Open access • Journal Article • DOI:10.1109/TPDS.2004.60

A Scalable Asynchronous Cache Consistency Scheme (SACCS) for mobile environments — Source link

Zhenhua Wang, Sajal K. Das, Hao Che, Mohan Kumar

Published on: 01 Nov 2004 - IEEE Transactions on Parallel and Distributed Systems (IEEE Computer Society)

Topics: Cache algorithms, Cache invalidation, Cache, Cache pollution and Page cache

SACCS: Scalable Asynchronous Cache Consistency Scheme for Mobile Environments

Zhijun Wang, Sajal Das, Hao Che and Mohan Kumar
Center for Research in Wireless Mobility and Networking
Department of Computer Science and Engineering
The University of Texas at Arlington, Arlington, TX 76019, USA

Abstract

In this paper, we propose a novel cache consistency maintenance scheme, called $Scalable\ Asynchronous\ Cache\ Consistency\ Scheme$ (SACCS), for mobile environments. It relies on the following three key features: (1) Use of flag bits at server and MU's cache to maintain cache consistency; (2) Use of an identifier (ID) for each entry in MU's cache after its invalidation in order to maximize the broadcast bandwidth efficiency; (3) Rendering all valid entries of MU's cache to $uncertain\ state$ when it wakes up. These three features make the SACCS a highly scalable algorithm with minimum database management overhead. Comprehensive simulation results show that the performance of SACCS is superior to those of existing algorithms.

I. Introduction

Wireless communication has increasingly become an important means for people to access various kinds of dynamically changing data objects, such as news, stock price, and traffic information. However, wireless mobile computing environments are limited by communication bandwidth and battery power [1], and have to live with mobile user's (MU) disconnectedness and mobility. Thus, data communication in wireless mobile networks is more challenging than that in wired networks.

$Caching$ frequently accessed data objects at the local buffer of an MU is an efficient way to reduce query delay, save bandwidth and improve system performance. But frequent disconnection and roaming of an MU make $cache\ consistency$ a difficult issue in wireless mobile computing environments. A successful strategy must efficiently handle both dissconnectedness and mobility. Broadcast has the advantage of being able to serve an arbitrary number of MUs with minimum bandwidth consumption. Thus, an efficient mobile data transmission architecture should carefully design its broadcast and cache management schemes to maximize $bandwidth\ utilization$ and minimize $average\ query\ delay$. A good mobile data transmission architecture should also be $scalable$, in the sense that it works efficiently for large database systems and supports a large number of MUs.

Acknowledgement: This work is supported by a grant from Texas Advanced Research Program, TXARP Grant Number: 14-771032.

In the literature, there are two types of cache consistency maintenance algorithms for wireless mobile computing environments: stateless and stateful. In the $stateless$ approach, the server is unaware of client's cache content. The client needs to check the validity of cached entries from the server before each query. Even though stateless approaches employ simple database management, their scalability and ability to support disconnectedness are poor. On the other hand, $stateful$ approaches are scalable, but incur significant overhead due to server database management. Therefore, there is a need for developing scalable and efficient algorithms for maintaining cache consistency in mobile environments.

Motivated by the need for a scalable and efficient cache consistency maintenance mechanism, we propose a novel algorithm, called $Scalable\ Asynchronous\ Cache\ Consistency\ Scheme$ (SACCS) that maintains cache consistency between the mobile support station (MSS) and MU's caches. SACCS is a highly scalable, efficient, and low complexity algorithm due to the following three key features: (1) Use of flag bits at server and MU's cache to maintain cache consistency; (2) Use of an ID for each entry in MU's cache after its invalidation in order to maximize the broadcast bandwidth efficiency; (3) Rendering all valid entries of MU's cache to $uncertain\ state$ when it wakes up.

Comprehensive simulation results show that SACCS offers superior performance over existing algorithms. For example, in a system with 5 types of MU access patterns and 5 types of data object update frequencies, SACCS can support about 44% and 270% more MUs than AS and Timestamp (TS) schemes, respectively. It also guarantees that the average access delay is no larger than 8 seconds.

The rest of the paper is organized as follows. Section II gives a brief overview of the related work. In Section III, a detailed description of the SACCS algorithm is presented. Section IV presents comprehensive simulation results of our algorithm and compares with those of existing approaches. The conclusions are drawn in Section V.

II. Related Work

We summarize existing stateless and stateful approaches in this section. In the $stateless$ approach [2]-[7], an MSS assumes no knowledge of MU's cache contents. An MSS sim-

ply sends IRs to its MUs periodically. At an MU, a data object request cannot be serviced until the next IR from the MSS is received. In the $stateful$ approach [8], an MSS maintains object state for each MU's cache and only broadcasts IRs for those objects.

Barbara and Imielinksi [2] proposed three stateless algorithms: Timestamps (TS), Amnesic Terminals (AT) and Signature (SIG). In these algorithms, the MSS broadcasts IRs, which include all data object IDs updated during the past kL seconds (where k is a positive integer), every L seconds. The advantage of these algorithms is that an MSS does not maintain any state information about its MU's caches, which makes the MSS database easy to manage. However, there are several drawbacks with these algorithms. First, they do not scale well to large databases and/or fast updating data systems, due to increased number of IR messages. Second, the average access lantency is always longer than half of the broadcast period, simply because all requests can be answered only after the next IR. Finally all cached data objects are dropped if the sleep time is longer than kL.

In order to handle the long sleep-wakeup patterns, several algorithms have been proposed. For example, in the bit-sequence (BS) algorithm due to Jing, et al. [3], all cache entries are deleted only when half or more of data entries in the cache have been invalidated. However, the model requires the broadcast of a larger number of IR messages than TS and AT schemes. Wu et al. [5] proposed an uplink validation check scheme that can deal with long sleep-wakeup patterns better than TS and AT. But this approach requires more uplink bandwidth and cannot deal with $very$ long sleep-wakeup patterns.

Very few stateful cache consistency maintenance algorithms have been proposed for wireless mobile computing environments. Kahol, et al. [8] proposed an asynchronous stateful (AS) algorithm to maintain cache consistency. In AS, an MSS records all retrieved data object for each MU. When an MU first retrieves a data object after it wakes up, the MSS sends an IR, based on the MU's cache content record and sleep-wakeup time, to that particular MU. Whenever an MSS receives an update from the original server for each recorded data object, it immediately broadcasts that data object's IR to MUs. The advantage of the AS scheme is that the MSS avoids unnecessary broadcast of IRs to MUs and can deal with any sleep-wakeup pattern without losing valid data objects. However, in order to maintain each MU's cache state in the MSS, the MSS must record all cached data objects for each MU. Hence an MU can only download data objects which it requested through the uplink. This makes the channel utility inefficient and sensitive to the number of MUs.

III. Scalable Asynchronous Cache Consistency Scheme (SACCS)

In this paper, we propose a novel scalable asynchronous cache consistency scheme (SACCS) to maintain MU's cache consistency for a read-only system. Strictly speaking, SACCS is a hybrid of stateless and stateful approaches in the sense that it maintains minimum state information. However, unlike the stateful algorithm [8] which requires the MSS to remember all data objects for each and every MU's cache, SACCS only requires the MSS to identify which data objects in its database might be valid in MU's caches. This makes the management of the MSS database much simpler. On the other hand, unlike existing synchronous stateless approaches, SACCS does not require periodic broadcast of IRs, thus greatly reducing IR messages that need to be sent through the downlink broadcast channel. Moreover, by adding uncertain and ID-only states in MU's caches, SACCS allows easy handling of arbitrary sleep-wakeup patterns and mobility, and strong cooperation among all MUs, which greatly improves broadcast channel efficiency. The following subsections describe the proposed algorithm in detail.

A. Data Structures and Message Formats

For every data object with unique identifier x, the data structure for MSS and MU's cache are as follows:

- (d_x, t_x, f_x): data entry format for each data object in MSS. Here d_x is the data object, t_x is the last update time for the data object and f_x is a flag bit.
- (d_x, ts_x, s_x): data entry format in an MU's cache. Here d_x is defined above, ts_x is the time stamp indicating the last updated time for the cached data object d_x, and s_x is a two-bit flag identifying four data entry states: $valid\ d_x$, $uncertain\ d_x$, $uncertain\ d_x$ with a $waiting\ query$ and $ID-only$, respectively.

The communication messages are as defined in Table I.

B. MU Cache Management

As the focus on this paper is the cache consistency maintenance design, we simply use the LRU (Least Recently Used) based replacement algorithm for the management of MU's caches. The impact of the cache replacement algorithms on SACCS is a subject of future study.

In our LRU based replacement scheme, a newly cached data object or a cached data object which receives a hit, is moved to the head of the cache list. When a data object needs to be cached while the cache is full, data entries with $s_x \neq 2$ from the tail are deleted to make enough space to accommodate this new data object (the data object with $s_x = 2$ must be kept because some requests are waiting for its confirmation).

Any refreshed data objects from uncertain state or ID-only state are placed in their original location and again, if necessary, enough data entries from the tail are removed.

C. Algorithm Description

We present two procedures, i.e., MSSMain() and MUMain(), for the SACCS. The MSS continuously executes the MSSMain() and each MU continuously executes the MUMain() procedure during its awake period.

The psudocodes MSSMain() and MUMain() are shown in Figures 1 and 2.

TABLE I
COMMUNICATION MESSAGES IN SACCS

Name	Sender	Receiver	comments
$Update(x, d_x^{'}, t_x^{'})$	original servers	MSS	indicating d_x has been updated to $d_x^{'}$ at time $t_x^{'}$
$Vdata(x, d_x, t_x)$	MSS	MUs	broadcast valid data object d_x with update time at t_x
$IR(x)$	MSS	MUs	indicating cached d_x is invalid
$Confirmation(x, t_x)$	MSS	MUs	indicating d_x is valid if $ts_x = t_x$
$Query(x)$	MUs	MSS	query for data object d_x
$Uncertain(x, ts_x)$	MUs	MSS	verifying if d_x in unertain state with update time ts_x is valid

```
MSSMain() {
  IF ( MSS gets Query(x) message )
      fetch data entry x from the database
      broadcast Vdata(x, d_x, t_x) to all MUs
      IF (f_x == 0)
          set f_x = 1
  IF ( MSS gets Uncertain(x, ts_x) message )
      fetch data entry x from the database
      IF ( t_x = ts_x )
          broadcast Confirmation(x, t_x) to all MUs
      ELSE
          broadcast Vdata(x, d_x, t_x) to all MUs
      IF ( f_x == 0)
          set f_x = 1
  IF ( MSS gets Update(x, d_x', t_x') from original server )
      update the database entry with ID x as: d_x = d_x' and t_x = t_x'
      IF (f_x == 1)
          broadcast IR(x) to all MUs and reset f_x = 0
}
```

Fig. 1. MSSMain

```
MUMain() {
  IF ( MU receives a request for d_x)
      IF (d_x is valid in cache list )
          answer the request with cached data object d_x
          move the entry into cache list head
      ELSE IF ( d_x is in uncertain state )
          send Uncertain(x, ts_x) message to MSS
          add ID x to query waiting list
          set s_x = 2 and move the entry into cache list head
      ELSE IF (the entry x is ID–only entry in cache )
          send Query(x) to MSS
          remove the entry in cache
          add ID x to query waiting list
      ELSE
          send Query(x) to MSS
          add ID x to query waiting list
  IF ( MU receives a Vdata(x, d_x, t_x) message )
      IF (x is in query waiting list )
          answer the request with d_x
          remove the uncertain entry x if it exists in cache
          add the valid entry x at cache list head
      ELSE
          IF (entry x is ID–only entry in cache )
              download d_x to original entry location in cache
          ELSE IF (entry x is uncertain entry in cache )
              IF ( ts_x < t_x )
                  download d_x to orginal entry location in cache
                  set ts_x = t_x and s_x = 0
              ELSE
                  set the entry to valid entry ( s_x = 0)
  IF ( MU receives a IR(x) message )
      IF (d_x is valid or uncertain in cache)
          delete d_x and set s_x = 3
  IF ( MU receives a Confirmation(x, t_x) message)
      IF ( the entry x is uncertain entry in cache list)
          IF ( ts_x = t_x)
              set s_x = 0
              IF ( ID x is in query waiting list)
                  answer the request with d_x
          ELSE
              delete d_x and set s_x = 3
  IF ( MU wakes up from the sleep state)
      set all valid ( s = 0) entry into uncertain state ( s = 1)
}
```

Fig. 2. MUMain

D. Cache Consistency Maintenance

Let us explain how SACCS maintains consistency between an MSS database and MU caches. We assume that the consistency between the MSS database and original servers is maintained through wired network consistency algorithms [9], [10].

For each cached data object, SACCS uses a single flag bit f_x, to maintain the consistency between the MSS and MU caches. When d_x is retrieved by an MU, f_x is set, indicating that a valid copy of d_x may be available in an MU's cache. If and when the MSS receives an updated d_x, it broadcasts an $IR(x)$ and resets f_x. This action implies that there are no valid copies of d_x in any MU's caches. Furthermore, while $f_x = 0$, subsequent updates do not entail broadcast of $IR(x)$. The flag f_x is set again when the MSS services a retrieval (including request and confirmation) for d_x by an MU.

In mobile environments, an MU's cache is in one of two states: (i) awake or (ii) sleep. If an MU is $awake$ at the time of $IR(x)$ broadcast, the d_x copy is invalidated and an ID-only

entry is maintained by the MU. The data objects of an MU in the $sleep$ state are unaffected until it wakes up. When an MU wakes up, it sets all cached valid data objects (including d_x) into the uncertain state. Consequently, sleeping MUs and the cached object are unaffected by missing $IR(x)$ broadcast.

E. Efficiency and Cooperation

As mentioned earlier, a good cache consistency maintenance algorithm must be scalable and efficient in terms of the database size and the number of MUs. SACCS can handle large and fast updating data systems because the MSS has some knowledge of MU's cache. Only data entries which have flag bits set result in the broadcast of IRs when data objects are updated. Consequently, the IR broadcast frequency is the minimum of the uplink query/confirmation frequency and the data object update frequency. In this way, the broadcast channel bandwidth consumption for IRs is minimized.

Besides IR traffic, all other traffic in SACCS is also minimized due to the strong cooperation among the MU's caches. Specifically, due to the introduction of the uncertain state and the ID-only state for the MU's caches. The retrieval of a data object, d_x, from the MSS issued by any given MU brings the x entries in the uncertain or ID-only state in all the awake MU's caches to a valid state. Moreover, a single uplink confirmation for x in the uncertain state causes the x entries in uncertain state for all the awake MU's caches to either valid or ID-only state. The addition of the uncertain state also allows an MU's cache to keep all the valid data objects after it wakes up from arbitrary duration of sleep time. In contrast, for AS and TS algorithms, all the invalidated data objects are completely deleted from the MU's cache. This allows little cooperation among the MUs, resulting in a dramatic increase of traffic volume between the MSS and the MUs as the number of MUs increases (see in Section V). Although [6] improves the scalability of TS by retaining the invalid data objects, it reduces the cache efficiency by having to keep the invalid data objects, rather than IDs as is the case in our SACCS approach, in the MU's cache.

In contrast with the AS scheme which requires $O(MN)$ buffer space in the MSS to keep all the MU's cache state, our approach only requires $one\ bit$ per data object in the MSS to indicate if the IR broadcast is required when the data object is updated. Moreover, the added management overhead is minimal requiring only a single bit check and set/reset.

F. Mobility

Typically, sychronous stateless approaches handle MU's mobility by assuming that all MSSs broadcast the same IRs [7]. However, when the number of MSSs is large, such systems are not scalable. There is no an efficient way to handle the MU's mobility in the literature. In our approach, an MU roaming into a new cell is treated as if it just woke up from the sleep state, i.e., all the valid data objects are set to the uncertain state. The consistency is guaranteed with this approach and all valid data objects are retained. Also SACCS is simple in the sense that it is transparent to the MSSs involved. In our approach, the roaming effect is nothing more than the addition of a new sleep-wakeup pattern and should not have any significant impact on the overall performance. In this paper, we emulate the roaming effect by a sleep-wakeup pattern.

IV. Performance Evaluation By Simulation

A. Simulation Setup

We consider a single cell system with one MSS database and multiple MUs with identical cache size. The request process for each MU is assumed to be Poisson distributed and the update processes for data objects are also Poisson distributed. Also the sleep-wakeup process is modeled as a two-state Markov chain. The following parameters are defined:

- C: the cache size for MU.
- W: the channel bandwidth (bps).
- λ: the average request arrival rate for an MU.
- T_u: the average update interval for a data object (sec).
- T_s: the period of a sleep-wakeup cycle for an MU (sec).
- s: the ratio of the sleep time to the sleep-wakeup period for an MU.
- b_p: the size ($bytes$) of a data object.
- b_u: the uplink message size ($bytes$).
- b_d: the downlink invalidation or confirm message size ($bytes$).
- D: the average query delay (sec), i.e., the time interval between the time request is issued and the time the result is received by the application.
- UPQ: the uplink per query.
- L: the invalidation broadcast period (for TS scheme)(sec).
- wsz: the broadcast window size (for TS scheme).

In our simulation, we use a single channel with bandwidth W for both downlink and uplink data transmission. All messages are queued and serviced based on a first-come-first-serve discipline. All requests are ignored when an MU is in the sleep state. Error recovering cost and software overhead are ignored. When a requested data object is available at an MU's local cache, the delay D is counted as 0. A $Zipf-like$ distribution MU access pattern[11] is used in the simulation.

In the following subsections, we present a comprehensive comparison of the proposed SACCS with AS and TS algorithms in terms of metrics D and UPQ for three different cases. The average access delay D is an important measurement of system performance, a shorter D, a better performance. The UPQ is related to cache hit ratio. 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. Hence, higher hit ratio, fewer UPQ.

B. Case Studies

We present *three* cases. In each case, $b_u = 64$ and $b_d = 64$ for both SACCS and AS, and $b_u = 10$ and $b_d = 10$ for TS.

The bandwidth is set as $W = 10000\ bps$. Cache size C is in units of the number of data objects and the maximum number of ID-only entries is also set to C in the first two cases. Table II shows the parameter values for all cases. Case 3 has different MU access patterns, data object sizes and data object update frequencies. We will provide them later in detail and use @ to denote them in Table II. Each case study corresponds to a parameter effect indicated by * in the column.

Case 1: Effect of MSS Database Size

Table III presents the simulation results of D and UPQ for the three algorithms with different database sizes.

From Table III, we observe that SACCS outperforms the other two algorithms (AS and TS) on both performance metrics for different database sizes (N). When the database size reaches 12800, D for TS is over $160\ sec$. But for SACCS and AS, one can see a slow increase of the performance metrics as N increases all the way up to $N = 12800$.

In summary, the performances of SACCS and AS are not sensitive to the database size N and thus can be scaled to large database sizes. However, the performance of TS is sensitive to the database size, especially in terms of the delay performance, and hence TS cannot be scaled to large database sizes.

Case 2: Effect of Data Update Rate

Table IV gives the simulation results of the effect of update interval T_u. Again, as can be seen, SACCS outperforms AS and TS for all these performance metrics. As expected, as T_u reduces, both performance metrics increase for all the algorithms. However, SACCS and AS exhibit slower rates of increase than TS for all the metrics. In particular, when T_u is reduced to $10\ sec$, the delay performance of TS is unacceptably large (about $50\ sec$). At this update interval, SACCS achieves more than 50% delay performance gain over AS. For UPQ, SACCS outperforms AS and TS by 5% to 30%.

Case 3:Effect of the Number of MUs

In this case study, we consider a system similar to a real situation. We assume that a cell has 5 different MU query patterns as: $\lambda = 1/(10 + 50 * i)$, $s = 0.9 - 0.2 * i$ and $T_s = 500 * (i + 1)\ sec$ with i = 0,1,2,3 and 4.

Each query pattern group has $M/5$ MUs in it. We assume the query patterns for all the groups follow Zipf-like ($z = 1$) distribution. The access popularity ranking for the neighboring groups is shifted by 10, i.e., group 1 has decreasing popularity from data object 1 to 1000 and group 2 from 11 to 1000 and then from 1 to 10, and so on.

We also assume that there are 5 types of data objects in the MSS as: $b_p = 500 * (i + 1)$ and $T_u = 10^{i+1}\ sec$ with i = 0,1,2,3 and 4. Each data type group has $N/5$ data objects.

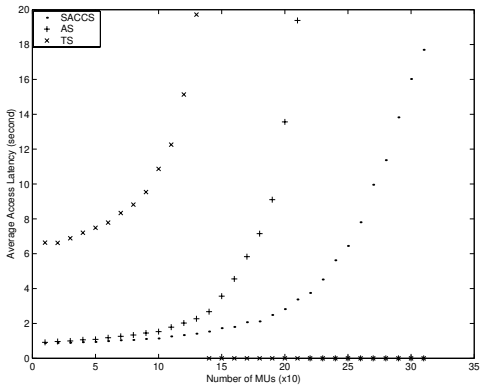

Fig. 3. Average Access Delay for 5 class MUs with 5 class data

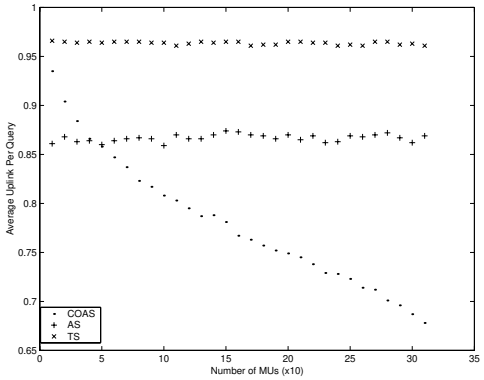

Fig. 4. Average Uplink per query for 5 class MUs with5 class data

The parameter values chosen above are based on the understanding that a more frequent query of MU normally means a shorter awake time and a faster updated data object usually has smaller size. The cache size $C = 150\ kbytes$. The maximum number of data ID-only entries that can be kept in each MU cache is set at 100.

Figures 3 and 4 depict D and UPQ versus M for the three algorithms, respectively. As one can see, SACCS scales much better than TS and AS in terms of both performance metrics. For example, at $D = 8\ sec$, the number of MUs that can be supported by TS, AS, and SACCS are about 70, 180, and 260, respectively. This means that SACCS can support about 44% and 270% more MUs than AS and TS, respectively. Also as M increases, UPQ stays a constant for both TS and AS, meaning that there is no cooperation among MUs. In contrast, UPQ drops almost linearly as M increases due to the strong cooperation among MUs. Once again, this implies SACCS is a highly scalable algorithm.

V. Conclusions and Future Work

In this paper, we proposed a Scalable Asynchronous Cache Consistency Scheme (SACCS) for mobile environments. Three key features of SACCS are: (i) use of flag bits at MSS

TABLE II
SIMULATION PARAMETER SET UP

Cases	N	M	C	λ	T_u	T_s	s	L	wsz	bp
$Case\,1$	$*$	100	100	$1/50$	1000	2000	0.5	20	5	1024
$Case\,2$	1000	100	100	$1/50$	$*$	1500	0.4	20	5	1024
$Case\,3$	1000	$*$	@	@	@	@	@	10	10	@

TABLE III
EFFECTS OF DATABASE SIZE FOR ZIPF-LIKE ($z = 1$) ACCESS PATTERN

N	100	200	400	800	1600	3200	6400	12800
$D(SACCS)$	0.175	0.296	0.429	0.561	0.669	0.783	0.898	1.014
$D(AS)$	1.014	1.234	1.458	1.688	1.901	2.117	2.394	2.688
$D(TS)$	12.364	13.176	13.862	14.429	14.984	15.492	17.455	161.987
$UPQ(SACCS)$	0.224	0.324	0.418	0.493	0.548	0.597	0.638	0.670
$UPQ(AS)$	0.690	0.737	0.779	0.815	0.837	0.849	0.877	0.887
$UPQ(TS)$	0.746	0.791	0.822	0.851	0.873	0.885	0.902	0.904

TABLE IV
EFFECTS OF DATA UPDATE FREQUENCY FOR ZIPF-LIKE ($z = 1$) ACCESS PATTERN

T_u	10	40	160	640	2560	10240
$D(SACCS)$	3.004	1.600	0.934	0.667	0.596	0.573
$D(AS)$	6.622	5.950	5.170	3.349	1.729	0.856
$D(TS)$	50.006	36.900	17.139	15.488	15.096	14.879
$UPQ(SACCS)$	0.839	0.707	0.585	0.512	0.507	0.512
$UPQ(AS)$	0.986	0.972	0.932	0.852	0.726	0.566
$UPQ(TS)$	0.998	0.985	0.946	0.883	0.837	0.815

and MU's caches to maintain cache consistency; (ii) use of an ID-only state for each entry in MU's cache after a data object becomes invalidated; (iii) all valid data entries are set to the uncertain state after an MU wakes up. These key features make the proposed algorithm highly scalable and efficient. Strictly speaking, SACCS is a hybrid of stateful and stateless algorithms. However, unlike stateful algorithms, SACCS maintains one flag bit for each data object in MSS to determine when to broadcast IRs. On the other hand, unlike existing synchronous stateless approaches, SACCS does not require periodic broadcast of IRs. Hence SACCS greatly reduces IR messages that need to be sent through the downlink broadcast channel. SACCS inherits the positive features from both stateful and statless algorithms. Comprehensive simulation results show that the proposed algorithm has significantly better performance than TS and AS schemes.

An LRU based cache replacement is used in this paper. Further work will include studying the impact of different replacement algorithms on the performance of SACCS. Future study will also investigate the MSS cache management algorithm and the effective transfer cached data objects among MSSs when MUs roam among different MSSs.

REFERENCES

[1] G. Forman and J. Zahorjan, "The challenge of mobile computing", *IEEE Computer*, 27(6), pp38-47, April 1994.

[2] D. Barbara and T. Imielinksi," Sleeper and Workaholics: caching strategy in mobile environments ", *In Proceedings of the ACM SIGMOD Conference on Management of Data*, pp1-12, 1994.

[3] J. Jing, A. Elmagarmid, A. Heal, and R. Alonso. "Bit-sequences: an adaptive cache invalidation method in mobile client/server environments", *Mobile Networks and Applications*, pp 115-127, 1997.

[4] Q. Hu and D.K. Lee, "Cache algorithms based on adaptive invalidation reports for mobile environments", *Cluster Computing*, pp 39-50, 1998.

[5] K.L. Wu, P.S. Yu and M.S. Chen, "Energy-efficient caching for wireless mobile computing", *In 20th International Conference on Data Engineering*, pp 336-345, 1996

[6] G. Cao, "A scalable low-latency cache invalidation strategy for mobile environments", *ACM Intl. Conf. on Computing and Networking (Mobicom)*, pp200-209, August, 2001

[7] K. Tan, J. Cai and B. Ooi, "An evaluation of cache invalidation strategies in wireless environments", *IEEE Trans. on Parallel and Distributed System*, 12(8), pp789-897, 2001

[8] A. Kahol, S. Khurana, S.K.S. Gupta and P.K. Srimani," A strategy to manage cache consistency in a distributed mobile wireless environment", *IEEE Trans. on Parallel and Distributed System*, 12(7), pp 686-700, 2001.

[9] H. Yu, L. Breslau and S. Shenker, "A scalable web cache consistency architecture", *In Proceedings of the ACM SIGCOMM*, August, 1999.

[10] J. Gwertzman and M. Seltzer, "World-Wide Web cache consistency", *In Proceedings of The USENIX Symposium on Internet Technologies and Systems*, December, 1997.

[11] L. Breslau, P. Cao, J. Fan, G. Phillips and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications, ", *Proceedings of IEEE INFOCOM'99*,pp126-134, 1999