International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. , No. (Mon-20..)

http://dx.doi.org/10.12785/ijcds/XXXXXX

Enhancing the Scalable Asynchronous Cache Consistency Scheme with the Clock Algorithm

Ramzi A. Haraty[1] **and Meghry G. Tchangoulian**[1]

[1]*Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon*

Received Mon. 20, Revised Mon. 20, Accepted Mon. 20, Published Mon. 20

Abstract: In recent years, mobile computing has seen significant advancements, largely driven by the development and expansion of wireless networks. This technological evolution has given rise to various models and strategies designed to enhance user experience by speeding up query responses, reducing network traffic, and optimizing the use of network resources. One notable model that exemplifies these advancements is the Scalable Asynchronous Cache Consistency Scheme (SACCS). SACCS employs a hybrid data consistency strategy to ensure the integrity of cached data, offering an alternative to traditional stateful and stateless approaches. Additionally, SACCS utilizes the Least Recently Used (LRU) replacement algorithm to manage its cached data items efficiently. In this paper, we delve into the workings of SACCS, particularly focusing on its integration with three other replacement strategies: CLOCK, Longest Distance First (LDF), and Least Frequently Used with Dynamic Aging (LFU-DA). Through comprehensive simulations, we assess the performance of these algorithms in comparison to pre-existing methods. Our findings reveal that the CLOCK replacement strategy outperforms the others, demonstrating superior efficiency in managing cached data. The implications of this study suggest that adopting the CLOCK strategy within SACCS can significantly enhance mobile computing performance, making it a valuable consideration for future developments in this field. This research contributes to the ongoing efforts to refine caching techniques and optimize network resource utilization in the ever-evolving landscape of mobile computing.

Keywords: Mobile Computing, cache consistency, replacement policy, Clock algorithm

1. INTRODUCTION

With the constant use of mobile phones and the need to access information as quickly as possible, caching can be used to satisfy the users' needs while still being in line with the constraints of the wireless network used. Nevertheless, designing a caching technique can be challenging since it needs to account for the fact that mobile phones can get disconnected from the Internet, due to poor connection or power-off, and hence the data that these caches might have would prove to be old. Thus, the caching technique should act in such a way that will ensure the validity of the data once the mobile phone gets reconnected to the Internet. To achieve this consistency, stateful and stateless approaches have been proposed in the literature. The stateful approach knows the content of the user's cache, whereas the stateless does not. Hence, the stateless approach must check the validity of the cache's content before answering a query every single time. This in turn translates into the stateful being more scalable than the stateless, despite having a large overhead [1]. To strike a balance between the two approaches, a new method has been introduced known as "Scalable Asynchronous Cache Consistency Scheme" (SACCS) that uses the Least Recently Used (LRU) algorithm to perform the replacement [1]. In this paper, we test out three other replacement strategies: Clock[2], Longest Distance First(LDF) [3] and Least Frequently Used with Dynamic Aging (LFUDA) [4] and compare them with some of the older strategies that have been used with SACCS [5]. This paper is divided into five sections. In the second section, we provide a literature review mentioning some other cache replacement policies and the different techniques that have been used for mobile cache consistency. In the third section, we give an overview of SACCS and in the fourth section, we introduce the three new replacement algorithms. In the fifth section, we provide the experimental results and discuss our findings. Finally, we conclude our work and suggest future directions.

2. Literature Review

The basic architecture of a wireless mobile computing consists of original servers connected to mobile support stations (MSS) via a wired network, and mobile users (MU) connected wirelessly to these stations as illustrated in figure 1 below [1]. The MUs have local caches that store within them data, and they communicate with the MSS, via the uplink channel, to retrieve the data that they have not stored in their caches. In case the MSS does not have the data in its cache, or the data needs to be updated, the MSS communicates with the original server, retrieves the updated/not found data and broadcasts it to the MUS via the downlink channel.

It is to be noted that the MUs can get disconnected and then get reconnected to the MSS, leading to the rise of data inconsistencies. Therefore, the main goal here is finding the right data consistency strategy to ensure that the updated version of the data is in the MUs caches.

E-mail: rharaty@lau.edu.lb, meghry.tchangoulian@gmail.com

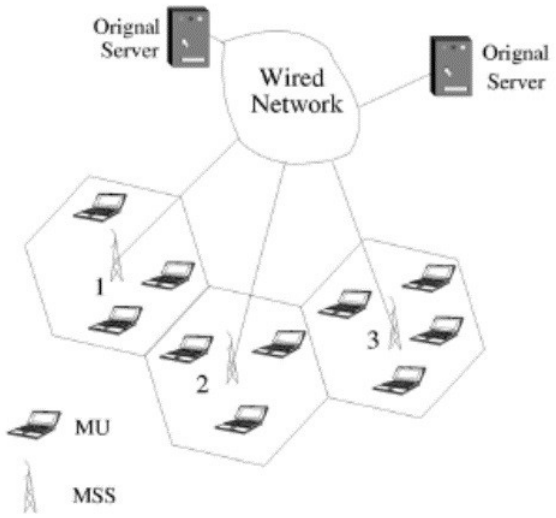

Figure 1. Mobile Computing Environment (Wang,2004)

A. Caching in Mobile Computing

Caching is a method that stores copies of the data in a local storage. This leads to decreasing the distance to access the content, which in turn speeds up data retrieval and decreases the bandwidth consumption [6]. However, the main issue with caching is figuring out what data to cache for the cache to be efficient i.e. having the requested data in the cache most of the time [1]. Another concern with caching is ensuring the data consistency among the different components of the distributed system since mobile units can go offline unexpectedly. For these reasons, several cache replacement strategies and models have been introduced and explored in the literature.

B. Cache Replacement Strategies

The cache has limited space, and when it becomes full, a replacement strategy needs to be used to get the requested data item and place it in the cache. These replacement strategies can be based on different factors such as frequency of requests, recency, and size to name a few. Therefore, there are different replacement strategies in the literature that can be used to do this [7].

1) Temporal Based Strategies

These strategies rely on the fact that users will most likely access the same data again, so that data will remain within the cache [7], [8]. Examples of such strategies are: Least Recently Used (LRU), Most Recently Used (MRU) [8], CLOCK [2] and Adaptive Replacement Cache (ARC) [9].

The ARC replacement algorithm is a mix of LRU and LFU, such that the cache is divided into two parts: T1 and T2. T1 is going to be used for the recently referenced data and T2 is going to be used for the frequently used data [9]. In addition, the cache has two lists B1 and B2 that keep track of the cache items that have been evicted from the cache. This algorithm adjusts the two partitions dynamically by checking in which of the lists B1 and B2 the hit has occurred. If the hit is in B1, then T1 increases in size by removing a data item from T2 and vice versa.

2) Location Based Strategies

These strategies rely on the fact that users will most likely access the same data again when they are still close to the same server. Examples of such policies are: Manhattan Distance Based Policy (MDBP) [10], Farthest Away Replacement (FAR) [11], Mobility Aware Replacement (MARS) [12] and Predicted Region Based Replacement Policy (PRRP) [13].

The MDBP [10] replaces the item with the highest Manhattan distance between the MU location and the location of the data in the cache, whereas FAR [11] replaces the data items that are found in the out-direction portion of the cache first as opposed to the in-direction since they are farther away from the server. As for the MARS [12] and the PBR [13] policies, they remove the data item that has the lowest cost. The former is calculated by the taking into consideration the location of the MU and the frequency of access of data while the latter takes into consideration the size of the data, the access probability, the location of the MU and the location to which the MU will move to.

3) Function Value Based Strategies

These strategies rely on a pre-defined function to calculate a specific value given certain parameters [14]. Depending on the algorithm, the calculated value is either used for rearranging the elements before replacement, or for the replacement of the element that has the lowest value. Examples of such algorithms include: Least Utility Value (LUV) [15], Stretch Access-rate Inverse Update frequency (SAIU) [16], Least Valuable First(LVF) [17] and Least Relative Value (LRV) [18]. The above-mentioned policies are similar in the sense that the cached item is replaced if the value attained from the calculation of the function is the lowest when the cache is full. However, they differ from each other mainly in terms of parameters used in the functions. For instance, the LUV [15] policy considers the reference probability and the cost of retrieving the data item whereas LVF [17] considers delay, frequency, and age of the data to calculate the function. As for SAIU [16], it considers the reference probability, the data size, the relative delay and the update frequency while LRV [18] considers the access probability and the gain value from removing an item from the cache when calculating the function.

4) Machine Learning Based Strategies

These strategies depend on machine learning algorithms and are divided into supervised learning [19], [20], [21], [22], unsupervised learning [23], [24], [25] and reinforcement learning [26], [27], [28], [29]. Most of the machine learning strategies mentioned above deal with caching data items that are the most popular.

- Supervised Machine Learning Based Strategies: Supervised learning uses labeled data, where the inputs and outputs are known, to build or train models based on the relationships found in the data. An example is in [19] where they introduce the Liquid State Machine (LSM) algorithm to increase the data requests' predictions for unnamed aerial vehicles (UAVs). Furthermore, there are several other approaches that focus on predicting, and later caching, popular data items such as those in [20], [21], [22].

- Unsupervised Learning Based Strategies: Unsupervised learning uses unlabeled data, where a model is built to detect patterns or cluster data into groups based on similarities. In [24] and in [23], the authors used k-means as their unsupervised machine learning strategy. In the former, they detected patterns in the data requests and used k-NN to cluster and cache the data in Ultra Dense Networks (UDNs). In the latter, they introduced the Cluster-Based Content Caching (CBCC) algorithm where they used features of the content such as access history and labels to calculate the expected popularity and compared it with the least popular data item in the cache, replacing it if the calculated items were more popular. They proved that their method is more efficient than the other replacement algorithms such as LFU, FIFO, LRU and LFUDA. As for [25], the authors introduced a new caching algorithm known as Content Cache Value and User Activity (CCVUA) for D2D edge caching. First, they cluster users upon their physical and social characteristics by using a spectral clustering method and match these clusters with differing base stations. Then, they use the CCVUA that considers both the value of the cache and the behavior of the user to cache the items.

- Reinforcement Learning Based Strategies: Reinforcement learning uses an agent to find the optimal results where the agent takes different actions, and receives rewards based on those actions. Thus, after training for a while it will learn to take the best actions to maximize the rewards. For example, the authors in [28] introduced the Grouped Linear Model (GLM) that uses the history of previous requests to predict future data requests and uses an RL approach with Model-Free Acceleration (RLMA) to replace the cache items. In [26], the authors proposed a new algorithm that uses a contextual multi-armed method to perform proactive caching while in [27] the authors introduced a new method known as change point detection with reinforcement learning (CPRL) that notices changes in the environment and adapts the caching strategy to optimally fit these changes. Whereas in [29], the authors introduced a new caching method, divided into three phases, based on reinforcement learning that utilizes the traffic of the requests as well as the size of the caches used.

For more information, please refer to [29].

C. Data Consistency Strategies

One of the main challenges of caching is ensuring the data consistency among the different elements of the mobile network. Several stateful [30], [31], [32], [33], [34], [35] stateless [30], [36], [37], [38], [39] and hybrid[1], [40], [41] data consistency strategies have been introduced in the literature to tackle this issue. Considering the stateful technique, the server is aware of the data items cached in the MUs' systems. This means that, despite the large overhead, the validity of the data does not need to be checked every time a query is made. As for the stateless approach, it is a technique where the server is unaware of the state of the data items cached in the MUs systems. Here, the cache needs to check the validity of the cache's content before answering a query every single time, making it less scalable. Therefore, the hybrid data consistency scheme is introduced that combines the main benefits of the two mentioned techniques. One of them is called SACCS that is described in the section below.

3. The Scalable Asynchronous Cache Consistency Scheme

To overcome the shortcomings of the stateless and stateful approaches of maintaining the data consistencies in the mobile environment, [1] proposed a new approach known as (SACCS) for read systems that uses LRU for the replacement algorithm. The idea is that SACCS keeps track of some of the states' information, specifically the MSS keeps track of what data items might be valid in the MUs cache. Hence, the database management is made simpler than that of the stateful approach that keeps track of all the data items in the MUs caches with their states. In addition, SACCS does not periodically broadcast the invalidation reports (IR) unlike the stateless techniques, which in turn reduces the IR messages that are being sent via the downlink channel.

Regarding the consistency of the databases between the MSS and the Server, it is kept via wired network consistency algorithms. As for the consistency between the MSS and the MU caches, it is maintained by associating each data item with the flag bit. The flag bit is changed whenever the data item is changed on the server to indicate that a valid data may be found in the MU's cache when it gets the data. This flag is reset only whenever the MSS gets an updated data item and sends the IR message to the MUs. Furthermore, the mobile user can either be in one of these two states: awake or sleep. Awake indicates that the MU is online and is connected to the Internet, whereas sleep indicates that the MU is disconnected due to a power off or a network connection issue. Therefore, if the MU receives an IR message when awake, then the data item is invalidated, and its state is changed to ID-only. However, if the MU is asleep, the data items are unaffected until the MU wakes up. In this case, when the MU wakes up, all the cached data items are set to an uncertain state. Furthermore,

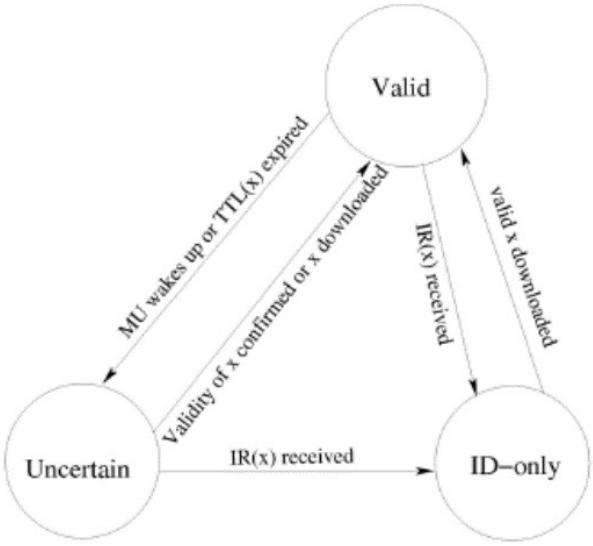

Figure 2. State Transition Diagram (Wang, 2004)

the data is "Valid" when it is found in the cache and its Time To Live (TTL) matches the TTL of its copy found in the MSS. Whereas the data is "Uncertain" when the data is in the cache of a MU that got reconnected after a disconnection, thus its state is no longer certain and needs to be checked. As for the "ID-Only", it is for the data that has been invalidated due to an IR received by the MSS. This process is further illustrated in figure 2.

As mentioned before, SACCS uses LRU as the cache replacement algorithm. This means that whenever a data item is added to the cache or is accessed (hit), that data item is added and or moved to the head of the cache. However, when the cache is full, data entries that are valid, invalid, and uncertain are removed from the end of the cache to make room for the new data item. As for validating cache data items, the entries with ID-Only and Uncertain states are kept in their same location, and data from the tail is deleted if the cache is full.

4. Cache Replacement Strategies for SACCS

A. Longest Distance First

The longest Distance First (LDF) is a replacement algorithm that replaces the data that has the longest distance from the current data entry [3]. Here, the page that is next to the current page in anti-clockwise rotation is replaced in case two data items have the same distance. As for the distance calculation, the idea is to place the unique data entries in a circular list and see how far the current entry is from the other entries present in the cache in clockwise and anticlockwise directions, where the smaller of the two is chosen. Out of all the distances calculated for the data entries, the one that is the greatest is replaced. When it comes to SACCS, we have the function Find LDF() that finds the data entry that has the longest distance from the current item. Therefore, the data item that has longest distance is placed at the tail of the cache to be replaced when the cache is full.

B. Clock

The Clock algorithm, an approximation of the LRU algorithm, replaces data based on the flag bit and the clock hand [2]. It keeps a circular list of all the data items in the cache, where each entry has a flag, set to 0 or 1, and a 'clock hand' that moves forward after each data insertion. Whenever the cache is full, the flag bit pointed to by the clock hand will be checked. If the flag bit is set to 0, then the data will be replaced, and the clock hand will move forward. However, if the flag bit is set to 1, then the flag bit will be set to 0 and the clock hand will move forward till it finds a data entry whose flag bit is 0 so that it can replace it. When it comes to SACCS, the new data item is added to the front of the list with its flag bit set to 1, and the clock hand is updated by moving one step backwards. We are moving backwards since we are adding items to the head of the cache that we have. If the data item has been referenced, then the flag bit is set to 1. However, if the cache is full and the flag bit of the data item pointed to by the clock is 0, then the data item is replaced. If the bit is 1, then the clock moves backwards until it finds a data item with its flag bit set to 0.

C. Least Frequently Used with Dynamic Aging

The Least Frequently Used with Dynamic Aging (LFU-DA)[4], [42] is an approach that is used to deal with the issue of cache pollution caused by LFU. We have cache pollution in LFU since it would keep highly requested data items in the cache for long periods of time even if they are not being accessed currently. Hence, by adding an aging factor to the frequency of accesses, it leads to these old items being removed from the cache eventually. The idea is to have a global variable Cache-Age, set to 0 initially, that is added to the frequency (F) of the cached data item, where the key $K_{(i)}$ is: $K_{(i)} = F_{(i)} + cache\ \ age$

When a cache is full, the data item that has the least key value is replaced, where the cache-age is set to the key value. In case two or more key values are the same, the data item that has been least recently used is removed. Furthermore, every time a data item is accessed or added to the cache, the key value is calculated again following the formula above. When it comes to SACCS, the new data item is added to the front of the list with the cache-age set to 0 initially. Every time a data item is accessed or added to the cache later on, its key value is updated. When the cache is full, the data item that has the least key value is removed, with the cache-age being set to its key value.

D. Other Cache Replacement Algorithms

We have compared the above-mentioned approaches with the previous work that has been done on other cache replacement algorithms to evaluate their performance[5].

1) Least Recently Used

The least recently used (LRU) algorithm is a replacement algorithm that replaces the least recently used item when the cache is full. It is the algorithm that has been initially proposed to be used with SACCS, where the least recently used items are found at the tail of the cache. When the cache is full, the data items at the tail are marked for removal.

2) First in First Out

The first in first out (FIFO) algorithm is a replacement algorithm that replaces the first item that has been cached. When it comes to SACCS, the first items that have been cached are at the end of the cache, and hence replaced because newly cached items are added to the head of the cache.

3) Most Recently Used

The most recently used (MRU) algorithm is a replacement algorithm that replaces the item that has been most recently used. When it comes to SACCS, the item that has been recently accessed is placed at the tail of the cache to be replaced when the cache is full.

4) Most Frequently Used

The most frequently used (MFU) algorithm is a replacement algorithm that replaces the item that has been most frequently used. When it comes to SACCS, every data item would have an access number that is incremented every time the item is accessed. The item that has been the most frequently used is placed at the tail of the cache to be replaced when the cache is full.

5) Least Frequently Used

The least frequently used (LFU) algorithm is a replacement algorithm that replaces the item that has been least frequently used. When it comes to SACCS, every data item would have an access number that is incremented every time the item is accessed. The item that has been the least frequently used is placed at the tail of the cache to be replaced when the cache is full.

5. Experimental Results

To evaluate the performance of the three suggested replacement algorithms with SACCS, they are implemented and compared to some of the previous algorithms that have been used with SACCS. To perform the simulation, the environment is set up with fixed and changeable parameters using the C++ programming language. We have used a one cell slot that has 100 mobile users, each with a 300-cache size, a thousand data entries that have different access commands, a random data size (in bytes) and an average data update interval (in sec) that is changeable. Furthermore, the downlink and uplink data transmission use one channel with a bandwidth of 1250 bps, where the message size is 64 bytes. Concerning the mobile units, they can be in either sleep or wakeup state that is randomized using the two-state Markov chain. The simulation is performed by varying

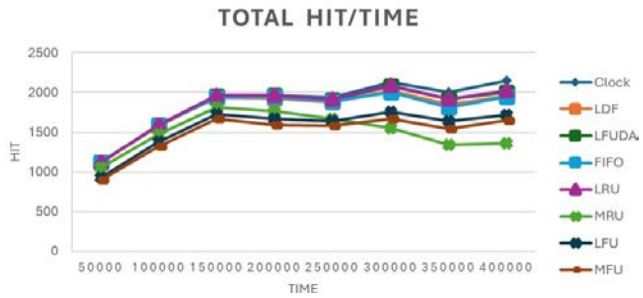

Figure 3. Total Hit Vs Time

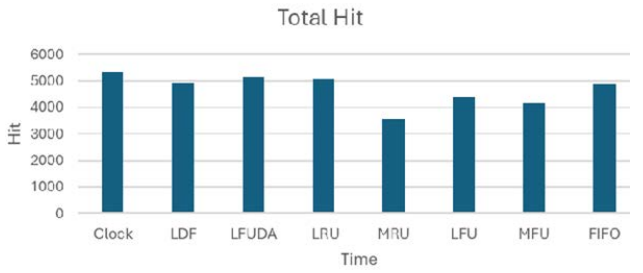

Figure 4. Total Hit

eight time slots. As for the performance metrics, we have used the total hit/time, total miss/time, total hit, total miss, miss ratio, total delay, average delay, bytes per query and downloaded bytes per query.

The total hit is the total number of times the data item queried is found in the MU's cache. Figure 3 below shows the total hit results versus the specific time while Figure 4 shows the total hit for each replacement algorithm.

As one can see from the results above, the CLOCK algorithm has the best total hit (5333) as opposed to the other tested algorithms. As for LRU (5086) it is the second best followed by the LFUDA (5052) and LDF (4963). The CLOCK algorithm has the best total hit due to setting the access bit to one every time a new data is added to the cache as opposed to only setting it to one during re-access, giving every data item an equal head start. Furthermore, the CLOCK hand moves, after filling the cache, only when the cache is full. Thus, when removing a data item, the one that is added first or that has been added first and not been referenced for a while will most likely be removed. Concerning LFUDA, it is close to LRU since both algorithms make use of the temporal locality, where the latter is replacing based on recency and the former based on frequency and recency. Furthermore, if we examine LFUDA, we note that it takes into consideration both the frequency and the recency for the replacement, thus it does not allow old items that have high frequencies to remain in the cache longer than necessary. As for LDF, it is shown to be better than FIFO since instead of relying on removing the item that has been added first, it removes the data item that is the farthest from the current data item. This in turn

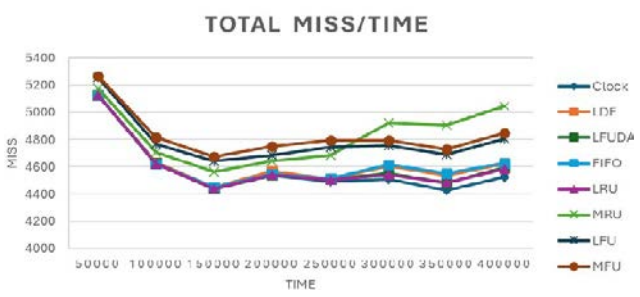

Figure 5. Total Miss Vs Time

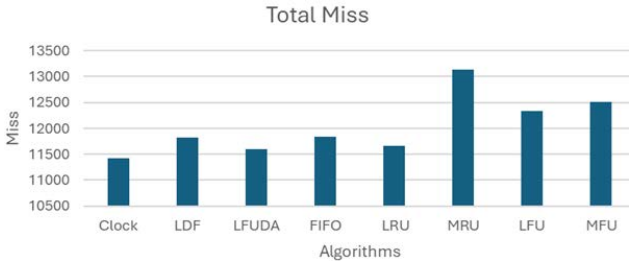

Figure 6. Total Miss

mimics prediction, where data items that are the farthest from each other, based on their initial accesses, are less likely to be accessed again.

As for the total miss, it is the total number of times the data item is invalid or not found in the MUs cache. Figure 5 below shows the total miss results versus the specific time on the different caching algorithms while Figure 6 shows the total miss for each replacement algorithm. Based on these two figures, the CLOCK (11,421) has the least total misses as opposed to the LFUDA (11598), LRU (11,666), LDF (11,816) and the other algorithms.

The total miss ratio is the ratio of all the number of times the data is not found in the cache over all the data items that are requested. Figure 7 below shows the total miss ratio results versus the specific time after running the simulation on the different caching algorithms. Based on this figure, the CLOCK has the least miss ratio as opposed to LFUDA followed by LRU, LDF and the other tested algorithms.

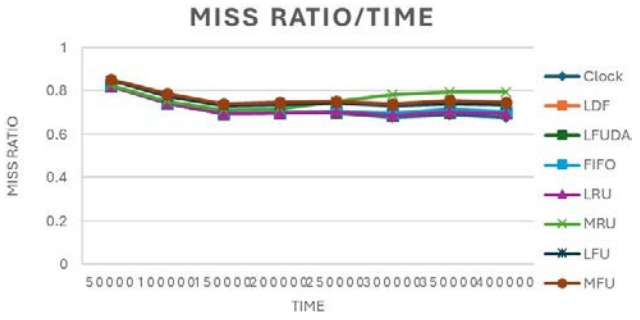

Figure 7. Miss Ratio Vs Time

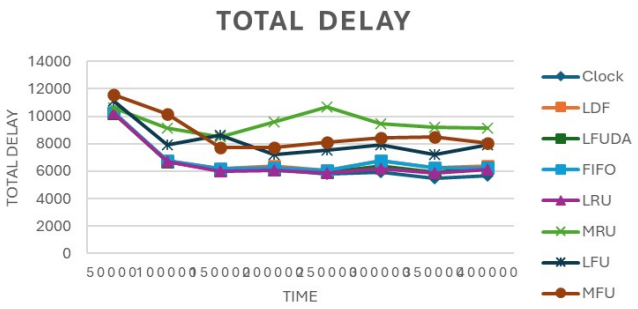

Figure 8. Total Delay

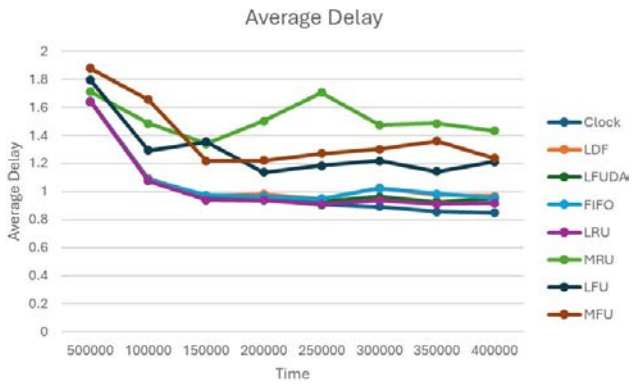

Figure 9. Average Delay

As for the total delay, based on the figure 8 above, the CLOCK has the least total delay (5652.57) followed by the LFUDA (5921.56), LRU (6121.15), FIFO (6275.44), LDF (6360.79) and the other tested algorithms. This is due to the fact that delay measures the time between a request and its receival, hence if the requested data items are in the cache most of the time, requesting missed data items would be low.

Concerning the average delay, based on the figure 9 above, between issuing a data request and receiving that request on average. below shows the average delay VS time after running the simulation on the different caching algorithms. Based on figure 9, the CLOCK (0.847589) has the least average delay as opposed to LFUDA (0.894496), LRU (0.916117), FIFO (0.954294), LDF (0.966686) and the other tested algorithms.

As for the bytes/query, Figure 10 below shows the bytes per query after running the simulation on the different caching algorithms. Based on the figure below, the CLOCK has the least bytes/query (754.433) as opposed to LFUDA (788.321), LRU (793.541), LDF(830.943) and the other tested algorithms.

The data downloaded/query shows the number of bytes downloaded per query. Figure 11 below shows the data downloaded/query after running the simulation on the different caching algorithms. Based on the figure below, the clock

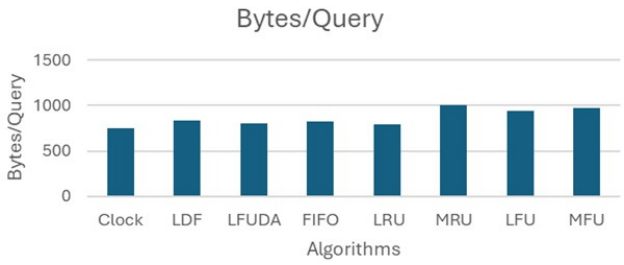

Figure 10. Bytes Per Query

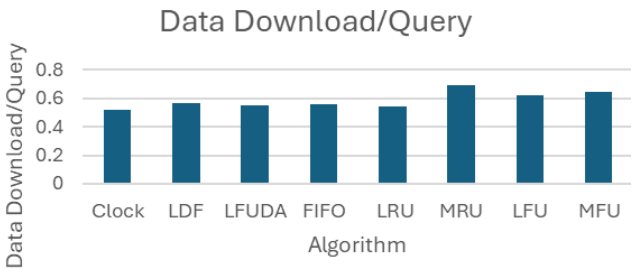

Figure 11. Data Download Per Query

has the least data download/query (0.515428) as opposed to LFUDA (0.539196) LRU (0.541938), LDF (0.562619) and the other tested algorithms. The low ratios are attributed to finding the data items in the cache, since data is downloaded only when there is a miss or there is an update.

Based on the results above, the proposed CLOCK algorithm seems to be the best in comparison to the other tested algorithms. This can be attributed to the fact that the Clock has the most hits with the least misses. Furthermore, having the most hits translates to having the least total and average delay, since the algorithm would spend less time getting data from the MSS as most of the data would be found in their caches. The same reasoning could be applied as to why the data downloaded per query is the least in the CLOCK algorithm. As for LFUDA, it is the second best due to having the second-best hits. The reasoning behind this could be since MUs tend to request the same content repeatedly, and LFUDA replaces the data items that have been accessed least frequently dynamically thanks to the cache age.

6. Conclusion

Caching has been proven to show improvements in bandwidth utilization in wireless mobile computing. However, the major issue with caching in these networks is efficiency and validity. In the former, we have stated efficiency since we need to ensure that the requested data items are found in the cache most of the time. As for the validity of the data, mobile devices are prone to disconnections from the network due to power-offs or bad network connections. For these reasons, several cache replacement algorithms and data consistency schemes have been studied in the literature. As mentioned, several cache replacement strategies have been explored in the literature that are temporal, location or function based, each with their improvements and drawbacks. The temporal based algorithms depend on the fact that data recently accessed will be accessed again, while the location-based algorithms depend on the fact that data items placed closer to the server will be accessed again. As for the function based, these algorithms depend on a value obtained from a function to make the cache replacement. In addition to these methods, several Machine learning-based algorithms have been proposed that mostly rely on learning to cache data items that are highly requested. Concerning the validity of data, several stateful, stateless and hybrid consistency approaches have been developed to ensure the validity of the cached items. Both the stateful and stateless approaches have drawbacks, where the former has a large overhead in order to keep track of the MUs states while the latter is redundant and not scalable due to sending IRs periodically. Hence, the aim of the hybrid method such as SACCS is to utilize the benefits of the stateful and stateless approaches while avoiding their drawbacks. In this paper, we have used SACCS with other replacement strategies such as CLOCK, LDF and LFUDA, and compared them with other replacement strategies. We note that the CLOCK algorithm is the most efficient out of LRU, LFUDA, LDF, FIFO, MRU, MFU and LFU since it has the highest hit rate and lowest bandwidth utilization with LFUDA coming at a close second. Despite this, it would be interesting to evaluate these algorithms after performing network analysis. Another future direction could be using machine learning models with SACCS to better cache data items at the MSS as well as the MUS.

References

[1] Z. Wang, S. Das, H. Che, and M. Kumar, “A scalable asynchronous cache consistency scheme (saccs) for mobile environments,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 11, pp. 983–995, 2004.

[2] F. J. Corbato, *A paging experiment with the multics system*. Massachusetts Institute of Technology, 1968.

[3] G. Kumar and P. Tomar, “A novel longest distance first page replacement algorithm,” *Indian Journal of Science and Technology*, vol. 10, no. 30, pp. 1–6, 2017.

[4] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin, “Evaluating content management techniques for web proxy caches,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 4, pp. 3–11, 2000.

[5] R. A. Haraty and L. Turk, “A comparative study of replacement algorithms used in the scalable asynchronous cache consistency scheme.” in *CAINE*, 2006, pp. 83–88.

[6] C. Wang, Y. He, F. R. Yu, Q. Chen, and L. Tang, “Integration of networking, caching, and computing in wireless systems: A survey, some research issues, and challenges,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 7–38, 2017.

[7] E. Hattab and S. Qawasmeh, “A survey of replacement policies for mobile web caching,” in *2015 International Conference on*

Developments of E-Systems Engineering (DeSE). IEEE, 2015, pp. 41–46.

[8] E. J. O'neil, P. E. O'neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," *Acm Sigmod Record*, vol. 22, no. 2, pp. 297–306, 1993.

[9] N. Megiddo and D. S. Modha, "{ARC}: A {Self-Tuning}, low overhead replacement cache," in *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, 2003.

[10] S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava, M. Tan *et al.*, "Semantic data caching and replacement," in *VLDB*, vol. 96, 1996, pp. 330–341.

[11] P. Venketesh and R. Venkatesan, "A survey on applications of neural networks and evolutionary techniques in web caching," *IETE Technical review*, vol. 26, no. 3, pp. 171–180, 2009.

[12] K. Y. Lai, Z. Tari, and P. Bertok, "Mobility-aware cache replacement for users of location-dependent services," in *29th Annual IEEE International Conference on Local Computer Networks*. IEEE, 2004, pp. 50–58.

[13] A. Kumar, M. Misra, and A. K. Sarje, "A predicted region based cache replacement policy for location dependent data in mobile environment," in *2006 International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 2006, pp. 1–4.

[14] H. ElAarag and S. Romano, "Comparison of function based web proxy cache replacement strategies," *2009 International Symposium on Performance Evaluation of Computer & Telecommunication Systems*, vol. 41, pp. 252–259, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:14518810

[15] N. Chand, R. C. Joshi, and M. Misra, "Cooperative caching in mobile ad hoc networks based on data utility," *Mobile Information Systems*, vol. 3, no. 1, pp. 19–37, 2007.

[16] J. Xu, Q. Hu, D. L. Lee, and W.-C. Lee, "Saiu: An efficient cache replacement policy for wireless on-demand broadcasts," in *Proceedings of the ninth international conference on Information and knowledge management*, 2000, pp. 46–53.

[17] F. M. Al-Turjman, A. E. Al-Fagih, and H. S. Hassanein, "A value-based cache replacement approach for information-centric networks," in *38th Annual IEEE Conference on Local Computer Networks-Workshops*. IEEE, 2013, pp. 874–881.

[18] L. Rizzo and L. Vicisano, "Replacement policies for a proxy cache," *IEEE/ACM Transactions on networking*, vol. 8, no. 2, pp. 158–170, 2000.

[19] M. Chen, W. Saad, and C. Yin, "Liquid state machine learning for resource and cache management in lte-u unmanned aerial vehicle (uav) networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1504–1517, 2019.

[20] M. T. Firouzjaee, K. Jamshidi, N. Moghim, and S. Shetty, "User preference-aware content caching strategy for video delivery in cache-enabled iot networks," *Computer Networks*, vol. 240, p. 110142, 2024.

ing in fog-ran," in *2017 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2017, pp. 1–6.

[22] S. S. Tanzil, W. Hoiles, and V. Krishnamurthy, "Adaptive scheme for caching youtube content in a cellular network: Machine learning approach," *Ieee Access*, vol. 5, pp. 5870–5881, 2017.

[23] B. Jia, R. Li, C. Wang, C. Qiu, and X. Wang, "Cluster-based content caching driven by popularity prediction," *CCF Transactions on High Performance Computing*, vol. 4, no. 3, pp. 357–366, 2022.

[24] G. Shen, L. Pei, P. Zhiwen, L. Nan, and Y. Xiaohu, "Machine learning based small cell cache strategy for ultra dense networks," in *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2017, pp. 1–6.

[25] Y. Zhang, W. Zhang, H. Hao, and K. Zhang, "Cluster caching strategy based on user characteristics in edge networks," in *2023 24st Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2023, pp. 36–41.

[26] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, 2017.

[27] J. Rostampoor, R. S. Adve, A. Afana, and Y. A. E. Ahmed, "Cprl: Change point detection and reinforcement learning to optimize cache placement strategies," *IEEE Transactions on Communica- tions*, vol. 72, no. 4, pp. 2339–2353, 2024.

[28] N. Zhang, K. Zheng, and M. Tao, "Using grouped linear prediction and accelerated reinforcement learning for online content caching," *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:3865918

[29] X. Zhang, G. Zheng, S. Lambotharan, M. R. Nakhai, and K.-K. Wong, "A reinforcement learning-based user-assisted caching strategy for dynamic content library in small cell networks," *IEEE Transactions on Communications*, vol. 68, no. 6, pp. 3627–3639, 2020.

[30] D. Barbará and T. Imieliński, "Sleepers and workaholics: caching strategies in mobile environments," *SIGMOD Rec.*, vol. 23, no. 2, p. 1–12, may 1994. [Online]. Available: https://doi.org/10.1145/191843.191844

[31] G. Cao, "On improving the performance of cache invalidation in mobile environments," *Mob. Netw. Appl.*, vol. 7, no. 4, p. 291–303, aug 2002. [Online]. Available: https://doi.org/10.1023/A:1015463328335

[32] N. Chand, R. Joshi, and M. Misra, "Energy efficient cache invalidation in wireless mobile environment," in *2005 IEEE International Conference on Personal Wireless Communications, 2005. ICPWC 2005.*, 2005, pp. 244–248.

[33] W. He, I.-R. Chen, and B. Gu, "A proxy-based integrated cache consistency and mobility management scheme for mobile ip systems," in *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, 2007, pp. 354–361.

[34] A. Madhukar, T. Özyer, and R. Alhajj, "Dynamic cache

[21] Y. Jiang, M. Ma, M. Bennis, F. Zheng, and X. You, "A novel caching policy with content popularity prediction and user preference learning-invalidation scheme for wireless mobile environments," *Wireless Networks*, vol. 15, pp. 727–740, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:40914157

[35] T. T. M. Nguyen and T. T. B. Dong, "An adaptive cache consistency strategy in a disconnected mobile wireless network," in *2011 IEEE International Conference on Computer Science and Automation Engineering*, vol. 4, 2011, pp. 256–260.

[36] J. Jing, A. Elmagarmid, A. S. Helal, and R. Alonso, "Bit-sequences: an adaptive cache invalidation method in mobile client/server environments," *Mob. Netw. Appl.*, vol. 2, no. 2, p. 115–127, oct 1997. [Online]. Available: https://doi.org/10.1023/A:1013616213333

[37] H. Safa, H. Artail, and M. Nahhas, "A cache invalidation strategy for mobile networks," *Journal of Network and Computer Applications*, vol. 33, no. 2, pp. 168–182, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804509001210

[38] M. Sain, S. Varanasi, Y.-J. Kang, and H. J. Lee, "Aim — adaptive invalidation mechanism for wireless networks," in *2017 19th International Conference on Advanced Communication Technology (ICACT)*, 2017, pp. 621–625.

[39] R. Tiwari and N. Kumar, "An adaptive cache invalidation technique for wireless environments," *Telecommunication Systems: Modelling, Analysis, Design and Management*, vol. 62, no. 1, pp. 149–165, May 2016. [Online]. Available: https://ideas.repec.org/a/spr/telsys/v62y2016i1d10.1007 s11235-015-0070-1.html

[40] M. Choi, W. Park, and Y.-K. Kim, "A hybrid cache cohrency scheme for ubiquitous mobile clients," in *2007 International Conference on Convergence Information Technology (ICCIT 2007)*, 2007, pp. 181–188.

[41] Y. Bao, R. Alhajj, and K. Barker, "Hybrid cache invalidation schemes in mobile environments," in *The IEEE/ACS International Conference onPervasive Services, 2004. ICPS 2004. Proceedings.*, 2004, pp. 209–218.

[42] W. Ali, S. M. Shamsuddin *et al.*, "Intelligent dynamic aging approaches in web proxy cache replacement," *Journal of Intelligent Learning Systems and Applications*, vol. 7, no. 04, p. 117, 2015.

Author 1 Name short biography ….

Author 2 Name short biography ….

Author 3 Name short biography ….