Weak Cache Consistency Driven Data Access Schemes in Wireless Networks

by

Rama Naga Sai Srikanth Varanasi, B.E

A Thesis

In

Computer Science

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for the Degree of

MASTERS OF SCIENCES

Approved

Dr. Sunho Lim

Chair of Committee

Dr. Michael Shin

Mark Sheridan
Dean of the Graduate School

May, 2016

ACKNOWLEDGEMENTS

As I begin to turn a new chapter in my life, there are countless people to whom I must express deepest gratitude for showing me the path and gateway to accomplish this huge goal. Foremost, I am very grateful towards my research advisor Dr. Sunho Lim for giving me an opportunity to work under his mentorship. His valuable advice and guidance always brought me back to reality. Although the reality might not always be what I wanted to hear, I knew I can count on your kind selves to help me realize my full potential.

I would also like to thank my parents, Sridhar and Madhavi Varanasi, to whom I dedicate this dissertation. From a young child, they have been nothing but nurturing, supportive, caring, and most of all, loving. There is no way I could have advanced without their constant love and support; to them, I will be forever indebted. I must also acknowledge and thank my other family members who have been supporting and given inspiration when my research was on a downward slope. I would also like to thank my friends for providing the needed Human element. And, of course, I cannot forget my confidant, Pranathi. She has kept me grounded, motivated me, and has a special place in my heart.

I want to give encouragement to those about to embark on this journey, it is not easy, but when you reach the final page of chapter five, youll know youve accomplished something huge. Finally, I would also like to extend my sincere thanks to Dr. Michael Shin in accepting to be my committee member and parting knowledge through his courses.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii
ABSTRACT v
List of Figures vi
1. INTRODUCTION 1
1.1 Classification Of Wireless Networks 1
1.2 Transmission Types 1
1.3 Caching 2
1.4 IR/UIR Approach 3
1.5 PER & CB Approach 5
1.6 Research Motivation 5
1.7 Organization of the Document 6
2. RELATED WORKS 7
2.1 Strong or Weak 7
2.2 Push, Pull or Hybrid 8
2.3 Stateful or statless 9
2.4 Summary 10
3. THE PROPOSED SCHEME 11
3.1 System Model 11
3.2 Consistency Sensitive Data Access Algorithms 13
3.3 The Proposed Scheme - PER-WC 14
3.3.1 Consistency Condition 14
3.3.2 Validity Window 14
3.4 Summary 15
4. EXPERIMENTS AND RESULTS 18
4.1 Simulation Testbed 18
4.2 Simulation Results 18
4.3 Consistency Changes 19
4.4 Consistency Window 20
4.5 Cache Hit Ratio 20
4.6 PER-WC Vs PER - Smart Access 21

4.7 Validity Window 21
4.8 Query Delay 22
4.9 Summary 23
5. CONCLUSION AND FUTURE WORK 24
Bibliography 25
APPENDIX 26

ABSTRACT

With the advent and advancement of mobile technology, there has been a signicant increase in accessing the Internet services and information wirelessly. One of key optimization techniques is to cache frequently accessed data items in a local cache. Invalidation report (IR) based cache invalidation and its variants have been deployed to energy efciently update the cached data items in single-hop wireless networks. Mobile nodes can access the cached data items to answer a query only after receiving an IR that is periodically broadcasted by a server. A strong consistency is implicitly assumed but it may unnecessarily increase the query delay for waiting the next IR. In addition, due to the inherent broadcast operation, the IR-based approaches may incur a non-negligible communication overhead to support nodes distributed over different wireless cells.

In this thesis, we propose a weak cache consistency driven data access scheme based on the poll-each-request (PER) cache invalidation framework, called Poll Each Request-Weak Consistency (PER-WC). Unlike prior approach, where every node has the same consistency level with the server, each node is able to set its own target consistency level independently to meet diverse consistency requirements. We design and develop a customized simulation framework to conduct our experiments using the CSIM, which is a popular development toolkit for discrete-driven simulation and modeling. We vary key simulation parameters to measure the performance, including target consistency, update interval, query interval, and window size. Our results show that the proposed scheme can reduce the query delay and adaptively adjust the consistency level.

LIST OF FIGURES

1.1 Single-Hop Network and Multi-Hop Network 2
1.2 Broadcast, Unicast and Multicast 2
1.3 Invalidation Report (IR/UIR) Approach 3
1.4 Poll Each Request 4
1.5 Call Back 4
3.1 System Model 11
3.2 The Generation of Query by Client. 12
3.3 The reception of server reply at Client 13
3.4 PER-WC 16
3.5 Validity Window 16
4.1 The performance of PER and PER-WC as a function of mean update and query intervals. 19
4.2 Adaptive Behaviour of Current Consistency 20
4.3 Change in Consistency with respect to consistency window changes) 20
4.4 PER-WC Vs PER 21
4.5 Query Delay Vs Update Interval 22
4.6 Query Delay Vs Query Interval 22

CHAPTER 1

INTRODUCTION

This is a Digital Era with 7.2 Billion smart gadgets and 2 Billion smartphones[1] exceeding human population. It just took 3 decades for such exponential development. As of 2015, the rate of growth was estimated to be 5 times as compared to the rate of increase in world population which accounts to 7 Billion[2] (2015). These facts draw our attention towards a thorough understanding of underlying technologies that drive the very existence of these devices. Proliferation of data and wireless networking standards act as a backbone for the successful running of these devices. Miniaturization, Affordability and ease-of-use have extrapolated in 3.7 Billion subscribers[1] around the globe. Enhancements in technologies such as 2G, 3G, 4G have enabled smooth integration of wireless capable devices into our lives.

1.1 Classification Of Wireless Networks

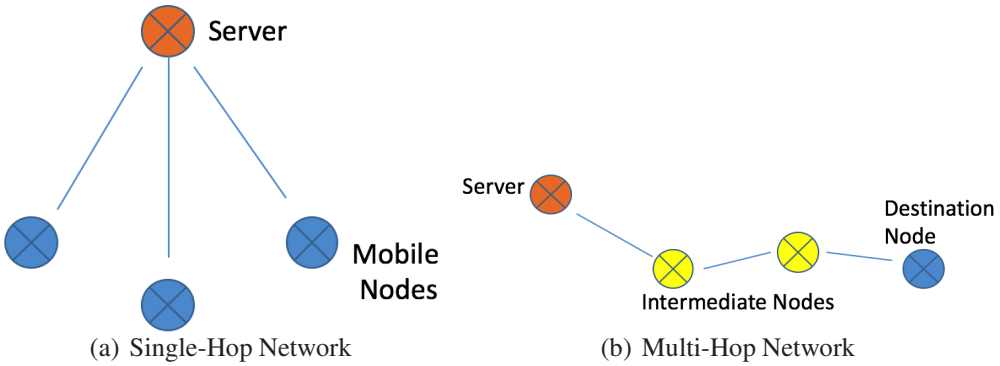
Wireless Networks are classified into 2 types namely single-hop networks and multi-hop networks. Figure 1 describes each of them in detail.

1. Single Hop Networks: Nodes are within the reach of Base Station and can communicate directly.
2. Multi-Hop Networks: Nodes are far away and cannot communicate directly; the traffic has to be forwarded by intermediate nodes

1.2 Transmission Types

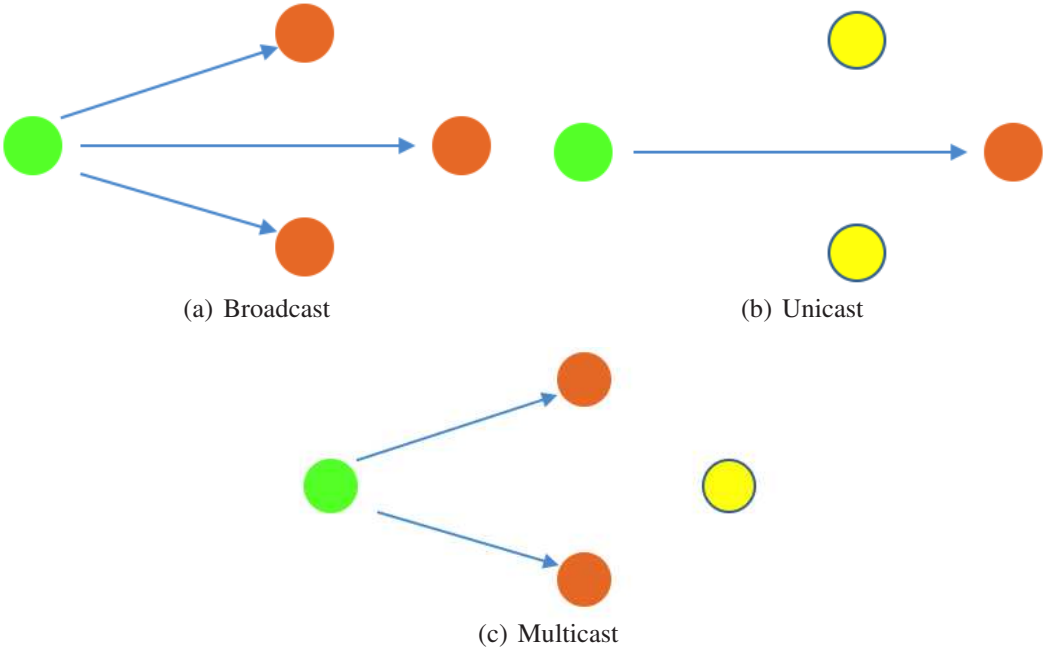
There are 3 types of transmission. They are: i) Broadcast ii) Unicast iii) Multicast

1. Broadcast: This kind of transmission happens when there is a single source which sends packets to all the nodes in the network.
2. Unicast: This kind of transmission happens when there is a single source which sends packets to single specified destination.
3. Multicast: Multicasting is the networking technique of delivering the same packet simultaneously to a group of clients.



(a) Single-Hop Network (b) Multi-Hop Network

Figure 1.1. Single-Hop Network and Multi-Hop Network



(a) Broadcast (b) Unicast

(c) Multicast

Figure 1.2. Broadcast, Unicast and Multicast

1.3 Caching

Today, there are many applications that deliver varied kinds of feeds to the user on a regular basis. These application constantly fetch data from the server and consume significant resources. In such scenarios, caching can be very useful and can optimize the data traffic, network bandwidth and the usage of local cache. And to perform caching efficiently, a node has to decide upon the following three factors. They are as follows: i) Cache Admission Policy ii) Cache Invalidation Policy iii) Cache Replacement Policy

1. Cache Admission Policy: It defines the kind of data admitted into a clients local cache.
2. Cache Invalidation Policy: After caching data items, there needs a control mechanism to periodically check the validity of the data items. This is monitored by the invalidation

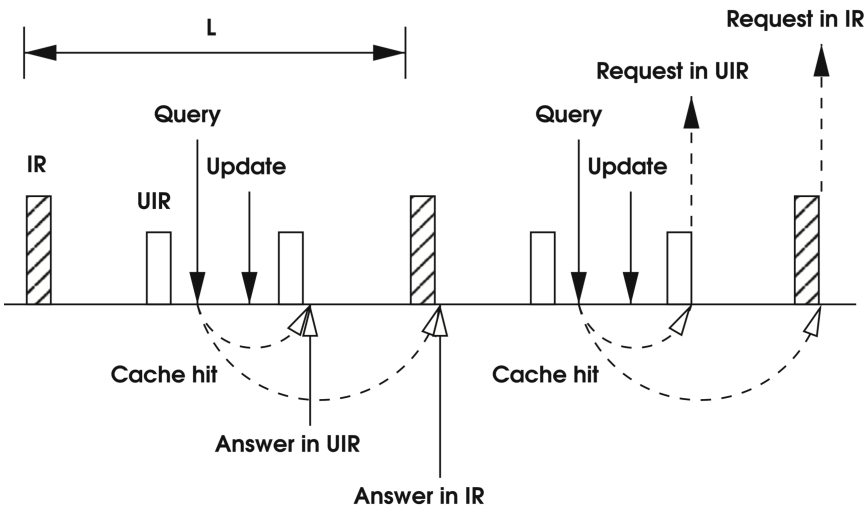


Figure 1.3. Invalidation Report (IR/UIR) Approach

policy.

3. Cache Replacement Policy: Once the cache is full, there is a need to eliminate some unused data items and create space for new items to be cached. Replacement policy manages unused data items.

For example, a node can cache frequently accessed data item so that the recurring non-negligible communication cost to request the same data item can be minimized or eliminated.

1.4 IR/UIR Approach

For a caching scheme, consistency of data items is very essential. Strong consistency denotes that the data items in clients cache are always valid with respect to server. Such high consistency is not always desired as with the time-insensitive data. Consistency level concurrently requires control. Based on the control setting, data items are either pushed by server proactively or pulled by clients. An example of pull control is clients requesting web services. While on the other hand, updating emails and messages automatically without user interaction is an example of push control. Finally, server maintains a table of mapping between clients and their requested data items. When such a mapping is provided, server tracks updates to data items and proactively sends to registered clients. Otherwise, server has no knowledge of clients in terms of data items. One such mechanism which efciently utilizes cache to answer user queries is Invalidation Report (IR). In IR approach, the servers broadcast the data periodically containing data items and their most recent timestamps in key-value pair structure. The IR also contains the update history information witnessed during the last X intervals. In

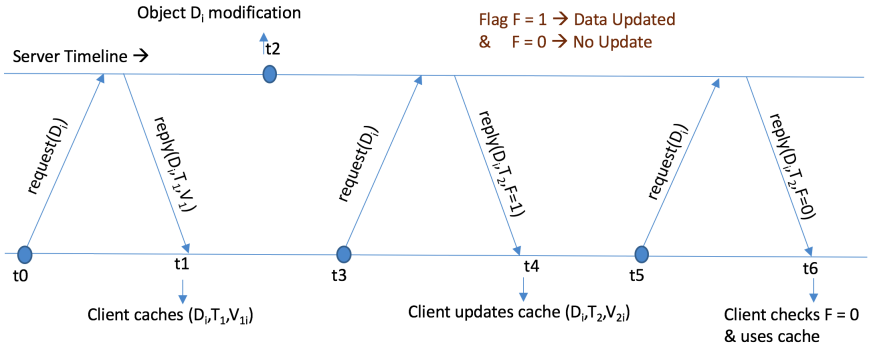


Figure 1.4. Poll Each Request

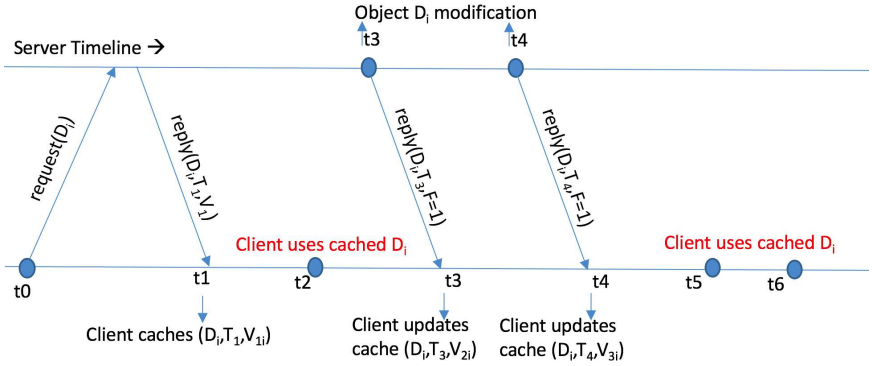


Figure 1.5. Call Back

Updated Invalidation Report (UIR) mechanism, the data items that are updated between two consecutive IRs are only broadcasted. Compared to IR, the UIR does not contain the update history. The server inserts several UIRs into each IR interval. Due to the periodic broadcast nature of IR/UIR, the nodes can operate in sleep state and save energy. In IR/UIR server is stateless, strong consistent client-initiated scheme.

In a few applications, user decisions are based on the accuracy of the feeds. If the data rendered is obsolete, it impacts the user. IR approach always maintains high consistency and the node is required to wait for the acknowledgement from the server incurring a long query delay. For example, many social networking applications are not critically update sensitive as compared to nancial apps which requires real-time update operation and results. This observation concludes that high consistency is not always mandatory and requirement of data consistency can be different for different applications. In such situations, caching data with weak consistency ensures low communication overhead.

1.5 PER & CB Approach

To overcome the limitations of broadcasting in IR/UIR, we adopted other strong consistent protocols such as Poll-Each-Read(PER) and Call-Back(CB) that are widely researched and adapted. PER is a stateless, client-initiated strong consistent scheme. In PER, clients request data as and when required. There is no server broadcast as in IR. When a client queries for a data object, the client polls server to check the validity of its cached data item. If the data is up-to-date, the server sends a confirmation message without any data object. Otherwise, the server sends a reply with the updated data item. CB is a stateful, server-initiated strong consistent invalidation procedure. When a data item at the server is modified, clients with corresponding cached data items are notified and clients invalidate them. If a client accesses for a data item with the server and if a cached copy exists, the client uses cached copy without communication overhead. In CB, clients copy is always valid. Lastly, PER can reduce wireless network traffic as compared to CB when updates to data items are frequent. On the contrary, CB reduces query latency as compared to PER.

1.6 Research Motivation

IR works effectively for a single cell environment. However as the number of cells within single Base Station(BS) increase, the effective number of IRs/cell decrease. This causes the nodes to receive inappropriate data update notifications for long periods of time. Also, every node in IR waits for a minimum of half IR interval in average to answer a query of cached data items. This incurs a long query delay and less effective ratio if cells increase. Therefore, we propose a weak consistency approach based on PER mechanism. Our contribution is as follows:-

- Developed PER-WC, an user defined consistency sensitive cache Invalidation strategy. The consistency condition is proposed which validates the number of data items cached in a node against those items at the server. This condition is used to explore current consistencies with the support of consistency window.
- Each client independently and flexibly decides on the consistency. Based on this condition, a validity window is derived which specifies the total number of cached items that can be answered without an uplink request. This approach opportunistically utilizes cached items and validates them once the validity window becomes 0, or if there is a request for new data item.
- As a consequence of having a validity window, a client can request multiple data items within a single request. Another outcome would be the close adaption of current consistency to user-defined target consistency possible due to the validity window.

- All decisions are made at client's side. Also, we integrate PER-WC with the existing PER cache invalidation frameworks. Our approach behaves as the existing; when the validity remains 0 and thus each request is sent to server. We achieved a highly scalable and no delay system in certain scenarios.

1.7 Organization of the Document

This document has been designed as follows. In Chapter 2, the research efforts in various aspects are reviewed. Different types of strong, weak, push, pull invalidation mechanisms are described and a few works related to these methods are discussed. Specifically, we focus more on the research works based on strong consistency models, as it is closely related to our application, adopted model and proposed algorithm. We also reason the importance and benefits of adaptive algorithms. In Chapter 3, we present our system model and the proposed approach in detail. In Chapter 4, we focus on the performance evaluation of the proposed approach and compare the performance of our invalidation model against other models. We give a detailed description of our analysis based on the query intervals, update intervals. For an in-depth analysis, we perform experiments on each parameter and show signs of improvement over the original PER schema. Chapter 5 summarizes the work done in the thesis and discusses the scope of improvement in the future.

CHAPTER 2

RELATED WORKS

In this chapter, we briefly review various research areas related to data access with an emphasis on the efforts related to the work presented in this thesis. As discussed in the Introduction Chapter, the purpose of a data access scheme is to develop data communication techniques between clients and server in a wireless resource constraint environment. Most of the cache invalidation strategies proposed and developed fall under the following three categories. They are (1) Co-operative based models (2) Adaptive Consistent models (3) Replacement models.

With the pervasive immersion of wireless communication, several research efforts focus on analyzing the data access and replacement schemes. In the following subsections, we go over various categories of work in the literature that each address a different aspect of this vast research area. In this chapter, we give an introduction and an insight into various approaches and then we specifically focus on three key areas: (1) Consistency level (Strong or Weak) (2) Consistency control (Push, pull and hybrid) and (3) Cache status maintenance (stateful or stateless).

2.1 Strong or Weak

Since previously studied popular techniques were PER and CB (inherently strong consistency based approaches), extensive research was conducted on them in literature. Lee et al. [3] proposed FW-DAS which focused on decreasing the data access latency and swiftly accessing the data by developing mechanism using both PER and CB. Three entities were introduced namely Node, Base Station(BS) and Application server(AS). The communication was assumed to be in a two-tier format. i) All the nodes in the network communicate with BS only (Tier-I) and ii) BS solely communicates with AS (Tier-II). Four mechanisms were developed with Tier-I and Tier-II. Each mechanism is a combination of PER-PER, PER-CB, CB-PER and CB-CB. PER-PER saves maximum energy consumption while CB-CB exhibits very low sometimes negligible latency. Finally, as the data object is always invalidated or valid in the cache, it is strong consistent.

Chen et al. [4] developed cache replacement policy based on updates. PER and CB were modified and refined to Server Based PER (SB-PER), Revised CB (R-CB) where server keeps information about all nodes. In SB-PER, the server records all activities of the client (Original PER is stateless) and maps all clients with the requested data items. This enables the server to decide cache replacements based on the updated values. While R-CB, the client requests access to update ratio of all the data items and then a decision for cache replacement policies is taken. The data item with the lowest access to update factor is evicted from the cache. Both the mechanisms are stateful and have strong consistencies.

Coming to weak consistent approaches, Lim et al.[6] proposed Consens a target consistency based approach with underlying assumption that every application doesn't require a 100% consistency. It's based on IR/UIR approach. Proposed were the following three suggestions to the existing approach. They are i) Opportunistic access, ii) Lazy request and iii) Multi-data transmission. Here, client is given a choice to select a consistency rate (Target Consistency - TC) for itself.

When the consistency condition (current consistency $>$ target consistency) is met, a node uses cached data item to answer user query. In short, it opportunistically skips contacting server for validation. In addition to the above, even if a data item is identified to be invalid, it can be used to answer queries. Lazy request was introduced to decrease the system consistency. Multi-data transmission was presented to reduce the long waiting delays by clients. As soon as the server broadcasts IR, the clients receive requested data immediately. In this way, a client can save energy and reduce delay maintaining consistency close to TC. It is a weak consistency approach as clients don't validate data items if the system performs better than expected.

2.2 Push, Pull or Hybrid

There are 3 ways of accessing data depending on which entity initiates the communication. If server initiates, then its called push based model because the server proactively sends updated data items to registered clients. And if a client initiates, then its called pull based model because the clients demanded updates as and when there is a request for one. Hybrid model functions as push and pull simultaneously. Depending on the network bandwidth availability, battery of clients and the number of nodes; various techniques can be applied.

Lee et al. [3] proposed FW-DAS. In second-tier, push mode is utilized to actively receive updates on BS aggressively and serving clients with updated information. In pull mode, the updated value of an object is notified to BS and it requests the item only on local popularity.

While in pure invalidation mode, the updated object is notified to BS and BS requests it upon client's demand only. This approach can be categorized as a hybrid approach.

In push based systems as described by Cao et al.[7], server proactively initiates cache invalidation. There is a requirement for periodic broadcasting or a control mechanism to properly push data to respective clients.

Another variation of hybrid model was studied by Kiwon et al. [8] as a part of multi-radio networks. Unlike prior approaches, client to client interactions were also considered. The requesting node gathers intelligence from the neighboring nodes. During the process both push and pull models are exercised. The main advantage of this can be observed when the energy of a node is critical and it can only spend energy for near-by communications with co-peers.

2.3 Stateful or statless

A server can be stateful(knowing) or stateless(unknowing). A stateful server keeps track of its clients. It maps each client with its cached data items and pushes updates to respective clients accordingly. On the other hand, a stateless server does not track its clients. It doesn't maintain the status of client's cache nor the updates are driven proactively.

Sung-hwa et. al [9] proposed a power aware delta-consistent stateless server approach. In this mechanism, every cached data item has a parameter delta (δ) which defines a expiry time period. If an object is requested before δ expiration, the query is immediately answered and considered valid. Or else, the client contacts the server to validate and update the data item with updated delta values.

Xu et. al [10] proposed their own stateful server approach for WMNs called CACC. It is an IR-based approach studied in co-operative and hierarchical fashion. Different and other entities were proposed to represent relationship. They are server, Gateway, Mesh Router and clients. Two new techniques were presented i.e. IR integration and IR resending. In IR integration, the Gateway buffers a group of IRs before sending to the router. This helps in IR-resending to nodes that missed original IRs. IR-resending happens only upon client requests.

Tiwari et al. [11] explored a stateless and synchronous server scheme based on sending only hot data items in IR increasing the effective cache hit ratio. It is true in general that stateless servers are scalable while stateful servers require adequate resources to become scalable and handle scalability with difficulty. In our point of view, having enough knowledge about clients will enhance server capabilities in taking better decisions about cache admission, control and

replacement policies.

2.4 Summary

Data access schemes are proposed to save bandwidth, reduce battery consumption and reduce latency and increasing data volume. In this chapter, related works were summarized. We discuss a few research efforts in these applications. An important aspect within this thesis is adaptive consistency and its effects. There has been extensive research in this area. The most familiar and important approaches are presented in this section. Hence, we present a few works on the topic of consistency level. Finally, we describe adaptive algorithms, their importance and applications in a few research works.

CHAPTER 3

THE PROPOSED SCHEME

3.1 System Model

Fig. 3.1 shows our system model. We consider a wireless network with clients having single-hop access to server. The server communicates with clients in one-to-one manner. Clients move independently in a randomized fashion. The multi-tenant server stores n items in its database (where n is the maximum). As Base Station (BS) connects to Server through the wired link, we assume the delay between them to be much lesser compared to the delay between clients and BS wirelessly. We use server and BS interchangeably as BS acts as source of information and is transparent to database server from node's point of view. Also, the server maintains a queue for client requests separately.

Every node has a cache and stores frequently requested data with it. Every cached data item has a time stamp (TS) and value associated with it. A client request is uniquely identified by the ID of the item/items requested. The BS updates as well as tracks updated data items whereas clients are only given read permissions on them. Finally, clients maintain a record of previously answered queries in a window called consistency window for inner operations.

In our approach, every node subscribes to a server with its own consistency level and accepts occasional inconsistencies. The consistency is a threshold value that is assigned by the user

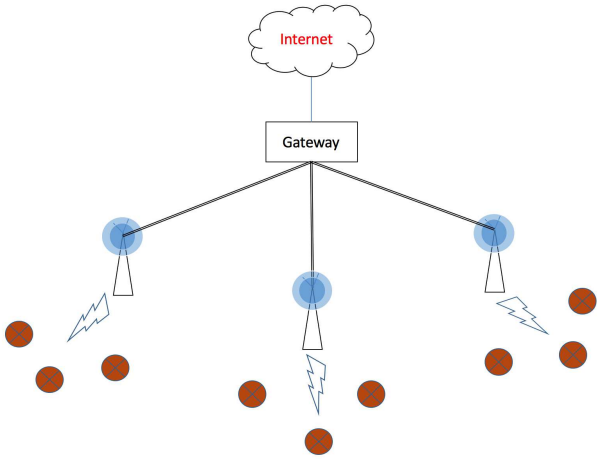


Figure 3.1. System Model

which indicates the update tightness between the source data item stored at the server and its cached local copy at the client. The consistency is considered strong and weak based on assigned threshold value which ranges between 1 and 0. The current consistency of the system is strongly influenced by the consistency window. Lastly, every node is eligible to input its threshold value as a system configuration parameter to enable consistency for various applications.

Previous research on PER emphasized on investigating the client's energy problem, to let the clients operate in sleep state when there is no request or query. It caused high query delay and long disconnection period which effected the overall performance of the system. Thus, we made a few assumptions and found that strong consistency was not always a mandatory criteria and users not getting effected critically when this assumption was relaxed. Also we believe users posses adequate knowledge on how frequently and rigorously they needed potential updates. Hence, we propose a user-defined consistency sensitive cache invalidation scheme, called PER-WC, to support diverse consistency requirements of users and applications.

Notations:

◇ d_i,t_i denote the data in cache and TOA t at node n_i ,
◇ Flag bit tracks availability of adap_window (default value = 0),
◇ C_i,Q_i denotes cache and queue (of size adap_window) for n_i
◇ T,T_{Curr} denote the target and current consistencies of n_i
◇ adaptive window is denoted by w

A. When a node, n_i, generates a query q for data item d

```
if d ∉ C_i
    if (w exists)
        Send request(d,0) and Q([d_i]) to Server;
    else
        Send request(d,0) to Server;

else if d ∈ C_i
    if (!w)
        send request(d,t) with flag = 1;
    else if (w exists)
        reply from cache;
        Queue d into Q with flag = 0;
        increment index of w;
    else if (w is full)
        Send Q([d_i]) & request(d,t) with flag = 1
```

Figure 3.2. The Generation of Query by Client.

B. When n_i, receives server reply

For Flag = 0
for each($d_i \in$ Q)
if($t_{i,c} < t_{i,s}$)
w[i] = 0 & i++;
else if(t_k==0)
use & update cache with (d_k, t_k);
else($t_{i,c} > t_{i,s}$)
w[i] = 1 & i++;
T_{curr} = hits/validity_window;
if $T_{curr} > T_{tar}$
inc_window();
else
w = 0;

For Flag = 1
if (d_k.value is empty)
validity_window[n] = 1;
T_{curr} = hits/validity_window;
if $T_{curr} > T_{tar}$
inc_window();
else
w = 0;
else if (d_k.value = "value")
use & update cache with (d_k, t_k);

Inc_Window()
Limit = No_Of_Hits/100
While (Limit $>=$ T_{tar})
No_Of_Hits $-=$ 1; New_window ++;
Limit = No_Of_Hits/100;
w = New_window ++;

Figure 3.3. The reception of server reply at Client

3.2 Consistency Sensitive Data Access Algorithms

We first discuss the operation of PER-WC in detail. Ideal scenarios is explained including setting up of validity window (W_v) and calculating current consistencies appropriately. The number of queries answered through cache is quantified by the validity window. Through consistency window (W_c), number of cache hits and/or number of misses are derived. In the proposed technique, client decides the necessary actions for themselves.

3.3 The Proposed Scheme - PER-WC

Initially a node tries to access i^{th} data item. Since it would be the first access to the item, the node has no cached value of it. It sends $request(i, 0)$ (where t=0 represents new request) and server responds with a $reply(i)$ with the new data. Similarly, a subsequent request for previously cached i^{th} data item is $request(i, t)$ where t is last accessed time. The server verifies time stamp 't' and send appropriate reply. If the requested data item is already updates at the server, then automatically sends a the updated value to the client. Otherwise, it a null value indicating the client to use its own cache. The presence or absence of the data value is denoted by flag being 1 and 0.

3.3.1 Consistency Condition

Every node defines a consistency condition either = 1 or a value less than 1. If it is set to 0.7, then 7 out of 10 queries are guaranteed to be valid answers (as adopted from [6]). Validity of a data item is verified by requesting and waiting for the server's reply indicating the current cached data item; valid. In other words, there is a Invalidation check with the server before answering certain queries. In order to achieve consistency in PER, an adaptive validity window(w_v) is proposed that controls the consistency.

3.3.2 Validity Window

Each node is able to set its own validity window. Cache utilization significantly reduces latency but data items become obsolete after certain time units. Thus, without validation from server there is a chance that cache becomes invalid and eventually the consistency reduces below the assigned target consistency. Therefore it is necessary to have server validation after certain number of cache attempts. Our adaptive validity window defines a limit on the number of opportunistic cache attempts.

To realize validity window, each node counts the number of invalid items (N_{inval}) and valid items (N_{val}) in a given consistency window (w_c). The sum of valid and invalid items constitutes our consistency window(w_c). Subsequently, the current consistency(τ_c) is derived based on the above mentioned parameters as following:

$(w_c) = (N_{inval}) + (N_{val})$ and $(\tau_c) = (N_{val}) / (w_c)$

An item is considered valid if it was cache hit , invalid if cache miss. More the valid items, more is the current consistency(τ_c). More the current consistency(τ_c), more is the validity window(w_v). The validity window is the outcome of the difference between current and target consistency(τ_t) i.e.,

$(w_v) = (\tau_c) - (\tau_t)$

The validity window (w_v) is inversely proportional to the target consistency (τ_t). If target

consistency (τ_t) reaches 1, the validity window (w_v) will become 0. This is because current consistency (τ_c) cannot be > 1 and if it is 1, the difference between current and target would be 0. This is the threshold condition where PER-WC integrates with PER and works like PER.

In Fig. 3.4, we see the detailed working of w_v. For query j, w_v = 0 and it cannot be answered from cache. As w_v = 0, current consistency is below target consistency and the node sends validation message to server and when receives reply with flag = 0; uses cached data item. As the node uses cached item which is verified by server to be valid; Wc is incremented by 1. Based on our formula (equation 1), n_{val} = 7 and new τ_c will be incremented to 7/9 = '0.77'. The new $\tau_c > \tau_t$ and new w_v is calculated to be '2' (based on equation 2). Now the node consecutively queries (k,l) which were answered from cache (for W = 2 and W = 1). After 2 queries w_v becomes 0 and then client sends request to server with 2 validation messages (k, l) and 1 request for new data item m. After server replied, the node caches data m and discovers that both k, l were valid answers and the new τ_c is 0.88. Thereafter, the new validity window w_v is found to be 13.

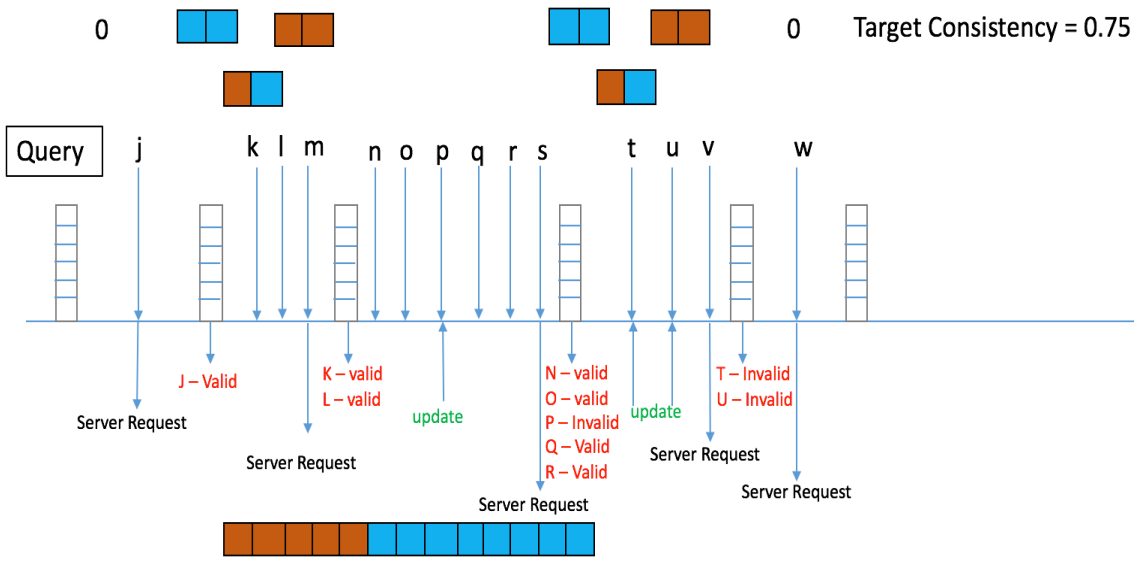
Now, the node has the ability to cache 13 items. The node starts caching n, o, p, q, r. The next request ('s') for a non-cached data item. If such a request for new data items arrives in between, the remaining validity window is neglected and reset to 0. This request (s) is immediately sent to server with previously answered queries n, o, p, q, r. This behavior of the window is expected because our system consistently adapts to user-defined target consistency (Without such a mechanism to control the consistency, the system performs poorly and eventually lead to an obsolete cache and cache misses).

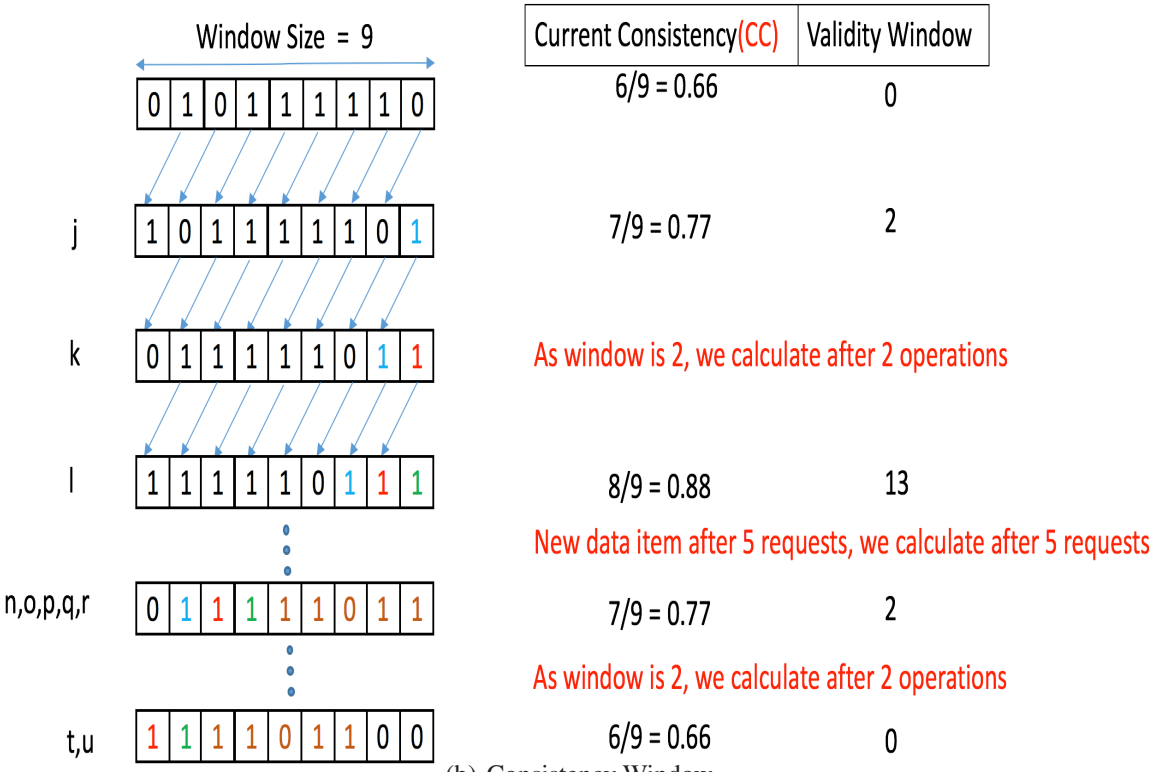
When server replies, other than p, all other answered queries were discovered to be valid. Again, the number of valid and invalid items in the consistency window are recalculated and the new validity window w_v results to be 2 but current consistency (0.77) remains the same. Lastly, two requests t, u are sent to server with t being valid and u as an invalid item. This drops the current consistency to 0.66 and the validity window is reset to 0. To conclude, as long as the consistency condition satisfies, a query for data item can be answered from cache.

Note: In calculating validity window, we assumed conversion of current consistency and target consistency decimal values directly to percentage and hits. Although in Fig. 3.4 , we assumed consistency window to be 9, but in our actual simulation we have configured it to be 100. Hence, the output is a direct conversion which is displayed in Fig. 3.5.

3.4 Summary

In this chapter, we have explained the PER-WC as a data access and invalidation scheme. We presented our system model and then reviewed validity window in depth with examples and pseudo code snippets. We also derive few formulae to calculate current consistency and validity window accurately. In the next chapter, we present our evaluation of the proposed extensions





(b) Consistency Window

Figure 3.4. PER-WC

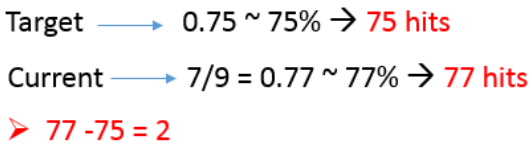


Figure 3.5. Validity Window

using experiments with collected and synthesized data.

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 Simulation Testbed

The proposed approach is carefully tested with the below test bed. The query and update inter arrival rate follow exponential distribution. The model is realistically designed with respect to data access; the entire data items in the database are classified into two types: hot and cold data items. Based on probability, 80% and 20% of query requests are for hot and cold data items, respectively. 33% of update requests are for hot data items but they are uniformly distributed within the hot subset. Likewise, 67% update requests are for cold subset. Similar data query and update patterns are found in [14]. Here, we do not consider multiple update frequencies for data items. Each node caches 5% of the data items in the database, i.e., 50 cache slots. We assumed N (1000) data objects at the server and the relative frequency for data objects follows a Zipf-like distribution. In terms of cache replacement policy, we have used Least Replacement Policy(LRU). The target consistency ranges from 0.7 to 0.95 with 0.05 steps, but we use 0.8 and 0.9 in most experiments unless otherwise stated to clearly see the performance differences. The consistency window is set to the last 100 queries. Summarized are the important simulation parameters in Figure 13.

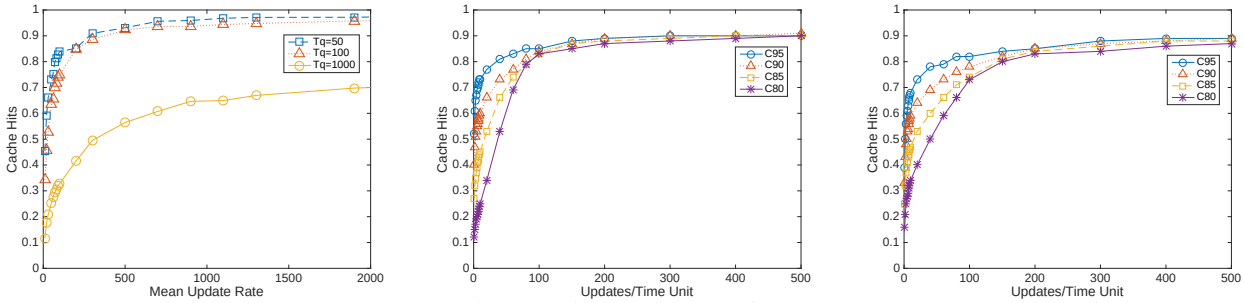
4.2 Simulation Results

We have performed extensive simulations on PER-WC. To evaluate PER-WC with PER, we have developed schemes on a discrete event-driven simulator based on C, CSIM. The key simulation parameters varied are: query rate, update rate, target consistency, validity window and consistency window. We first evaluate the performance of proposed PER-WC schemes in terms of query delay, cache hits, number of PER-WC uplink requests, adaptive consistency.

The existing schemes based on PER has been modified to become aware of sensitive consistency: PER-weak consistency (represented as PER-WC). We have included PER scheme as the base case for comparisons.

Table 4.1. Simulation Parameters

Parameter	Value
Number of nodes	100
Database Size (N)	1000 items
Hot data items	5% of DB
Cold data items	95% of DB
Mean Query Interval time(t_q)	50 – 100s
Mean update Interval time (t_u)	10 – 1000s
Adaptive Window	5 – 30
Target Consistency (T)	0.7 – 0.95
Consistency window (w_c)	100 queries



(a) Tq = 50,100,1000 (b) Tq = 50, Queries = 396229 (c) Tq = 100, Queries = 199973

Figure 4.1. The performance of PER and PER-WC as a function of mean update and query intervals.

4.3 Consistency Changes

The consistency changes were monitored for the entire simulation period. Target consistencies were varied from 0.7 to 0.9. In order to perform as expected, the system runs without expected results for certain time of simulation. Once a node caches enough data items, it shows a wave-like pattern in current consistency. This wave-like pattern denotes the close adaptability of current consistency to target consistency. Depending on the consistency condition and current consistency, PER-WC scheme can judiciously decide whether to access the cached data items to answer the queries directly or poll the server to validate the cached data items before answering the queries.

In Fig. 4.2, nodes flexibly change their target consistencies either between 0.9 and 0.95 or 0.8 and 0.9 or 0.7 and 0.8. After the cold stage, the current consistency of each node is quickly adjusted to its target consistency with a little fluctuation. In Fig. 4.2, we only shows the consistency changes of PER-WC scheme versus the simulation time. As the consistency increased to 1, the fluctuation in consistency decreased and vice-versa (i.e. The amount of fluctuation also varied with the consistency level). The variation in fluctuation per unit time is maximum when current consistency is 0.7 and minimum when current consistency is 0.9.

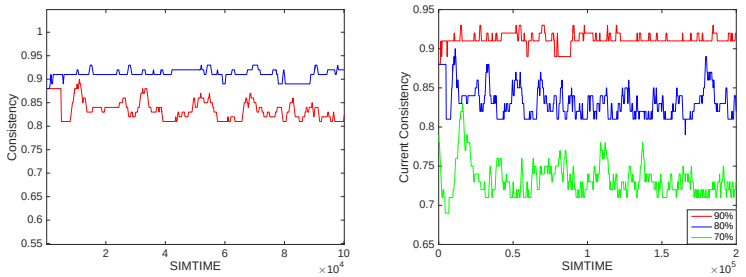
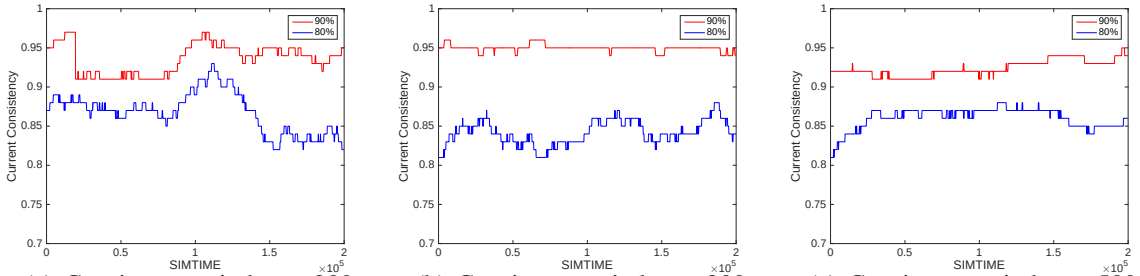


Figure 4.2. Adaptive Behaviour of Current Consistency



(a) Consistency window = 200 (b) Consistency window = 300 (c) Consistency window = 500

Figure 4.3. Change in Consistency with respect to consistency window changes)

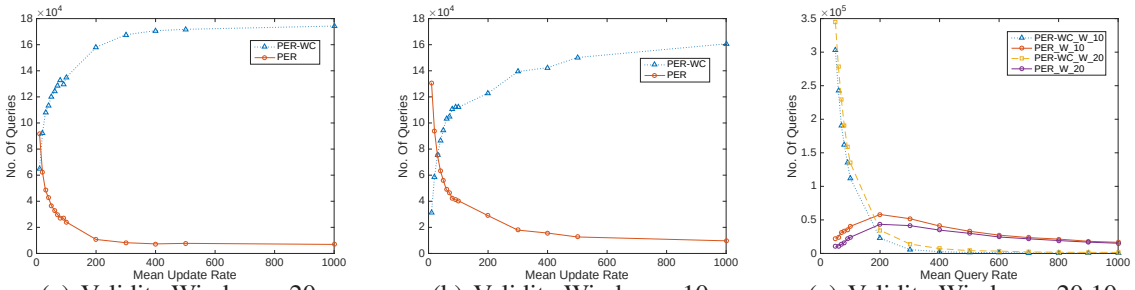
4.4 Consistency Window

All the simulation results generated in our experiments are with consistency window = 100. The effects on consistency changes previously explained are based on the validity of query history. We also experimented with last 200, 300 and 500 query histories for consistency window (W_c) to see how quickly the current consistency changes.

The total number of queries in the system were 200000, when the query interval was 100. Each node was able to successfully sent and receive 2000 queries in its lifetime. Thus, W_c = 200 (1/10th of total queries), 300 (15% of total queries), 500(1/4th of total queries) were used for the analysis. As we can expect, the current consistency changes faster with the smaller consistency window and vice-versa (Fig. 4.3). However, if the consistency window is too small, then the gap between the current and target consistencies increases. Hence, we set the consistency window to be 100 in this paper.

4.5 Cache Hit Ratio

We evaluate the cache hit as a function of update interval. In the PER-WC scheme, a cache hit occurs in the following cases: (i) The consistency condition is not satisfied, and the validated local cached data (against server's copy) is identified to be still valid and (ii) The consistency condition is satisfied, and a cached data item used in advance turns out to be valid after polling the server.



(a) Validity Window = 20 (b) Validity Window = 10 (c) Validity Window = 20,10

Figure 4.4. PER-WC Vs PER

In Fig. 4.1a, at the beginning of the simulation, the number of valid cached items is higher (0.2-0.6) for higher consistencies (0.8-0.95) but eventually they do not affect the cache hit much. PER-WC shows little variation in expected results until update rate = 100 and henceforth has the similar cache hits as PER regardless of the query intervals. As shown in Fig. 4.1, the PER-WC scheme shows slightly lower cache hit than the PER scheme depending on the target consistencies. As the target consistencies decreased to 0.8, validity of cached data items decreased and cache hit ratio decreased. The cache hit ratio is ranged from (85% - 90%).

4.6 PER-WC Vs PER - Smart Access

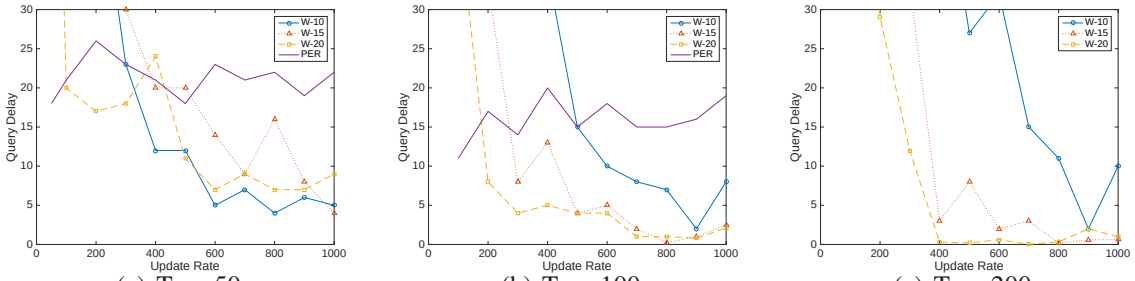
We measure the smart PER-WC accesses as a function of update and query intervals. A PER-WC access occurs, if the consistency condition is met and valid cached item is accessed. In Fig 4.4, for a certain query interval, the PER-WC scheme shows more number of smart accesses as the update interval increases. Due to the aggressive usage of cached data items in PER-WC approach, each node has the high probability of accessing valid cached data items leading to high number of PER-WC accesses.

As shown in Fig. 4.4, Initially number of PER requests were higher than PER-WC requests and become equal very quickly. But as the validity window increased, the number of smart accesses to PER-WC increased and PER requests sharply fall. The integrity of the sum of PER-WC & PER (PER-WC + PER) requests is maintained through the simulation.

Finally, as the query interval increases, the number of smart accesses reduces. PER-WC scheme falls acutely with an slight but considerable increase in query rate, Also, more numbers of PER-WC accesses are observed in the less target consistency (τ_t = 0.8)setting because of more chances in accessing cached data items.

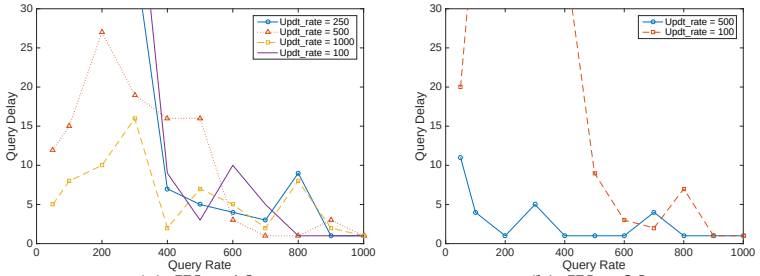
4.7 Validity Window

For a specified target consistency, if current consistency increases, validity window increases. It works in a feedback fashion, more the number of user queries answered by valid cached data



(a) Tq = 50 (b) Tq = 100 (c) Tq = 200

Figure 4.5. Query Delay Vs Update Interval



(a) W = 10 (b) W = 20

Figure 4.6. Query Delay Vs Query Interval

items, higher the next validity window. Simulations were performed for validity window = 30 (70%), 25 (75%), 20 (80%), 15 (85%), 10 (90%) and 5 (95%) respectively. The local maxima and minima in the graph depict the percentage of increase and decrease in current consistency against target consistency (in short, difference between current and target). If $\tau_c <= \tau_t$, then validity window becomes and remains 0. This is because at 0, PER-WC has no threshold to use cached data items to serve user queries and thus behaves as PER.

With an increase in query rate, the nodes aggressively request data items and hence the probability of cached data items to be valid increases and the validity window increases. Also, an increase in update rate causes the validity window to increase because it takes longer time for the cached data items to become invalid (or updates to cached data items becomes slow).

As the number of hot data items increase, validity window increases and remains above 0. On the contrary, as cold data items increase, it effects validity window in a non-negligible but considerable way. Here, the validity window shows very slow incremental growth.

4.8 Query Delay

We evaluate the query delay as a function of update and query intervals. In Fig. 4.5), the PER-WC scheme show better performance than PER schemes. Here, as the query and update rates increase, overall query delays reduce. In PER-WC scheme, items at the server become less frequently updated as update rate increase. This causes the items at the local cache of a client to

be valid for longer periods of time. Thus, the query delay reduces because more queries can be answered by the latest, updated cached data items without any form of validation.

In particular, the PER-WC scheme is sensitive to the target consistency. More number of cached items are used to answer the queries and the probability of a query being answered by a cached copy increases with the decrease in target consistency (e.g., s = 0.9-0.7).

In Fig. 4.6, as the query interval increases, the PER-WC scheme (s = 0.8) shows lower query delay than the PER-WC (s = 0.9). This is because more cached data items are accessed to answer queries without polling the server. The query delay gap is also clearly witnessed in Fig. 4.5, as the update interval increases.

4.9 Summary

In this chapter, we presented our performance evaluation results to compare the query delay, cache hit ratio of PER-WC to original PER. We observed that the adaptive nature of PER-WC occurs when the consistency window is restricted to 100. In the next chapter, we discuss the future research directions and the ongoing work. We conclude that on all our contributions to enhance the performance of PER when strong consistency assumption is relaxed.

CHAPTER 5

CONCLUSION AND FUTURE WORK

The existence and rise of wireless networks has increased the need for quality wireless links, reduction in bandwidth consumption and optimizing the battery life of wireless capable devices. For this purpose, we focus on the data access and invalidation scheme.

In this thesis, we developed PER-WC scheme, which is a consistency sensitive approach to existing PER mechanism. Specifically, we have evaluated the effects of adaptive consistency on query latencies. Our proposed model extends PER scheme and improves query latency by aggressively using cached data items. Another feature of PER-WC is its smooth integration with user-defined consistency. We introduced a window to adaptively change the current consistencies of the clients. PER-WC scheme behaves as PER when the validity window = 0. From the simulation results it is also evident that our algorithm outperforms the existing approach.

Finally, as discussed earlier about Call-Back (CB) mechanism, we would like to extend our proposed approach to CB and examine if it is feasible to arrive at any promising results. We would also like to test the nodes for energy efficiency. Compared to PER, we expect better energy savings as our proposed mechanism does not depend on much radio activity.

BIBLIOGRAPHY

[1] (2015) Number of mobile subscribers. [Online]. Available: https://gsmaintelligence.com/

[2] (2015) Population in the world. [Online]. Available: http://www.census.gov/popclock/world

[3] G. Lee, I. Jang, S. Pack, and X. Shen, Fw-das: Fast wireless data access scheme in mobile networks, Wireless Communications, IEEE Transactions on, vol. 13, no. 8, pp. 42604272, 2014.

[4] H. Chen, Y. Xiao, and S. V. Vrbsky, An update-based step-wise opti- mal cache replacement for wireless data access, Computer Networks, vol. 57, no. 1, pp. 197212, 2013.

[5] H. Chen, Y. Xiao, and X. S. Shen, Update-based cache access and replacement in wireless data access, Mobile Computing, IEEE Trans- actions on, vol. 5, no. 12, pp. 17341748, 2006.

[6] S. Lim, Y. Lee, J. Cheon, M. Min, and W. Wang, User-defined consistency sensitive cache invalidation strategies for wireless data access, Computer Communications, vol. 41, pp. 5566, 2014.

[7] G. Cao, A scalable low-latency cache invalidation strategy for mobile environments, Knowledge and Data Engineering, IEEE Transactions on, vol. 15, no. 5, pp. 12511265, 2003.

[8] K. Lee, I. Jang, S. Pack, and W. Lee, Design and analysis of cooper- ative wireless data access algorithms in multi-radio wireless networks, Wireless networks, vol. 19, no. 1, pp. 1729, 2013.

[9] S.-H. Lim, S. W. Lee, M. Sohn, and B.-H. Lee, Energy-aware optimal cache consistency level for mobile devices, Information Sciences, vol. 230, pp. 94105, 2013.

[10] W. Xu, W. Wu, H. Wu, J. Cao, and X. Lin, Cacc: A cooperative approachto cache consistency in wmns, Computers, IEEE Transactions on, vol. 63, no. 4, pp. 860873, 2014.

[11] R. Tiwari and N. Kumar, An adaptive cache invalidation technique for wireless environments, Telecommunication Systems, pp. 117, 2015.

APPENDIX

1.CODE TO SET ALL INITIAL CONDITIONS OF THE SYSTEM

```
//==================== Definitions ============================
#define NUM_CLIENT 100
#define NUM_DB_ITEM 1000
#define DATA_ITEM_SIZE 16*1024*8 // bits
#define MAX_ITEM_VALUE 999
#define BANDWIDTH 384000 // bits/sec
#define NUM_CACHE_ITEM 50

#define NUM_HOT_DATA_ITEMS 50 // the number of Hot Data Items
#define HOT_DATA_ACCESS_PROB 0.80 // Hot Data Access Probability
#define HOT_DATA_UPDATE_PROB 0.33 // Hot Data Update Probability
#define prob_of_validity 0.9 // Probability of inserting Valid Items

//((double)DATA_ITEM_SIZE/(double)BANDWIDTH)
#define TRANS_TIME_PER_ITEM  0.10
#define SERVER_HOLD          0.50

#define NUM_CONS_WINDOW 100
#define adap_window_size     10
#define TargetConsistency    0.90

//=========================================================
```

2.UPDATE FUNCTION: HOW SERVER UPDATES DATA ITEMS

```
//==================== updateProc() ====================
void updateProc()
{


// update an item of hot data
if(uniform(0,1) < (double)HOT_DATA_UPDATE_PROB) {
id = random(0, (int)NUM_HOT_DATA_ITEMS - 1);
totalHotUpdates++;
}
```

```
else { // cold data update
id = random((int)NUM_HOT_DATA_ITEMS, (int)NUM_DB_ITEM - 1);
totalColdUpdates++;
}

server.myDB[id].value = random(1, MAX_ITEM_VALUE);
server.myDB[id].TS = clock;

server.updateList[id] = 1;
}

}
//================== End updateProc() =================
```

3.SERVER FUNCTION TO BUILD AND CREATE REPLIES FOR REQUESTED DATA

```
//==================== updateMsgData() ====================
TIME updateMsgData(NODE_PTR_TYPE msgData) {
NODE_PTR_TYPE front = NULL;
NODE_PTR_TYPE tmpNode = NULL;

TIME delayData = TRANS_TIME_PER_ITEM;
int i = 0;

if((tmpNode = popFromReqList()) != NULL)
{

flushMsgData(msgData);
front = msgData;
front->client_id = tmpNode->client_id ;

// FOR THE GIVEN TARGET CONSISTENCY
{
if(tmpNode->adap_window[i].TS == 0) // NEW DATA
{
NewDatas++;

// NEW DATA WITH NO SERVER UPDATE
if(server.myDB[tmpNode->adap_window[i].id].TS==0)
{

front->adap_window[i].TS = clock;
front->adap_window[i].value = tmpNode->adap_window[i].value;
}

// SERVER HAS UPDATED DATA
else
{
DataUpdates++;
```

```
front->adap_window[i].TS = server.myDB[tmpNode->adap_window[i].id].TS;
front->adap_window[i].value
 = server.myDB[tmpNode->adap_window[i].id].value;
}
}

// VALIDATION MSG: VALID CACHED DATA
else if(tmpNode->adap_window[i].TS >=
 server.myDB[tmpNode>adap_window[i].id].TS)
{ // DATA AT CLIENT IS UPTODATE
Cachehits++;
front->adap_window[i].TS = tmpNode->adap_window[i].TS;
front->adap_window[i].value = 0;

}

// VALIDATION MSG: INVALID CACHED DATA
else if(tmpNode->adap_window[i].TS < server.myDB[tmpNode
->adap_window[i].id].TS)
{ // DATA AT CLIENT BECAME INVALID
DataUpdates++;
front->adap_window[i].TS =
 server.myDB[tmpNode->adap_window[i].id].TS;
front->adap_window[i].value =
server.myDB[tmpNode->adap_window[i].id].value;

}
}
// SENDING REPLY TO CLIENTS
send(client[front->client_id].input, (long)msgData);

}
else
return delayData;

return delayData;
}
//================== End updateMsgData() ==================

4.ADAPTIVE CONSISTENCY INITIALIZATION FOR EACH CLIENT

//==================== initConSens() ====================
void initConSens(int n) {

client[n].tarCons = tarCons; // TARGET CONSISTENCY
client[n].curCons = 0; // CURRENT CONSISTENCY
client[n].cntConsWindow = 0; // CONSISTENCY WINDOW
client[n].indexConsWindow = 0;
client[n].validitiy_window = 0; // VALIDITY WINDOW
```

```
client[n].indexAdapWindow = 0;


// POPULATING CONSISTENCY WINDOW
for(i = 0; i < (int)NUM_CONS_WINDOW; i++) {
if(uniform(0,1) <= (double)prob_of_validity)
{
client[n].myConsWindow[i] = 1;
counter++;
}
else
{
client[n].myConsWindow[i] = 0;
}
}

// CURRENT CONSISTENCY
client[n].curCons = (double)counter/(double)NUM_CONS_WINDOW;


current = client[n].curCons * 100;
target = client[n].tarCons * 100;

// CHECK IF CURRENT CONSISTENCY IS GREATER THAN TARGET CONSISTENCY
if(current >= target)
{
client[n].validitiy_window = (current - target);
client[n].CountWindow[client[n].CountIndex++] =
 client[n].validitiy_window;
}
}
//=====================================================

5.GENERATION OF A QUERY AND SEND A REQUEST

//==================== queryProc() ====================
void queryProc(int n) {

create("query");

while(clock < SIMTIME)
{
hold(expntl(Tquery));

if(sent_flag == 1) // FLUSHING DATA AFTER SENDING A QUERY TO SERVER
{
sent_flag = 0;
sent_items++;
int p=0;
```

```
msg->client_id = -1;
for(p = 0; p < ((int)adap_window_size); p++)
{
msg->adap_window[p].id = -1;
msg->adap_window[p].TS = 0.00;
msg->adap_window[p].value = -1;
}
client[cl_id].indexAdapWindow = 0;
//client[cl_id].validitiy_window = 0;
}

dupCnt = 0;
id = -1;

// SELECTING HOT DATA ITEM ID

if(uniform(0,1) < (double)HOT_DATA_ACCESS_PROB) {
do {
id = random(0, (int)NUM_HOT_DATA_ITEMS - 1);

if(++dupCnt > ((int)NUM_HOT_DATA_ITEMS * 20)) {
break;
}
} while(client[cl_id].myQuery[id].flag != 0);
//totalHotQueries++;
}

// SELECTING A COLD DATA ID
else {
do {
id = random((int)NUM_HOT_DATA_ITEMS, (int)NUM_DB_ITEM- 1);
}while(client[cl_id].myQuery[id].flag != 0);
}

// REMOVING DUPLICATE ENTRIES IN THE VALIDITY WINDOW

for(i = 0 ; i <= ((int)adap_window_size-1) ; i++)
{
if(id == msg->adap_window[i].id)
{
duplicate_flag = 1;
break;
}
}

if(duplicate_flag == 1 || id == -1)
{
duplicate_flag = 0;
continue;
```

```
}

// LOADING THE MESSAGE TO SERVER
msg->adap_window[client[cl_id].indexAdapWindow].id = id;
msg->adap_window[client[cl_id].indexAdapWindow].TS =
 client[cl_id].myCache[id].TS;
msg->adap_window[client[cl_id].indexAdapWindow].value =
 client[cl_id].myCache[id].value;
totalRequests++;


// NEW AND NON_CACHED ITEMS
if(client[cl_id].myCache[id].isCached == 0)
{
msg->client_id = cl_id; // Loading Client's ID
send(server.input, (long)msg); // Sending Msg to Server
// Propogation Delay Per Item
hold(expntl((double)TRANS_TIME_PER_ITEM));
sent_flag = 1;
client[cl_id].validitiy_window = 0; // Updating the validity Window
}
//PER-WC , CACHED ITEM WHEN VALIDITY WINDOW IS AVAILABLE
else if(client[cl_id].myCache[id].isCached == 1 &&
 client[cl_id].validitiy_window > 0)
{
client[cl_id].indexAdapWindow++;
client[cl_id].validitiy_window--; // Decreasing Validity Window By 1
// Query Delay
totalQueryDelay += (clock - client[cl_id].myQuery[id].time);
}

//REQUEST FOR CACHED ITEM WHEN VALIDITY WINDOW IS NOT AVAILABLE
else if(client[cl_id].myCache[id].isCached == 1 &&
 client[cl_id].validitiy_window <= 0)
{

       // FEW QUERIES WERE ALREADY LOADED IN MSG
if(client[cl_id].indexAdapWindow > 0)
{
// FLAG = 2 indicates queries answered by cached data item
client[cl_id].myQuery[id].flag = 2;
 // QUERY DELAY
totalQueryDelay += (clock - client[cl_id].myQuery[id].time);

}
// VALIDATION MSG WHEN CURRENT CONSISTENCY IS BELOW TARGET CONSISTENCY
else if(client[cl_id].indexAdapWindow == 0)
{
// FLAG = 3 indicates the query that needs to be validated by server
```

```
client[cl_id].myQuery[id].flag = 3;
no_of_no_window_data_with_flag3++;
}

msg->client_id = cl_id;
send(server.input, (long)msg);
hold(expntl((double)TRANS_TIME_PER_ITEM)); // PROPOGATION DELAY
sent_flag= 1;
client[cl_id].validitiy_window = 0; //RESETTING VALIDITY WINDOW
}
client[cl_id].QueryCnt++; // COUNT TOTAL QUERIES
}
return;
}
//================== End queryProc() ==================

6.WHEN CLIENT RECEIVES REPLY FROM SERVER, EXTRACT CACHE HITS
 and CONSISTENCY CHANGES and FIND QUERY DELAY

//==================== receiveMsgData() ====================
void receiveMsgData(int n)
{
receive(client[n].input, (long *) &rcvMsg); // RECEIVE REPLY
front = rcvMsg;
FOR A SET CONSISTENCY
{

id = front->adap_window[i].id;
totalAnswers++; // COUNT TOTAL REPLIES

// A REPLY TO NON_CACHED DATA ITEM
if(client[n].myCache[id].isCached != 1)
{
deleteACachedItem(n);    // REPLACING A CACHED DATA ITEM
client[n].cacheCnt++;
}

// LOADING THE DATA INTO THE CACHE
client[n].myCache[id].isCached = 1;
client[n].myCache[id].isValid = 1;
client[n].myCache[id].timeInvalid = 0;
client[n].myCache[id].id = front->adap_window[0].id;
client[n].myCache[id].TS = front->adap_window[0].TS;

//CACHED ITEM IS INVALID
if(front->adap_window[i].value > 0)
{
client[n].myCache[id].value = front->adap_window[i].value;
}
```

```
// CACHED ITEM IS VALID
else
{
client[n].hitCnt++;
incCurCons(); // INCREASE CONSISTENCY
}

if(client[n].myQuery[id].flag == 1 || client[n].myQuery[id].flag == 3)
{
// QUERY DELAY
totalQueryDelay += (clock - client[n].myQuery[id].time);
}

// RESETTING
client[n].myQuery[id].flag = 0;
client[n].myCache[id].lastAccessedTime = clock;
}

return;
}
//================== End receiveMsgData() ==================
```

7.FUNCTION TO INCREASE AND DECREASE THE CONSISTENCY

```
//==================== incCurCons() ====================
void incCurCons(int n) {

client[n].myConsWindow[client[n].indexConsWindow++] = 1;

if(client[n].indexConsWindow == (int)NUM_CONS_WINDOW) {
client[n].indexConsWindow = 0;
}

client[n].curCons = getCurCons(n); // GET CURRENT CONSISTENCY

// INCREMENTING WINDOW
current = client[n].curCons * 100;
target = client[n].tarCons * 100;

// INCREASE

if(current > target)
{
client[n].validitiy_window = (current-target);
client[n].CountWindow[client[n].CountIndex++] =
 client[n].validitiy_window;
}

// DECREASE
```

```
else if (current <= target)
{
client[n].validitiy_window = (current-target);
client[n].CountWindow[client[n].CountIndex++] =
 client[n].validitiy_window;
}

return;
}
//================== End incCurCons() ==================
```

8.FUNCTION TO RETRIEVE CURRENT CONSISTENCY USING CONSISTENCY WINDOW

```
//==================== getCurCons() ====================
double getCurCons(int n) {

if(client[n].cntConsWindow < (int)NUM_CONS_WINDOW ) {
client[n].cntConsWindow++;
}
// COUNTING VALID AND INVALID ITEMS IN CONSISTENCY WINDOW
for(i = 0; i <=((int)NUM_CONS_WINDOW-1); i++)
{
if(client[n].myConsWindow[i] == 1)
{
cntInvalid++;
}
}
return (double)cntInvalid / (double)NUM_CONS_WINDOW;
}
//================== End getCurCons() ==================
```

9.CACHE REPLACEMENT POLICY- LRU

```
//==================== deleteACachedItem() ====================
void deleteACachedItem(int n) {


for(i = ((int)NUM_DB_ITEM - 1); i >= 0; i--) {
// not cached
if(client[n].myCache[i].isCached == 0) {
continue;
}

if(client[n].myCache[i].isValid == 0) {
curr = i;
break;
}
}
```

```
// if there is no invalid item, and then find the oldest of valid - LRU
if(curr == -1) {
for(i = ((int)NUM_DB_ITEM - 1); i >= 0; i--) {
// not cached
if(client[n].myCache[i].isCached == 0) {
continue;
}
if(curr == -1) {
curr = i;
}
else if(client[n].myCache[i].lastAccessedTime
< client[n].myCache[curr].lastAccessedTime) {
curr = i;
}
}
}
// drop an oldest and/or invalid item
client[n].myCache[curr].isCached = 0;
client[n].myCache[curr].isValid = 0;
client[n].myCache[curr].value = 0;
client[n].myCache[curr].TS = 0;
client[n].cacheCnt--;

return;
}
//================== End deleteACachedItem () ==================//
```