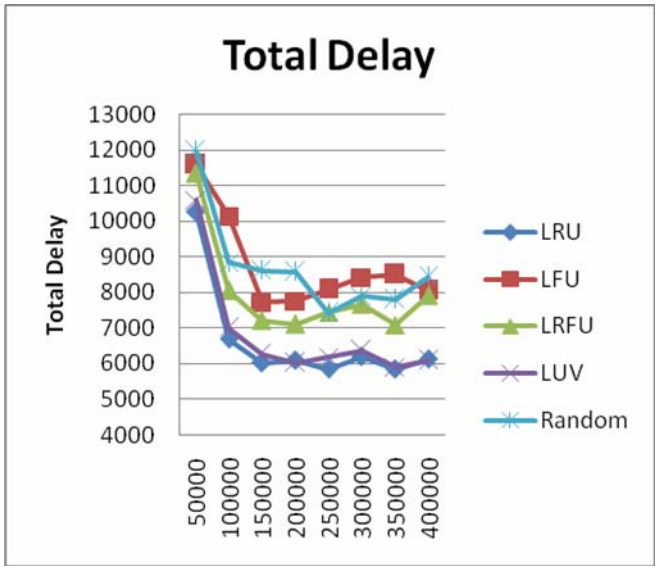

Fig. 8 Total Delay vs. Time

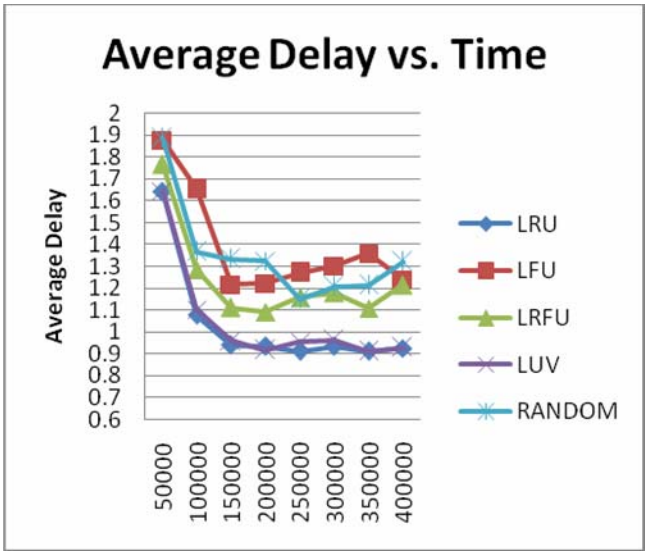

Fig. 9 Average Delay Vs. Time

In Figure 10. based on the Table 2. results show that LUV results in less bytes/query, outperforming the other four cache replacement strategies. This is due to the fact that LUV not only considers the most recent data information but also future references according to their fetch cost. The decision of evicting data objects with low fetching costs is a smart way to save power consumption at a later stage.

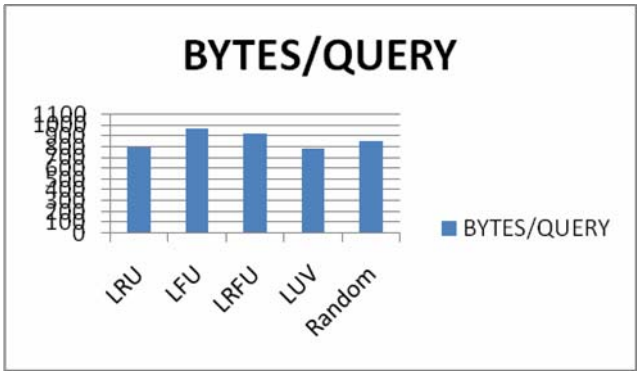

Fig. 10 Bytes per Query

TABLE II BYTES PER QUERY

	BYTES/QUERY
LRU	**793.541**
LFU	**978.204**
LRFU	**920.849**
LUV	**791.037**
Random	**859.57**

In Figure 11. based on the results in the Table 3. LUV has the lowest ratio for data download/query. This means the value-based cache replacement strategy LUV is quite efficient and its selection of the victims set had saved unnecessary downloads. Since the algorithm favors the data objects that have low fetch cost values, it has saved fetching costs, which implies that it is less power consuming.

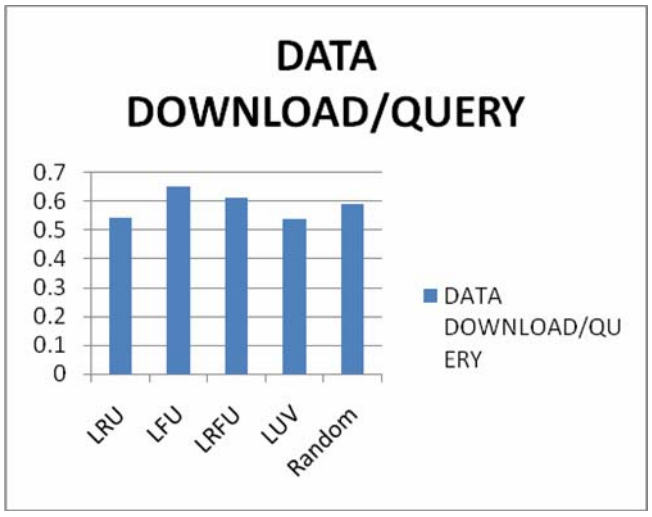

Fig. 11 Data Download per Query

TABLE III DATA DOWNLOAD PER QUERY

	DATA DOWNLOAD/QUERY
LRU	**0.541938**
LFU	**0.647951**
LRFU	**0.609404**
LUV	**0.534617**
Random	**0.585622**

VI. CONCLUSION

Caching is a good solution for mobile environments that are characterized with several constraints such as low bandwidth for uplink, irregular connections, and limited client resources. However, caching has some limitations too. Several approaches had been suggested to maintain cache consistency. In the stateful approaches the server knows what data was cached in which mobile unit while in stateless approaches the server is unaware of the information. Both approaches having drawbacks, an efficient and scalable hybrid caching maintenance approach SACCS has been suggested, which is based on LRU. A cache being limited in size, a cache replacement strategy plays a central role. In this work, we proposed the value-based function LUV for cache replacement algorithm to be implemented with SACCS. Based on the complete history, LUV selected the set of victims considering the potential of objects that can be referenced in the near future and at the same time the cost of fetching the data strategy, and it was shown to be an efficient strategy. A good replacement policy is one that is used as infrequently as possible to generate the same hit rates. The proposed strategy was compared with other strategies which belong to different categories of cache replacement strategies. Since in our simulation we used a fixed parameter λ to calculate the function value for a data object, for the future we propose to find an adaptive function for the λ parameter that adjusts according to the query rate and client disconnection. Also, in the function, we need to consider the update frequency of the data object.

REFERENCES

[1] Barbara, D. and Imielinski, T. (1994). Sleepers and Workaholics: Caching Strategies in Mobile Environments. ACM, SIGMOD.

[2] Cao, G. (2002, June). Proactive Power-Aware Cache Management for Mobile Computing Systems. IEEE. Transactions on Computers. Volume 51. No. 6. pp. 608-621.

[3] Jing, J. Elmagarmid, A. Helal, A. and Alonso, R. (1997). Bit-Sequences: An Adaptive Cache Invalidation Method in Mobile Client/Server Environments. ACM. Mobile Networks and Application 2. pp. 115-127. 1997.

[4] Kahol, A. Khurana, S. Gupta, S.K.S. and Srimani, P.K. (2001, July). A Strategy to Manage Cache Consistency in a Disconnected Distributed Environment. IEEE. Transactions on Parallel and Distributed Systems. Vol. 12. No.7. pp. 686-700.

[5] Wang, Z. Das, S. Che, H and Kumar. M (2003). SACCS: Scalable Asynchronous Cache Consistency Scheme for Mobile Environments. IEEE, Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03). pp.1-6.

[6] Wang, Z. Das, S.K. Che, H and Kumar, M. (2004, November). A Scalable Asynchronous Cache Consistency Scheme (SACCS) for Mobile Environments. IEEE Transactions on Parallel and Distributed Systems. Vol. 155, no. 11, pp. 983-995.

[7] Bahn, H. Koh, K. Sam, N. and Min, S. L. (2002). Efficient Replacement of Nonuniform Objects in Web Caches. IEEE. June, 2002. pp.65-73.

[8] Barbara, D. (1999). Mobile Computing and Databases - A Survey. IEEE. Transactions on Knowledge and Data Engineering, Volume 11, No.1, January/February 1999, pp. 108-117.

[9] Wu, K.L. Yu, P.S. and Chen, M.S. (1996). Energy-Efficient Caching for Wireless Mobile Computing. IEEE. pp. 336-343.

[10] Hu, Q. and Lee, D.K. (1998) Cache Algorithms Based on Adaptive Invalidation Reports for Mobile Environments. ACM. Cluster Computing. Volume 1. pp. 39-50.

[11] [Cao, G. (2002). Adaptive Power-Aware Cache Management for Mobile Computing Systems. http://www2002.org/CDROM/poster/88.pdf.

[12] Cao, G. (2002) On Improving the Performance of Cache Invalidation in Mobile Environments. Mobile Networks and Application, 7, pp. 291-303. Kluwer Academic Publishers. Netherlands.

[13] Cao, G. (2003, September/October). A Scalable Low-latency Cache Invalidation Strategy for Mobile Environments. IEEE. Transactions on Knowledge and Data Engineering. Volume 15, No. 5. pp. 1251-1265.

[14] Madhukar, A. and Alhajj, R. (2006, April 23-27). An Adaptive Energy Efficient Cache Invalidation Scheme for Mobile Databases. ACM. SAC 2006. April 23-27, 2006, Dijon, France. pp. 1122-1126.

[15] Shen, H. Kumar, M. Das, S.K. and Wang, Z. (2005). Energy-Efficient Data Caching and Prefetching for Mobile Devices Base on Utility. Mobile Networks and Applications 10, pp. 475-486.

[16] Xu, J. and Hu, Q. (2001). An Optimal Cache Replacement Policy for Wireless Data Dissemination Under Cache Consistency. IEEE. pp.267-274.

[17] Yin, L. and Cai, Y. (2003) A Generalized Target-Driven Cache Replacement Policy for Mobile Environments. Proceedings of Symposium on Applications and the Internet, 2003. pp. 14-21. 27-31 January 2003. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.9274

[18] Shen, H. Kumar, M. Das, S.K. and Wang, Z. (2004). Energy-Efficient Caching and Prefetching with Data Consistency in Mobile Distributed Systems. IEEE. Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04).

[19] Seifert, A. and Scholl, M. H. (2002). A Multi-Version Cache Replacement and Prefetching Policy for Hybrid Data Delivery Environments. ACM. Proceedings of the 28th VLDB Conference, Honk Kong, China, 2002.

[20] Santhosh, S. and Shi, W. (2005) A Semantic-Based Cache Replacement Algorithm for Mobile File Access. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.9274.

[21] Rabinovich, M. and Spatscheck, O. (2002). Web Caching and Replication (2nd ed.) Boston: Addison-Wesley.

[22] Aggarwal, C. Wolf, J.L. and Yu, P.S. (1999, January-February). Caching on the World Wide Web. IEEE. Transaction on Knowledge and Data Engineering, Volume 11, No. 1, January/February 1999. pp. 94-107.

[23] Podlipnig. S and Boszormenyi, L. (2003, December). A Survey of Web Cache Replacement Strategies. ACM Computing Surveys. Volume. 35. No. 4. pp.374-398.

[24] Lee, D. Choi, J. Kim, J.H. (1999). On the Existence of a Spectrum of Policies that Subsumes the Least recently Used (LRU) and Least Frequently Used (LFU) Policies. ACM. SIGMETRICS '99 5/99 Atlanta, Georgia, USA. pp. 134-143.