An Optimal Cache Replacement Policy for Wireless Data Dissemination under Cache Consistency*

Jianliang Xu
HK University of Science and Technology
Clear Water Bay, HK
xujl@cs.ust.hk

Qinglong Hu
IBM Almaden Research Center
San Jose, CA, USA
qhu@us.ibm.com

Wang-Chien Lee
Verizon Laboratories Inc.
Waltham, MA, USA
wang-chien.lee@verizon.com

Dik Lun Lee
HK University of Science and Technology
Clear Water Bay, HK
dlee@cs.ust.hk

Abstract

A good cache management method for mobile wireless environments has to handle problems associated with limited client resources and frequent client disconnections, in addition to standard problems found in wired environments, such as variable data sizes and data updates. In this paper, we propose a gain-based cache replacement policy, Min-SAUD, *for wireless data dissemination when cache consistency must be enforced before a cached item is used.* Min-SAUD *considers several factors that affect cache performance, namely access probability, update frequency, data size, retrieval delay, and cache validation cost.* Min-SAUD *is* optimal *in terms of the* stretch *performance measure. Preliminary experimental results show that in most cases the* Min-SAUD *replacement policy substantially outperforms two existing policies, namely* LRU *and* SAIU.

1 Introduction

Mobile wireless environments suffer from various constraints such as scarce bandwidth and limited client resources. Client data caching has been considered a good solution for coping with the inefficiency of wireless data dissemination [1, 3]. In this paper, we study the replacement problem for client data caching.

Cache replacement policies for wireless data dissemination were first studied in the *broadcast disks* project [1], where the $\mathcal{PIX}$ policy was proposed. In $\mathcal{PIX}$, the cached data item with the minimum value of $access\_probability/broadcast\_frequency$ was evicted for cache replacement. Caching and its relation with the broadcast schedule was empirically investigated in [8]. [12] proposed an optimal memory update policy that minimizes the expected latency over all data items under a given broadcast schedule. However, these previous studies assumed that data items had the same size and ignored data updates and client disconnections. Little work has investigated cache replacement policies in a realistic wireless environment where updates, disconnections, and variable data sizes are common.

In a preliminary study [13], we developed a cache replacement policy, *SAIU*, for wireless on-demand broadcast. *SAIU* took into consideration four factors that affect cache performance, i.e., access probability, update frequency, retrieval delay, and data size. However, an optimal formula for determining the best cached item(s) to replace based on these factors was not given in the preliminary study. Also, the influence of the cache consistency requirement was not considered in *SAIU*.

In this paper, we propose an optimal cache replacement policy, called *Min-SAUD*, which accounts for the cost of ensuring *cache consistency* before each cached item is used. We argue that cache consistency must be required since it is crucial for many applications such as financial transactions, and that a cache replacement policy must take into consideration the cache validation delay caused by the underlying cache invalidation scheme. In addition, *Min-SAUD* considers access probability, update frequency, retrieval delay, and data size in developing the gain function which determines the cached item(s) to be replaced.

The analytical study that we perform shows that *Min-SAUD* has the optimal performance in terms of the *stretch*

*This work was supported by grants from the Research Grant Council of Hong Kong (Grant numbers HKUST-6077/97E and HKUST-6241/00E).

performance measure under the standard assumptions of the *independent reference model* [5] and Poisson arrivals of data accesses and updates. The adoption of the independent reference model makes sense, because it reflects the access behavior on the web as shown in [4]. On the other hand, Poisson arrivals are usually used to model data access and update processes [7].

To evaluate the performance of the *Min-SAUD* policy, we conduct simulation experiments in the scenario of on-demand broadcast, where the analytical assumptions are relaxed. In the simulation experiments, we compare *Min-SAUD* to two cache replacement policies, i.e., *LRU* and *SAIU* [13]. Preliminary results show that *Min-SAUD* achieves the best performance under different workloads. In particular, the performance improvement of *Min-SAUD* over the other schemes becomes more pronounced when the cache validation delay is significant. This indicates that cache validation cost plays an important role in cache replacement policies.

The rest of this paper is organized as follows. Section 2 describes the system architecture and the performance metric used in this study. The cache replacement policy, *Min-SAUD*, and its optimality analysis and implementation issues are presented in Section 3. Section 4 introduces the simulation model for performance evaluation. The simulation results are presented in Section 5. Finally, Section 6 concludes the paper.

2 System Architecture and Performance Metric

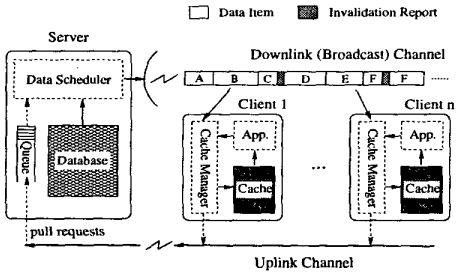


Figure 1. A Generic System Architecture

In this section, we give a brief description of the system architecture and the performance metric adopted in this paper. Figure 1 depicts a generic architecture of the wireless data dissemination systems studied. We assume that the system employs on-demand broadcast for data dissemination. That is, the clients send pull requests to the server through the uplink channel. In response, the server disseminates the requested data items to the clients through the broadcast channel based on a scheduling algorithm [2]. The clients retrieve the items of their interest off the air by monitoring the broadcast channel.

Push-based broadcast is a common alternative to on-demand broadcast for wireless data dissemination [1]. In push-based broadcast, a fixed set of data is periodically broadcast based on precompiled data access patterns. In fact, push-based broadcast can be seen as a special case of on-demand broadcast, where uplink cost is zero and data scheduling is based on the aggregate access patterns. Consequently, the result presented in this paper can also be applied to push-based broadcast.

As illustrated, there is a cache management mechanism in a client. Whenever an application issues a query, the local cache manager first checks whether the desired data item is in the cache. If it is a cache hit, the cache manager still needs to validate the consistency of the cache-hit item with the master copy at the server. Thus, it retrieves the next invalidation report (see below for details) from the broadcast channel to validate the item. If the item is verified as being up-to-date, it is returned to the application immediately. If it is a cache hit but the value is obsolete or it is a cache miss, the cache manager sends a pull request to the server in order to schedule broadcast of the desired data. When the requested data item arrives in the wireless channel, the cache manager keeps it in the cache and answers the query. The issue of cache replacement arises when the free cache space is insufficient to accommodate a data item to be cached.

Techniques based on *Invalidation Report (IR)* have been proposed to address the cache consistency issues [3, 6]. Interleaved with the broadcast data, *IR*'s are periodically disseminated on the broadcast channel. An *IR* consists of the server's updating history up to w broadcast intervals (w can be a constant or a variable). Every mobile client maintains the timestamp T_l of the last cache validation. Thus, upon reception of an *IR*, a client checks to see whether its T_l is within the coverage of the *IR* received. If yes, the client starts the invalidation process in accordance with the *IR* type. Otherwise, it drops the cache contents entirely (w is a constant) [3] or ignores the *IR* and sends its T_l to the server in order to enlarge w of the next *IR* (w is a variable) [6].

As in a previous study [13], in this paper we use *stretch* to evaluate the performance of cache replacement policies:

> **Stretch** [2]: the ratio of the access latency of a request to its *service time*, where service time is defined as the ratio of the requested item's size to the broadcast bandwidth.

Compared with access latency, which does not count the difference in data size/service time, it is believed that stretch is a more reasonable metric for items with variable sizes.

3 The Proposed Optimal Cache Replacement Algorithm

Cache replacement policy plays a central role in the cache management. Traditional cache replacement policies (e.g., *LRU*), while suitable for cached items with the same size and miss penalty, do not perform well in wireless data dissemination [13]. In the following, we first introduce a new gain-based cache replacement policy, *Min-SAUD*. Then, we show that the proposed policy results in the optimal access performance in terms of *stretch*. Finally, we address some of the implementation issues.

3.1 The Min-SAUD Replacement Policy

In this subsection, a gain-based cache replacement policy, *Minimum Stretch integrated with Access rates, Update frequencies, and cache validation Delay* (denoted as *Min-SAUD*), is proposed for the wireless data dissemination systems under cache consistency. To facilitate our discussion, the following notations are defined (note that these parameters are for one client only):

- D: the number of data items in the database.
- C: the size of the client cache.
- $\overline{a_i}$: mean access arrival rate of data item i.
- $\overline{u_i}$: mean update arrival rate of data item i.
- x_i: the ratio of update rate to access rate for data item i, i.e., $x_i = \overline{u_i}/\overline{a_i}$.
- p_i: access probability of data item i, $p_i = \overline{a_i}/\sum_{k=1}^{D} \overline{a_k}$.
- b_i: retrieval delay from the server (i.e., cache miss penalty) for data item i.
- s_i: size of data item i.
- v: cache validation delay, i.e., access latency of an effective invalidation report.
- d_k: the data item requested in the k-th access,[1] $d_k \in \{1, 2, \ldots, D\}$.
- C_k: the set of cached data items after the k-th access, $C_k \subseteq \{1, 2, \ldots, D\}$.
- U_k: the set of cached data items that are updated between the k-th access and the $(k+1)$-th access, $U_k \subseteq C_k$.
- V_k: the set of victims chosen to be replaced in the k-th access, $V_k \subseteq (C_{k-1} - U_{k-1})$.

The key issue for cache replacement is to determine a victim item set, V_k, when the free space in the client cache is insufficient to accommodate the incoming data item in the k-th access. In [13], we have observed that a cache replacement policy should choose the data items with low access probability, short data retrieval delay, high update frequency, and large data size for replacement. As described in the introduction, a cache replacement policy should also take into account the cost of cache validation. Thus, in *Min-SAUD*, a gain function incorporating these factors is defined for each cached item i:[2]

$$gain(i) = \frac{p_i}{s_i}\left(\frac{b_i}{1+x_i} - v\right). \tag{1}$$

The idea is to maximize the total gain for the data items kept in the cache. Thus, to find space for the k-th accessed data item, the *Min-SAUD* policy identifies the optimal victim item set V_k^*, $V_k^* \subseteq (C_{k-1} - U_{k-1})$, such that

$$V_k^* = arg \min_{V_k \subseteq (C_{k-1} - U_{k-1})} \sum_{i \in V_k} gain(i) \tag{2}$$

$$s.t. \sum_{i \in V_k} s_i \geq \sum_{j \in (C_{k-1} - U_{k-1})} s_j + s_{d_k} - C. \tag{3}$$

It is easy to see that *Min-SAUD* reduces to $\mathcal{PIX}$ when *Bdisk* is used and data items have equal size and are read-only, since under this circumstance the data retrieval delay of an item is the inverse of its broadcast frequency [1]. Therefore, *Min-SAUD* can be considered as a generalization of $\mathcal{PIX}$.

3.2 Analysis of the Min-SAUD Policy

In this section, we show that *Min-SAUD* is an optimal policy in terms of stretch. The *independent reference model* [5] is assumed in the analysis. To facilitate our analysis, we assume that the arrivals of data accesses and updates for data item i follow the Poisson processes. Specifically, t_i^a and t_i^u, the inter-arrival times for data accesses and updates of data item i, follow exponential distributions with means of $\overline{a_i}$ and $\overline{u_i}$, respectively. In other words, the density functions for t_i^a and t_i^u are $f(t_i^a) = \overline{a_i} e^{-\overline{a_i} t_i^a}$ and $g(t_i^u) = \overline{u_i} e^{-\overline{u_i} t_i^u}$, respectively. Under the data consistency requirement, even when the cache is hit by a query, the client still needs to wait for an effective *IR* to validate the cached copy. The chance that the data item is updated during this waiting period is very slim. Thus, to simplify our analysis, we assume that all the cache hits would be validated by the next effective *IR* as up-to-date. Finally, we assume that the cache access latency is zero since it is negligible comparing to the latency of access to the server.

The following theorem shows that *Min-SAUD* is an optimal cache replacement policy.

[1] The client accesses are assumed to be numbered sequentially.

[2] The gain function was derived while trying to prove the optimality of an initially defined heuristic gain function for cache replacement.

Theorem 1 *The replacement policy* Min-SAUD *gives better access cost, in terms of stretch, than any other replacement policy.*

Proof: To save space the proof is omitted, interested reader is referred to [14] for details. □

3.3 Implementation Issues

In this subsection, we address three critical implementation issues, namely *heap management*, *estimate of running parameters*, and *maintenance of cached item attributes*, for the *Min-SAUD* policy.

3.3.1 Heap Management

In *Min-SAUD*, the optimization problem defined by Equations (2) and (3) is essentially the $0/1$ knapsack problem, which is known to be *NP-hard*. Thus, a well-known heuristic for the knapsack problem is adopted to find a sub-optimal solution for *Min-SAUD*:[3]

> Throw out the cached data item i with the minimum $\frac{gain(i)}{s_i}$ value until the free cache space is sufficient to accommodate the incoming item.

This heuristic can obtain the optimal solution when the data sizes are relatively small compared to the cache size [11].

A (binary) min-heap data structure is used to implement the *Min-SAUD* policy. The key field for the heap is the $gain(i)/s_i$ value for each cached data item i. When the events of cache replacement occur, the root item of the heap is deleted. This operation is repeated until sufficient space is obtained for the incoming data item. Let N denote the number of cached items and M the victim set size. Every deletion operation has a complexity of $O(logN)$. An insertion operation also has an $O(logN)$ complexity. Thus, the time complexity for every cache replacement operation is $O(MlogN)$. In addition, when an item's $gain(i)/s_i$ value is updated, its position in the heap needs to be adjusted. The time complexity for every adjustment operation is $O(logN)$.

3.3.2 Estimate of Running Parameters

Several parameters are involved in computation of the $gain(i)$ function. Among these parameters, p_i is proportional to $\overline{a_i}$; s_i can be obtained when item i arrives; v is a system parameter. In most cases, $\overline{u_i}$, b_i, and $\overline{a_i}$, are not available to the clients. Thus, we need methods to estimate these values.

[3] We do not distinguish *Min-SAUD* and this heuristic in the rest of this paper.

As in [13], a well-known exponential aging method is used to estimate $\overline{u_i}$ and b_i at the server-side. They are piggybacked to the clients when data item i is delivered; the timestamp of the last update on item i, t_i^{lu}, is also piggybacked so that the client can continue to update $\overline{u_i}$ based on the received *IR*'s. The client caches the data item as well as its $\overline{u_i}$, t_i^{lu}, and b_i values. The maintenance of these parameters (along with some other parameters) will be discussed in the next subsection.

Different clients may have different access patterns, while some of their data accesses are answered by the cache. It is difficult for the server to know the real access pattern for each client. Consequently, access arrival rate $\overline{a_i}$ is estimated at the client-side. The following two estimate methods are used in our simulation:

- **Exponential Aging (EA):** When a query for item i is issued by a client, if there is no information about item i maintained in the cache, $\overline{a_i}$ is set to 0 and t_i^{la} is set to the current time; otherwise, $\overline{a_i}$ is updated using the following formula:

$$\overline{a_i} = \alpha_a/(t^c - t_i^{la}) + (1 - \alpha_a) \cdot \overline{a_i}, \tag{4}$$

  where t^c is the current time, t_i^{la} is the timestamp of the last access to item i, and α_a is a constant factor used to weight the most recent access for the running access frequency estimate.

- **Counting Average (CA):** The exponential aging method might not be accurate since it does not age the access frequency for the time period since the last access. An alternative method is to estimate average access rate for each item since it was cached. Whenever the parameter information of item i enters the cache for the first time, $count_i$ is set to 0 and t_i^{bt} is set to the current time. Afterwards, $count_i$ is increased by 1 for each new query for item i, and $\overline{a_i}$ is calculated using the following formula:

$$\overline{a_i} = count_i/(t^c - t_i^{bt}). \tag{5}$$

3.3.3 Maintenance of Cached Item Attributes

To realize the *Min-SAUD* policy, a total of six parameters, namely $s_i, \overline{u_i}, t_i^{lu}, b_i$, and $\overline{a_i}$, t_i^{la} for *EA* (or $count_i$, t_i^{bt} for *CA*), are needed for each cached data item. We refer to them as the *cached item attributes* (or simply *attributes*). Similar to [9], we employ a heuristic to maintain the cached item attributes. The attributes for the currently cached data items are kept in the cache. Let GIS_{min} be the minimum $gain(i)/s_i$ value for the currently cached data items. For those data items that are not cached, we only retain the attributes for an item j whose $gain(j)/s_j$ is larger than

GIS_{min}. This heuristic is adaptive to the cache size. If the cache size is large, it can accommodate more data items and hence a relatively small GIS_{min} value. As a result, attributes for more data items can be retained in the cache. On the other hand, if the cache size is small, fewer data items are contained and the GIS_{min} value is relatively large, thus less attributes are kept.

4 Simulation Model

The simulation model used for performance evaluation is similar to that used in [13]. It is implemented using *CSIM* [10]. A single cell environment is considered. The model consists of a single server and $NumClient$ clients. On-demand broadcast is employed for wireless data dissemination.

The default system parameter settings are given in Table 1. The database is a collection of $DatabaseSize$ data items and is partitioned into disjointed regions, each with $RegionSize$ items. Data item sizes vary from s_{min} to s_{max} and have the following two types of distributions:

- **INCRT**: $size_i = s_{min} + \frac{(i-1)(s_{max}-s_{min}+1)}{DatabaseSize}$;
- **DECRT**: $size_i = s_{max} - \frac{(i-1)(s_{max}-s_{min}+1)}{DatabaseSize}$.

Combined with the skewed access pattern, *INCRT* and *DECRT* represent clients' preference for frequently querying smaller items and larger items, respectively (further see Section 4.1).

IR's are broadcast periodically on the broadcast channel with an interval of $BroadcastInt$. The broadcast channel has a bandwidth of $BroadcastBW$. It works in a *preempt-resume* manner with *IR*'s having the highest broadcast priority and the rest of messages having equal priority. This strategy ensures that *IR*'s can always reach the clients in time [6]. The uplink channel has a bandwidth of $UplinkBW$.

Parameter	Setting
$NumClient$	200
$DatabaseSize$	2,000 data items
$RegionSize$	20 data items
s_{min}	1KB
s_{max}	100KB
$BroadcastBW$	115 Kbps
$UplinkBW$	14.4 Kbps
$BroadcastInt$	20 seconds
$ConMsgSize$	512 bytes

Table 1. Default System Parameter Settings

Parameter	Setting
$ThinkTime$	10 seconds
p	0.1
$CacheSizeRatio$	5%
$ParaSize$	4 bytes each
$DiscTime$	200 seconds
θ	0.80

Table 2. Default Client Parameter Settings

Parameter	Setting
$UpdateTime$	80 seconds
$Update\text{-}Cold/Update\text{-}Hot$	80/20

Table 3. Default Server Parameter Settings

4.1 Client Model

Each client is simulated by a process and runs a continuous loop that generates a stream of queries. After the current query is finished, the client waits for a period of $ThinkTime$ and then makes the next query request.[4] When a client is in the *thinking* state, it has probability p to enter the disconnected state every IR broadcast interval. The time that a client is in a disconnected state follows exponential distribution with a mean of $DiscTime$. Each client has a cache of $CacheSize$ size, which is a $CacheSizeRatio$ ratio of $DatabaseSize$. Cache size is defined as $CacheSize = \frac{1}{2} \times (s_{max} + s_{min} - 1) \times DatabaseSize \times CacheSizeRatio$. In order to maintain fairness to the different caching schemes, the $CacheSize$ parameter includes both the space needed for storing item attributes and the space available for storing data. Each cached parameter occupies $ParaSize$ bytes.

The client access pattern follows a $Zipf$ distribution with parameter θ [15]. The data items are sorted such that the item 0 is the most frequently accessed, and the item $DatabaseSize - 1$ is the least frequently accessed. In other words, with an *INCRT* size setting, the clients access the smallest item most frequently; with a *DECRT* size setting, the clients access the largest item most frequently. $Zipf$ distributions are frequently used to model non-uniform access patterns. The probability of accessing any item within a database region is uniform and the $Zipf$ distribution is applied to these regions. Table 2 summarizes the default client parameter settings.

4.2 Server Model

The server is modeled by a single process. Table 3 gives the server parameter settings. The clients' requests

[4] Since one query is initiated after the completion of the last query, the access arrivals do not follow the Possion process. Thus, the analytical assumptions are relaxed in the simulation.

are buffered at the server, and an infinite queue buffer is assumed. After broadcasting the current item, the server chooses an outstanding request from the buffer as the next candidate, according to the scheduling algorithm used. Compared with queueing delay and data transmission delay, the overheads of scheduling and request processing at the server are negligible. Therefore, they are not considered in the model.

Data updates are generated by the server process with an exponentially distributed update inter-arrival time with a mean of $UpdateTime$. A $Cold/Hot$ update pattern is assumed in the simulation model. Specifically, the uniform distribution is applied to all the database regions. Within a region, $Update\text{-}Cold\%$ of the updates are for the first $Update\text{-}Hot\%$ items, and $Update\text{-}Hot\%$ of the updates are for the rest. For example, we assume in the experiments that, within a region, 80% of the updates occur on the first 20% of data items (i.e., update-hot items) and 20% of the updates occur on the remaining 80% of data items (i.e., update-cold items).

5 Performance Evaluation

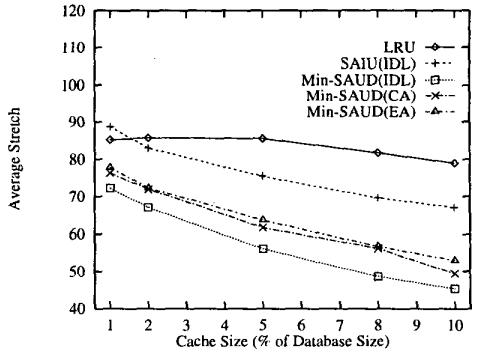
This section shows the preliminary experimental results for the performance of the *Min-SAUD* policy. *Average stretch* is the performance metric employed in this paper. In the experiments, we employ *LTSF* [2] as the on-demand broadcast scheduling algorithm and *AAW_AT* [6] as the cache invalidation scheme, since they demonstrated superior performance over other schemes [2, 6]. In *LTSF*, the data item with the largest total current stretches is chosen for next broadcast, where the current stretch of a pending request is the ratio of the time the request has been in the system to its service time. In *AAW_AT*, the updating history window w and content organization of the next *IR* are dynamically decided based on the system workload.

The results are obtained when the system has reached a stable state; i.e., each client has issued at least 5,000 queries after its cache is full, so that the warm-up effects of the client cache and the broadcast channel are eliminated. For the exponential aging estimate method, we set $\alpha = 0.25$ [1, 11]. Unless it is mentioned explicitly, the broadcast bandwidth is fully utilized.

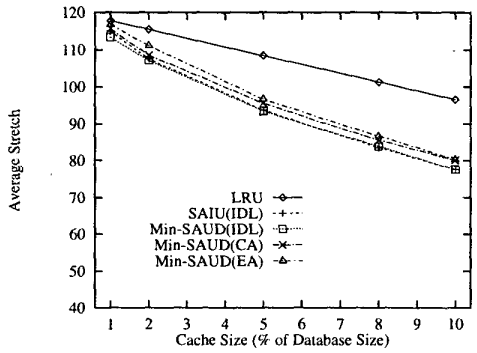
The *LRU* policy is included as a yardstick in the performance evaluation. We also compare *Min-SAUD* to *SAIU*, which makes use of a cost function of $\frac{b_i \cdot \bar{a}_i}{s_i \cdot \bar{u}_i}$ to determine the victims [13].

5.1 Experiment #1: Impact of Cache Size

This subsection investigates the performance of the cache replacement schemes under different cache sizes. The simulation results are shown in Figure 2. To estimate



(a) INCRT



(b) DECRT

Figure 2. Performance under Various Cache Sizes

data access rates, *Min-SAUD(EA)* uses Equation (4), while *Min-SAUD(CA)* uses Equation (5). *Min-SAUD(IDL)* and *SAIU(IDL)* are assumed to have perfect knowledge of data access and update frequencies.

In Figure 2, it is obvious that *Min-SAUD* achieves the best stretch performance. On average, the improvement of *Min-SAUD(IDL)* over *LRU* for *INCRT* and *DECRT* is 30.7% and 11.8%, respectively; and the improvement of *Min-SAUD(IDL)* over *SAIU(IDL)* for *INCRT* is 24.6%. *Min-SAUD(IDL)* and *SAIU(IDL)* have a similar performance for *DECRT*. The improvement for *INCRT* is greater than that for *DECRT*. This can be explained as follows. First, since *Min-SAUD* takes into consideration data size, it caches more frequently accessed items in an *INCRT* size setting, whereas it has to balance between caching more items and caching more frequently accessed items in a *DECRT* size setting. Thus, *Min-SAUD* outperforms *LRU* to a greater extent for *INCRT* than *DECRT*. Second, because the influence of cache validation delay on the stretch performance is more significant for *INCRT* than *DECRT* (see next subsection for

details), *Min-SAUD*, which takes into consideration cache validation delay, improves the performance more greatly for *INCRT*. As the cache size increases, the improvement of *Min-SAUD* over *LRU* and *SAIU* becomes more significant (for example, for *INCRT*, from 15.3% to 42.1% over *LRU* and from 18.6% to 32.3% over *SAIU*). This implies that *Min-SAUD* can utilize the cache room more effectively.

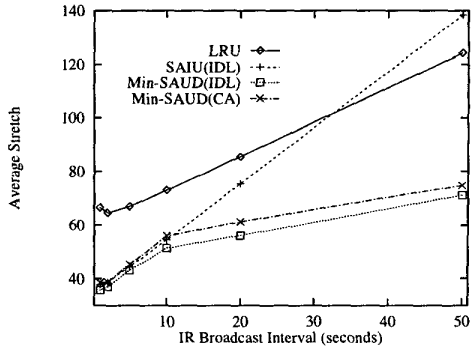
Because the system parameters (i.e., access and update frequencies) in *Min-SAUD(CA)* and *Min-SAUD(EA)* are only estimates, *Min-SAUD(CA)* and *Min-SAUD(EA)* perform slightly worse than, but pretty close to, *Min-SAUD(IDL)*. Moreover, they outperforms, in most cases, *SAIU(IDL)* which has perfect knowledge of data access and update frequencies. *Min-SAUD(CA)* demonstrates better performance than *Min-SAUD(EA)*. This suggests that *CA* estimates the access frequencies more accurately than *EA*. Therefore, in the following subsection we evaluate *Min-SAUD(CA)* only.

5.2 Experiment #2: Impact of Cache Validation Delay

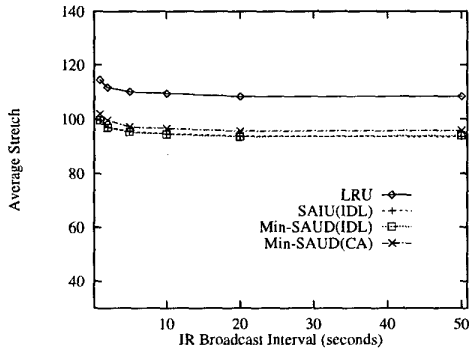
As pointed out in the previous sections, cache validation cost can have a great impact on the performance of a cache replacement policy. Such influence is investigated by experiments in this subsection. Figure 3 illustrates the performance when the *IR* broadcast interval is varied from 1 second to 50 seconds. Note that the larger the *IR* broadcast interval, the longer the cache validation delay.

Compare *INCRT* and *DECRT* size settings in Figure 3; the influence of cache validation delay on stretch is much more significant for *INCRT* than *DECRT*. The reason is as follows. When most of the queries are hit in the cache, the access latency is dominated by cache validation delay. In contrast, when the query hit ratio is low, the access latency is dominated by data broadcast speed. In other words, the higher the query hit ratio, the more dominant the cache validation delay in the performance. Furthermore, for a certain cache validation delay, the influence on stretch is more significant for smaller data items (with shorter service times). As a result, since *INCRT* has more (smaller) cached data items than *DECRT*, the cache validation delay has more impact on stretch for *INCRT* than *DECRT*.

When different cache replacement policies are considered, *Min-SAUD* performs the best in all cases. Compared with *SAIU*, *Min-SAUD* adapts to different *IR* broadcast intervals much better. For example, in an *INCRT* size setting the performance of *Min-SAUD* degrades 98% when the *IR* broadcast interval is increased from 1 second to 50 seconds, whereas the stretch of *SAIU* degrades 265%. This convinces us of the need to integrate the cost of cache validation in a cache replacement policy.



(a) INCRT



(b) DECRT

Figure 3. Performance under Different IR Broadcast Intervals

6 Conclusion

In this paper, we have investigated the cache replacement issue in a realistic wireless data dissemination environment. We relieved the restrictions on data size, data update, and client disconnection made in most of the previous work, making the proposed scheme possible for practical use. Different from the existing work, we take into account the cost of cache validation in the design of a cache replacement policy under cache consistency. An optimal gain-based cache replacement policy, *Min-SAUD*, which incorporates various factors, such as data item size, retrieval delay, access probability, update frequency, and cache validation delay, was proposed.

We showed by analysis that *Min-SAUD* achieves the optimal stretch performance under the assumptions of the independent reference model and the Possion processes of data accesses and updates. Simulation experiments were also conducted to evaluate the performance of *Min-SAUD*. Preliminary results demonstrated that in most cases *Min-SAUD* performs substantially better than the well-known

LRU policy and the *SAIU* policy, especially when the cache validation delay is a significant system factor. *Min-SAUD(CA)*, a practical realization of the *Min-SAUD* policy, showed a similar performance to *Min-SAUD(IDL)* which has perfect knowledge of access and update frequencies.

For future work, we are carrying out more experiments to extensively evaluate the performance of the *Min-SAUD* policy. We plan to extend the cache replacement policy to a cache admission policy for client disk caching. As shown in the simulation results, there is still room to improve the parameter estimate methods. Prefetching techniques can be combined into the current scheme. It would be an interesting topic to investigate the performance of *Min-SAUD* in the web environment.

References

[1] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communications environments. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 199–210, San Jose, CA, USA, May 1995.

[2] S. Acharya and S. Muthukrishnan. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 43–54, Dallas, TX, USA, October 1998.

[3] D. Barbara and T. Imielinksi. Sleepers and workaholics: Caching strategies for mobile environments. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 1–12, Minneapolis, MN, USA, May 1994.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE INFOCOM'99*, pages 126–134, New York, NY, USA, March 1999.

[5] E. G. Coffman, Jr., and P. J. Denning. *Operating Systems Theory*. Prentice Hall, NJ, USA, 1973.

[6] Q. L. Hu and D. L. Lee. Cache algorithms based on adaptive invalidation reports for mobile environments. *Cluster Computing*, 1(1):39–48, February 1998.

[7] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, New York, NY, USA, 1991.

[8] V. Liberatore. Caching and scheduling for broadcast disk systems. Technical Report 98-71, Institute for Advanced Computer Studies, University of Maryland at College Park(UMIACS), December 1998.

[9] P. Scheuermann, J. Shim, and R. Vingralek. WATCHMAN: A data warehouse intelligent cache manager. In *Proceedings of the 22nd VLDB Conference*, pages 51–62, Mumbai, India, September 1996.

[10] H. Schwetman. *CSIM user's guide (version 18)*. MCC Corporation, http://www.mesquite.com, 1998.

[11] J. Shim, P. Scheuermann, and R. Vingralek. Proxy cache design: Algorithms, implementation and performance. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11(4):549–562, July/August 1999.

[12] L. Tassiulas and C. J. Su. Optimal memory management strategies for a mobile user in a broadcast data delivery system. *IEEE Journal on Selected Areas in Communications (JSAC)*, 15(7):1226–1238, September 1997.

[13] J. Xu, Q. L. Hu, D. L. Lee, and W.-C. Lee. SAIU: An efficient cache replacement policy for wireless on-demand broadcasts. In *Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 46–53, McLean, VA, USA, November 2000.

[14] J. Xu, Q. L. Hu, W.-C. Lee, and D. L. Lee. An optimal cache replacement policy for wireless data dissemination under cache consistency. Technical Report, Hong Kong University of Science and Technology, January 2001.

[15] G. K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, MA, USA, 1949.