Handling Client Mobility and Intermittent Connectivity in Mobile Web Accesses*

Purushottam Kulkarni, Prashant Shenoy and Krithi Ramamritham**

Department of Computer Science, University of Massachusetts, Amherst
{purukulk,shenoy,krithi}@cs.umass.edu

Abstract. *Wireless devices are being increasingly used to access data on the web. Since intermittent connectivity and client mobility are inherent in such environments, in this paper, we examine the impact of these factors on coherent dissemination of dynamic web data to wireless devices. We introduce the notion of eventual–delta consistency, and in the context of push and pull–based dissemination propose (i) buffering techniques at the proxy to mask the effects of client disconnections and (ii) application–level handoff algorithms to handle client mobility. Our experimental evaluation demonstrates that push is better suited to handle client disconnections due to its lower message overhead and better fidelity, while pull is better suited to handle client mobility due to its lower handoff overheads.*

1 Introduction

1.1 Motivation

Recent advances in computing and networking technologies have led to a proliferation of mobile devices with wireless networking capabilities. Like traditional wired hosts, wireless mobile devices are frequently used to access data on the web. These devices differ significantly from their wired counterparts with respect to their capabilities and characteristics—(i) mobile clients can move from one location to another, while the location is fixed for wired hosts and (ii) disconnections and reconnections are frequent in mobile wireless environments, while network failures are an exception in the wired domain. A concurrent trend is the increasing use of dynamic data in today's web. Unlike static web pages that change infrequently, dynamic web data is time–varying and changes frequently (once every few seconds or minutes). In this paper, we address the challenges arising from the intersection of these two trends, namely, the access of dynamic web data using networked mobile wireless devices.

1.2 Problem Formulation and Research Contributions

Consider a mobile client that accesses dynamic data items over a wireless network (see Figure 1(a)). Like in many wired environments, we assume that client web requests are sent to a proxy. The proxy and the server are assumed to communicate over a wired network, while the proxy and the client communicate over a wireless network via a base station(access point); the latter network can be either be a wireless LAN such as 802.11b or a wireless WAN such as a GSM data network. Each base station in the wireless network is assumed to be associated with a proxy.

* This research was supported n part by NSF grants CCR-0098060, EIA-0080119 and CCR-0219520.

** Also with the Indian Institute of Technology Bombay, India.

Intermittent connectivity and client mobility are common in such wireless environments; we discuss each issue in turn. We assume an environment where mobile clients can get disconnected at any time—disconnections occur primarily due to poor network coverage at a particular location or due to the mobile device powering down to save energy. Since the proxy and the server are unreachable during a disconnection, dynamic data items cached at the client can not be refreshed and may become stale. Consequently, to prevent a violation of coherency guarantees, client disconnections should be detected in a timely fashion so that preventive measures can be initiated. Observe that, since the proxy and the server are on a wired network, the proxy can continue to receive updates from the server during a disconnection (but can not propagate these updates to the client). Hence, data cached at a proxy continues to be coherent even during a disconnection—an observation that can be exploited for efficient resynchronization of the client cache upon reconnection. Thus, coherency maintenance in the presence of intermittent connectivity requires: (i) techniques for timely detection of client disconnections and reconnections, and (ii) techniques for efficient resynchronization of cache state at the client upon reconnection.

A second consideration in wireless environments is client mobility. Since a mobile client can move locations, the proxy may decide to hand over responsibility for servicing the client to another proxy in the client's vicinity. Such a *handoff* is desirable since a nearby proxy can reduce network overheads and provide better latency to client requests; the handoff involves a transfer of client–specific cache state from the initiator to the recipient proxy. The design of such application–level handoff mechanisms involves two research issues: (i) the proxy needs to decide *when* to initiate the handoff and (ii) *how* to perform a handoff, the steps involved in such a procedure. These design decisions have implications on the overheads imposed by handoffs and the latency seen by user requests. For instance, the handoff process should be seamless to end-clients—the client should not miss any updates to cached data due to the handoff and all temporal coherency guarantees should be preserved with minimal overheads.

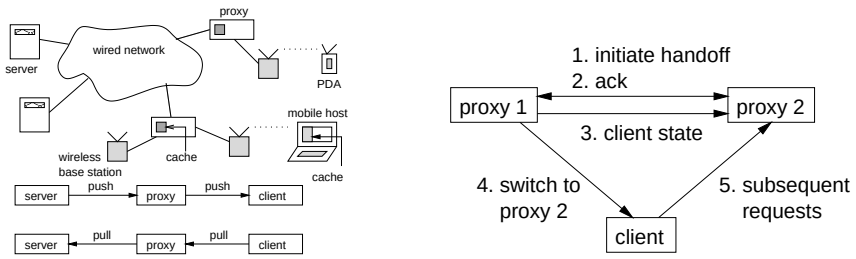
Thus, in the context of intermittent connectivity and client mobility we make three contributions in this paper. First, we propose coherency semantics appropriate for dynamic data access in mobile environments. We then consider two canonical techniques for dynamic data dissemination—push and pull—and show how to adapt these techniques to (i) reduce the overheads of coherency maintenance in the presence of client mobility and (ii) provide temporal coherency guarantees even in the presence of intermittent connectivity.

2 Handling intermittent connectivity in mobile environments

2.1 Coherency semantics

Due to the time–varying nature of dynamic data, cached versions of data items need to be *temporally coherent* with the data source. To ensure this property, we define a coherency semantics called *eventual–delta* consistency (Δ_e) for mobile environments. Δ_e semantics provide stricter guarantees when a mobile client is connected and weaken the guarantees upon a disconnection. Formally, Δ_e consistency is defined as follows

$$\Delta_e\text{consistency} \Rightarrow \begin{cases} |S_t - C_t| \leq \Delta & \text{if connected} \\ C_t \rightsquigarrow S_t & \text{if disconnected} \end{cases} \quad (1)$$



(a) System model (b) Handoff mechanism

Fig. 1. System model and mechanisms for intermittent connectivity and client mobility.

where C_t and S_t denote the state of a data item d at the client and the server, respectively, at time t, Δ denotes the desired coherency bound, and $C_t \rightsquigarrow S_t$ indicates that C_t will eventually catch up with S_t on reconnection.

Coherency mechanisms that implement Δ_e consistency can be *buffered* or *unbuffered*. In the unbuffered scenario, each new update from the server overwrites the previously cached version—the proxy cache only maintains the version corresponding to the most recently received update. In the buffered scenario, the proxy buffers recent updates from the server that are yet to be propagated to the client. Thus, updates not yet seen by the client can be buffered at the proxy and delivered next time the client refreshes its cache.

2.2 Maintaining Δ_e consistency using Push and Pull

There are two possible techniques to implement Δ_e consistency — *push* and *pull*. In the push approach, cache coherency is maintained by pushing updates to dynamic data items from the server to the proxy and from proxy to the clients. In the pull approach, the onus is on the proxy to poll the server for updates and on the client to poll the proxy to fetch the updates.

We assume that a client registers each data item of interest with the proxy and specifies a coherency tolerance of Δ for each item. In the push approach, the proxy in turn registers these data items with the corresponding server along with the corresponding coherency tolerances. The server then tracks changes to each such data item and pushes all updates that exceed the tolerance Δ to the proxy. The proxy in turn pushes these updates to the client. In the event the client is disconnected, the updates are buffered and propagated to the client upon reconnection. The proxy uses heartbeat messages and client requests as implicit heartbeats to detect client disconnections and reconnections [3]. In the pull approach, the proxy periodically polls the server to fetch updates. To meet client–specified coherency guarantees the proxy should poll the server at a minimum rate of Δ. The time between successive refreshes, *time-to-refresh(TTR)*, for each data item can be statically or dynamically determined. The TTR value has an impact on the fidelity of data at the proxy and the number of polls. The proxy buffers all the updates fetched from the server and on a client poll delivers those that are new since the previous poll. Observe that in the push approach, both the proxy and the server are *stateful*, whereas in the pull approach only the proxy is *stateful*.

1. $P_1 \rightarrow P_2$: initiating handoff for C
2. $P_2 \rightarrow P_1$: ACK
3. $P_1 \rightarrow P_2$: C needs $\{< d_1, \Delta_1, S(d_1) >, \ldots\}$
4. $P_2 \rightarrow P_1$: ACK
5. $P_1 \rightarrow S$: Send updates for object in C to P_2
6. $S \rightarrow P_1$: ACK
7. `if` C `is connected`
   i. $P_1 \rightarrow C$: Switch to P_2
   ii. $C \rightarrow P_1$: ACK
   `else if` C `is disconnected`
   Update record of handoff for C at P_1
8. $P_1 \rightarrow P_2$: Take-over client C $\{< d_1, S(d_1) >, \ldots\}$
9. $P_2 \rightarrow P_1$: ACK
10. $P_1 \rightarrow S$: Update state of objects in C transferred from P_1
11. $S \rightarrow P_1$: ACK

(a) Handoff algorithm for push

1. $P_1 \rightarrow P_2$: initiating handoff for C
2. $P_2 \rightarrow P_1$: ACK
3. $P_1 \rightarrow P_2$: C needs $\{< d_1, ttr_1, S(d_1) >, \ldots\}$
4. $P_2 \rightarrow P_1$: ACK
5. `if` C `is connected`
   i. $P_1 \rightarrow C$: Switch to P_2
   ii. $C \rightarrow P_1$: ACK
   `else if` C `is disconnected`
   Update record of handoff for C at P_1
6. $P_1 \rightarrow P_2$: Take-over client C $\{< d_1, S(d_1) >, \ldots\}$
7. $P_2 \rightarrow P_1$: ACK

(b) Handoff algorithm for pull

Fig. 2. Handoff algorithms

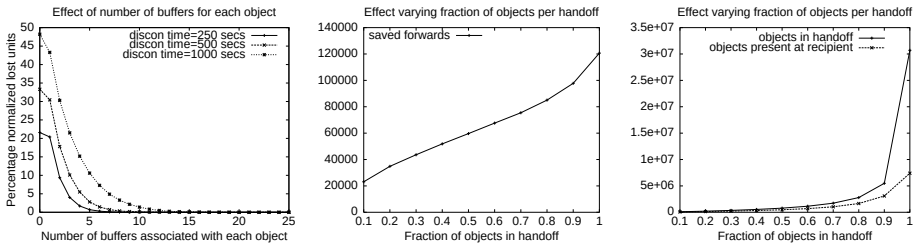
3 Handling client mobility

As a mobile client moves from one location to another, it switches between base-stations to maintain connectivity. As explained earlier, each mobile client uses a proxy to request web content and it is possible for a mobile client to use the same proxy to service its requests even when it moves locations (since it can always communicate with this proxy so long it remains connected). However, other proxies may be more advantageously located with respect to the new client location. In such a scenario, having the mobile client use a nearby proxy to service its requests may result in improved response times and lower network overhead. To enable such a transition, the proxy needs to hand over responsibility for servicing client requests to the new proxy and also transfer client–specific state to the new proxy. The important steps of such a procedure are: (i) initiating a handoff, (ii) transferring client–state and (iii) committing the handoff (refer to Figure 1(b) and see [3] for details).

Several issues arise in the design of such application–level handoff mechanisms. First, the proxy needs to be aware of the client's physical location so that it can determine whether a handoff is necessary. A second issue is how to ensure that the handoff is seamless and transparent to the end–client. A third issue in the design of handoff algorithms is when and how client–specific state information is transferred from the initiator proxy to the recipient proxy. Depending on the exact mechanisms, following types of handoffs are possible: (i) *Optimistic versus pessimistic*: Objects are transferred lazily in pessimistic handoffs, while they are transferred in an eager fashion in optimistic handoffs and (ii) *Complete versus partial*: In complete handoffs, information about *all* objects accessed by the client is transferred, while in partial handoffs the recipient proxy transfers information only about a fraction of the objects. Figures 2(a) and 2(b) are handoff algorithms for push–based and pull–based mechanisms respectively, detailed explanations of the two algorithms can be found in [3].

4 Experimental Evaluation

We evaluate the efficacy of our approaches using simulations. We use a Fixed-path mobility model [3] to simulate client mobility and an ON–OFF process to simulate client



(a) Buffers per object (b) Saved forwards (c) Objects in handoff

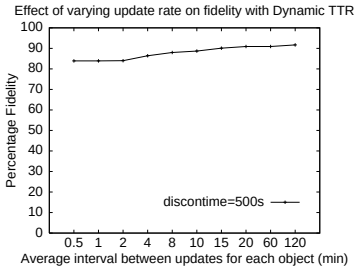
Fig. 3. Effect of intermittent connectivity and client mobility in push–based data dissemination.

disconnections and reconnections. The simulation environment consists of a number of cells and mobile clients that use the above processes to simulate mobility and intermittent connectivity during web accesses. The workload used for our experiments consisted of the publicly available proxy DEC trace and a synthetic trace (see [3] for details). Updates to objects at the server are simulated by an update process.

To study the effect of intermittent connectivity of clients, we varied three different parameters, one at a time: the maximum disconnection time of clients, the size of the per–object circular buffer and the average time between updates of mutable objects at the server. As the dependent variable, we measured the percentage of *normalized lost units*. Figure 3(a) shows the impact of varying size of the circular buffer associated with each object on the normalized lost units in a push–based approach. We see that, for a given disconnection time, the loss curve has a "knee" beyond which the loss percentage is small. Choosing a buffer size that lies beyond the knee ensures that the proxy can effectively mask the effect of a disconnection (by delivering all updates received during such periods). In general, we find that a small number of buffers (15–20) seem to be sufficient to handle disconnections as large as 15 minutes. As few as 5 buffers can reduce the loss rate from 33% to 2.8% for disconnections of up to 500 seconds.

Figure 3(b) shows the benefit of using application–level handoffs to handle client mobility. The measured metric *saved–forwards* is the number of requests served from the proxy as a result of previous handoffs. Greater the fraction of client–specific state transferred, larger are the savings at an increased message overhead (see Figure 3(c)).

Next we compare the push and pull approaches. Figure 4(a) plots fidelity of data for push and pull in presence of disconnections. As expected, the push–based and the pull with static TTR approaches yield perfect fidelity. Pull with dynamic TTR, which dynamically adjusts TTR values based on observed rate of change of object yields 83.9%–91.7% fidelity. Figure 4(b) compares the average message overhead per handoff in cases of push and pull. We see that pull has lower handoff overhead than push regardless of the fraction of client–state transferred. A detailed experimental evaluation of our techniques is presented in [3].



(a) Fidelity of objects at clients

Fraction of objects in handoff	Pull	Push
0.1	5.34	7.14
0.2	9.25	11.02
0.3	14.35	16
0.4	20.65	22.5
0.5	28.82	31.15
0.6	40	44.15
0.7	57.14	64.43
0.8	87.8	102.94
0.9	174	215.64
1	1323.75	2057.85

(b) Handoff overhead

Fig. 4. Comparison of push and pull based dissemination approaches.

5 Related work

File systems such as CODA [2] and distributed systems like Bayou [1] have investigated disconnected operations for mobile clients, techniques for hoarding files and maintaining consistency in weakly connected systems. Several other techniques for maintaining coherency of data in disconnected environments and disseminating data using broadcasts exist and we compare them in [3]. While the previous efforts are for disseminating static web data and for reconciling changes at clients with other hosts on reconnection, our solution is better suited for dynamic data that changes frequently and disseminating data from servers to mobile read–only clients.

6 Conclusions

In this paper, we studied the impact of client mobility and intermittent connectivity on disseminating dynamic web data to mobile hosts. We introduced the notion of eventual–delta consistency and in the context of push–based and pull–based dissemination proposed: (i) proxy–based buffering techniques to mask the effects of client disconnections and (ii) application–level handoff algorithms to handle client mobility. As part of ongoing work, we are implementing these techniques in a prototype proxy for mobile environments.

References

1. W. K. Edwards, E. D. Mynatt, K. Petersen, M. Spreitzer, D. B. Terry, and M. Theimer. Designing and Implementing Asynchronous Collaborative Applications with Bayou. In *ACM Symposium on User Interface Software and Technology*, pages 119–128, 1997.
2. J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *Thirteenth Symposium on Operating Systems Principles*, volume 25, pages 213–225, 1991.
3. P. Kulkarni, P. Shenoy, and K. Ramaritham. Handling Client Mobility and Intermittent Connectivity in Mobile Web Access. Technical Report TR02–35, University of Massachusetts, Amherst, 2002.