Recovering Mobile Users in the Context of Internet Transactions (iTXs)

Kaushik Dutta[*], Debra VanderMeer,[*] Anindya Datta,[*†] Krithi Ramamritham[‡]

December 14, 2001

1 Introduction

With the expansion of Web sites to include business functions, a user interfaces with e-businesses through an interactive and multi-step process, which is often time-consuming. A loss of connection, or any other system failure, can result in the loss of work accomplished prior to the disruption. This work must then be repeated upon subsequent reconnection – often at significant cost in time and computation. In mobile wireless environments, this "disconnection-reconnection-repeat work" cycle may cause web clients to incur substantial monetary as well as resource (such as battery power) costs. With the introduction of high-speed 3G networks, the use of mobile devices to access Internet applications will no doubt increase dramatically, making this problem more acute.

Consider a scenario in which a user is buying an airline ticket over his wireless Internet connection. He first logs on to the airline site with his frequent flier number, then checks his frequent flier mileage. He then enters his preferred travel dates and destination, and chooses among the itineraries offered by the site, selects his seat, enters his credit card information, and receives a confirmation of the purchase. If the user's wireless connection drops (as occurs frequently [6]) late in this interaction, e.g., during the purchase step, he must reconnect to the Internet, and restart the sequence of actions. This includes expensive steps, such as processor-intensive login and authentication, I/O intensive database lookups, as well as consume valuable connections to the database. Significant amounts of time and effort are wasted in redoing previously completed work, and, in the case of a wireless environment, there are additional costs in battery resources and airtime. In the scenario described above, we would like to be able to avoid the repetition of work (computation, communication, I/O) required after a connection disruption. Ideally, we would like *to reduce the cost of recovering a user's interaction.*

Much like a database transaction, in many ways, this interaction consists of a number of actions aimed at achieving a particular goal (or set of goals). However, the recovery of such interactions is quite different from the classical notion of transaction recovery. In recovering transactions, the focus is on ensuring that the underlying database system is rendered consistent. Thus, if a transaction prematurely aborts, the transaction is rolled back and is resubmitted after the database system recovers.

Here, though, the database and application systems have the necessary recovery components in place to recover the database system. *These system recovery components do not address the situation where the client portion of a user's interaction history is lost in a connection failure.* What is needed here is a mechanism through which users can efficiently and quickly restart from an appropriate point prior to disconnection (yet subsequent to the beginning of the interaction) so that the interactions can resume from that intermediate point. **Thus, our concern is not "system recovery", but rather "user recovery".**

Note that the "user recovery" problem is not equivalent to the classical database transaction-processing and recovery problem. To illustrate this point, we briefly discuss the differences between the two problem domains.

The theory and application of transaction-based processing in the context of database systems are well-researched topics. Early work, such as [1], describes the basic tenets of transaction processing associated protocols. Long transactions (e.g., Sagas) are discussed in [5]. The metaphor of transaction processing has been extended to other application areas. The ACTA framework [3] describes a formal framework for extending the idea of a transaction to other areas. For example, [2] describes workflow processing using transactions and workflow templates, while [7] utilizes transactions to model mobile interactions.

All the works cited above have one idea in common: the activity modeled as a transaction consists of a pre-defined sequence of operations (typically called a *template* in the workflow literature). However, user activity with the Web often progresses in a "stream-of-consciousness" fashion, where neither the user nor the site knows the which operation will come next in the interaction.

[*]Georgia Institute of Technology, Atlanta, GA
[†]Contact author: 3490 Piedmont Road, Suite 1100, Atlanta, GA 30305; adatta@cc.gatech.edu
[‡]IIT-Bombay, Univ. Massachusetts-Amherst

In the context of the user recovery problem, we can describe a user's interaction with one or more Web sites using the metaphor of a transaction; however, we must relax the restriction that the "transaction" models an interaction with a pre-defined sequence of operations. Rather than using pre-defined templates, a recovery mechanism should *observe* the sequence of operations as they occur, logs a "state" corresponding to each operation, saves these "states" in a log, and utilizes these "states" to recover the user to a useful point in his interaction.

2 Why not use existing techniques to solve this problem effectively?

This problem is difficult to solve using existing technology, primarily due to the need to efficiently store recovery information in a location accessible to a recovery protocol. We now consider several alternative approaches based on existing technologies to solving this problem and demonstrate why these are insufficient. In particular, we examine in detail the use of persistent cookies and server-side logging. Both these approaches utilize *persistent cookies*, which allow the storage of a small amount of information in the client's hard disk (in the *cookies.txt* file). Typically, Web sites use this feature to store information for identifying users over multiple visits to the site, perhaps a unique user identification string and an encrypted login and password.

1. Use persistent cookies to store recovery information. Here, we store the user's session id, as well as enough other information to recover the user to a useful interaction state, in a persistent cookie on the client's machine; that is, after each action, all this information would be written to disk in a new cookie. The client must store all the information needed to recover his interaction in the cookie file on his local persistent storage, since all memory-resident data related to his interaction could be lost during a disruption.

The first problem with this approach arises from the limits imposed on persistent cookies in the standard: persistent cookie files are limited in both the space allotted to a single cookie (4KB), and the number of cookies that a particular domain can set for a given user (20 per domain) [4]. Storing recovery information at the client end in this format is likely to require multiple cookies, since recovery information must be stored for each click. Given this, and the fact that many sites use several of their "allotted" cookies to store user-identification and profile information, it is likely that a recovery protocol utilizing persistent cookies may in some cases run out of "cookie space", resulting in the inability to store recovery information, and potentially preventing recovery due to the lack of needed data.

Aside from the issues with space, there is a second problem with storing recovery information on the client side. This arises due to the need for recovery information to be flushed to disk after every click. Here, every user action introduces the need to write to persistent storage, thereby introducing significant disk latency on the client side.

2. Store a persistent cookie containing only a session id on the client, with the remaining information needed for recovery stored on the web site. Here, a simple session id is not enough to recover a user; the user must be able to submit not only the relevant session id, but also an appropriate HTTP request, as well as other session-specific information (such as would be contained in hidden fields) in order to re-establish his session with a Web site.

In this case, the interaction information must be stored on the site's persistent storage (since, in spite of the perceived abundance of memory, transient memory space on web and application servers is typically a hotly-contended resource). This raises a serious performance issue on the site – interaction information would need to be written to disk *for each user click*, introducing an enormous I/O overhead burden. In addition, if a user's interaction spans multiple sites, as is the case with the United Airlines reservation system described above, recovery information will be spread across multiple autonomous servers, leaving open the question of how a recovery protocol would handle such distributed recovery data.

It is clear from the above discussion that a persistent cookie-based solution, even with some help from server site(s), will not address the practical problems associated with interaction recovery.

2.1 The Interaction Recovery Problem

We now discuss the overall scope of the interaction recovery problem by identifying several interesting subproblems. In the context of a specific user interaction, a user interacts with one or more Web sites through a sequence of actions aimed at achieving a specific goal or set of goals. Since these characteristics resemble those of a transaction, in what follows, invoking the transaction metaphor, we describe user interaction with a Web site as an *Internet Transaction* (**iTX**). Therefore, here, we are concerned with recovering users' **iTXs**. To design a recovery mechanism to mitigate this problem, the following points need to be addressed:

1. How can we describe and capture a user's "state" during the course of an iTX? Determining the components of a user's state is a prerequisite to understanding exactly what it is that we are recovering for a user. What is needed here is a usable and implementable model of interaction states of users, coupled with a set of data structures and algorithms for capturing this state in an operational setting.

2. To which state should a user be recovered? Given a set of user states produced by a user's **iTX**, how can we decide to which state to recover a user? Ideally, we would simply choose the user's most recent state before

failure. However, this is not always possible or correct, since a user's state may have become invalid, for instance, if the data in the underlying database is modified between failure and reconnection. While these problems are not specific to connection disruptions, they *do* impact the design of recovery mechanisms, and need to be addressed in this light.

Another complexity that arises is the potential for multi-objective interactions in an **iTX**. Here, the sequence of user actions as observed by a Web site (or multiple sites) may not correspond to a linear sequence of actions toward a single objective, but rather may consist of interleaved progress toward multiple user objectives. Consider, for instance, the example of purchasing an airline ticket online. Here, after entering his preferred travel plan, but before confirming the purchase of his ticket, our user might check his frequent flier mileage. Effectively, in the same interaction session, the user is performing two tasks: (a) mileage checking, and (b) ticket purchase, where, the mileage-checking action is not directly related to the ticket-buying action. The recovery protocol must deal with the complexities arising from a user expressing more than one objective in the same **iTX**.

3. How can we actually recover users after a failure? Given a means of describing **iTXs** and user states, how can we use this information to recover a user to a useful state in his **iTX**? Here, we need to identify the necessary features and properties of a recovery protocol, taking into account architectural considerations of wireless connections to the Internet. Based on these requirements, we can proceed to define the actual data structures and algorithms for the recovery mechanism, and define how this mechanism will operate in the overall mobile Internet architecture.

3 Conclusion

With the expansion of Web sites to include business functions, a user interfaces with e-businesses through an interactive and multi-step process, which is often time-consuming. A loss of connection, or any other system failure, can result in the loss of work accomplished prior to the disruption. This work must then be repeated upon subsequent reconnection – often at significant cost in time and computation. In some environments, such as wireless scenarios, this "disconnection-reconnection-repeat work" cycle may cause web clients to incur substantial monetary as well as resource (such as battery power) costs as well. In this paper, we proposed a protocol for "recovering" a user to an appropriate recent interaction state after such a failure. The objective was to minimize work that needs to be redone upon restart after failure.

Whereas classical database recovery focuses on recovering the system, i.e., all transactions, our work considers the problem of recovering a particular user interaction with the system. This challenging recovery problem encompasses several interesting subproblems: (a) modeling user interaction in a way that is useful for recovery; (b) characterizing a user's "recovery state"; (c) determining the state to which a user should be recovered; and (d) defining a recovery mechanism. To address these, we began with two foundation notions: (a) **iTX**s, which are sequences of user actions with one or more Web sites, and (b) user-site interaction states, which represent the information exchanged between the user and a Web site during in the course of a single user action on the site.

References

[1] R. Agrawal and D.J. DeWitt. Integrated concurrency control and recovery mechanisms: Design and performance evaluation. *ACM Transactions on Database Systems*, 10(4):529–564, 1985.

[2] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor, and C. Mohan. Advanced transaction models in workflow contexts. In *Proceedings of the International Conference on Data Engineering*, pages 574–581, 1996.

[3] P.K. Chrysanthis and K. Ramamritham. Acta: A framework for specifying and reasoning about transaction structure and behavior. In *Proceedings of the 1990 ACM SIGMOD Conference*, 1990.

[4] Netscape Communications Support Documentation. Persistent client state http cookies. http://home.netscape.com/newsref/std/cookie_spec.html, 2001.

[5] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the 1987 ACM SIGMOD Conference*, 1987.

[6] U. Varshney and R. Vetter. Emerging mobile and wireless networks. *Communications of the ACM*, 43(6):73–81, 2000.

[7] G.D. Walborn and P.K. Chrysanthis. Pro-motion: Management of mobile transactions. In *Proceedings of the Symposium on Applied Computing*, pages 101–108, 1997.