PERFORMANCE ANALYSIS OF ALGORITHMS FOR SUPPORTING DISCONNECTED WRITE OPERATIONS IN WIRELESS WEB ENVIRONMENTS

NGOC ANH PHAN

A Master Thesis
Submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in Partial Fulfillment of the Requirements for the Degree of

Master of Science
in
Computer Science and Applications

Ing-Ray Chen, Chair

Athman Bouguettaya

Denis Gracanin

November 1999

Virginia Politechnic Institute and State University

ABSTRACT

PERFORMANCE ANALYSIS OF ALGORITHMS FOR SUPPORTING DISCONNECTED WRITE OPERATIONS IN WIRELESS WEB ENVIRONMENTS

Ngoc Anh Phan

M.S. thesis directed by Dr. Ing-Ray Chen

A mobile user may voluntarily disconnect itself from the web server to save battery life and also to avoid the high communication price. To allow web pages to be updated while the mobile user is disconnected from the web server, updates can be staged in the mobile unit and propagated back to the web server upon reconnection. In this thesis, we investigate methods for supporting disconnected write operations and develop a performance model which helps identify the optimal length of the disconnection period under which the cost of update propagation is minimized. We validate the analytic model with simulation in the thesis. We also show how the result can be applied to real-time web applications with a deadline requirement to propagate updates of web pages. The analysis result is particularly applicable to web applications which allow wireless mobile users to modify web contents while on the go. The algorithms that we have developed can be generally applied to other data items such as files and databases.

Index Terms — Wireless mobile systems, web access, disconnected operations, performance analysis, coherency interval, hoarding, reintegration.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to all those who helped and supported me throughout various phases of this thesis:

To Dr. Ing-Ray Chen, the thesis director, who provided me the opportunity to do research at Virginia Tech, suggested the topic for this thesis, and provided valuable ideas and assistance throughout my research work and the writing of this thesis. It was a great pleasure to work with him, and without his tremendous help this thesis would not be a reality.

To Dr. Athman Bouguettaya and Dr. Denis Gracanin for serving on my thesis committee and providing valuable suggestions to this thesis.

To Glenn Chinery for proof reading and providing useful advice to this thesis.

To Virginia Tech Computer Science Department faculty and staff members, my friends and my family who gave me a lot of encouragement to finish this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Disconnected Web Operations

The objective of this thesis is to investigate methods for supporting disconnected write operations in wireless web environments. We begin with developing a performance model which helps to identify the optimal length of the disconnection period under which the cost of update propagation is minimized, then we validate the analytic model with simulation. We show how the result can be applied to real-time web applications in wireless environments with a deadline requirement to propagate updates of web pages.

A mobile unit (MU) performing web access can voluntarily disconnect itself from the server to save its battery life and avoid high communication prices in wireless networks. Before disconnecting, the MU can prefetch into its local cache frequently used web pages (or resources) selected based on the MU's specification or the history of past web access operations. While disconnected, the MU accesses only these prefetched web pages. In addition to supporting read-only web browsing, write operations to these prefetched pages can also be performed by recording new values into a log. When the MU is reconnected to the system, the operational log entries

can be reintegrated back to the server to resolve conflicts with updates performed at other sites. In the literature, the three phases of supporting disconnected operations are termed *hoarding*, *disconnection* and *reintegration*, respectively [2]. In this thesis we are concerned mainly with web pages. However, the algorithms developed can be generally applied to other data items such as files and database tables.

1.2. Mobile Architecture

The World Wide Web has become a global distributed information system with a rapidly increasing users community. Many predict a new emerging, gigantic market where millions of mobile users will carry mobile units such as small, battery powered, and handheld terminals, capable of communicating over a wireless connection (for example, Palm Pilot VII [1]). The advent of mobile computing, coupled with the recent developments in wireless communications and personal computer technology, has created a new challenging area of research in mobile information access. Accessing web based information sources is likely to be one of the most important applications of mobile computing. Providing system support to wireless information services is important for the design of wireless communication networks.

In general, the network configuration consists of fixed backbone networks that communicate with a number of MUs. To facilitate a continuous coverage for MUs, the backbone network will have to be augmented with *mobile support stations* (MSSs) that are capable of directly communicating with MUs within a limited geographical area or cell [2]. Figure 1.1 presents the corresponding architecture of a wireless environment. Each MSS is equipped with a transmitter and receiver and provides coverage in one wireless cell. Two basic communication channels are

provided: the *uplink* channel from the MU to the MSS and the *downlink* channel from MSS to the MU. The available bandwidth in each outdoor cell will be severely limited. For example, the bandwidth for wireless radio is only 9.6 Kbps. On the other hand, a wireless LAN designed for indoor use has a bandwidth of 2 Mbps and possibly more in the near future. Thus the wireless bandwidth outdoors is at least two orders of magnitude less than what is available indoors or fixed networks. The low bandwidth in outdoor wireless networks is one of the key constraints that has to be taken into consideration in the design of wireless information services.

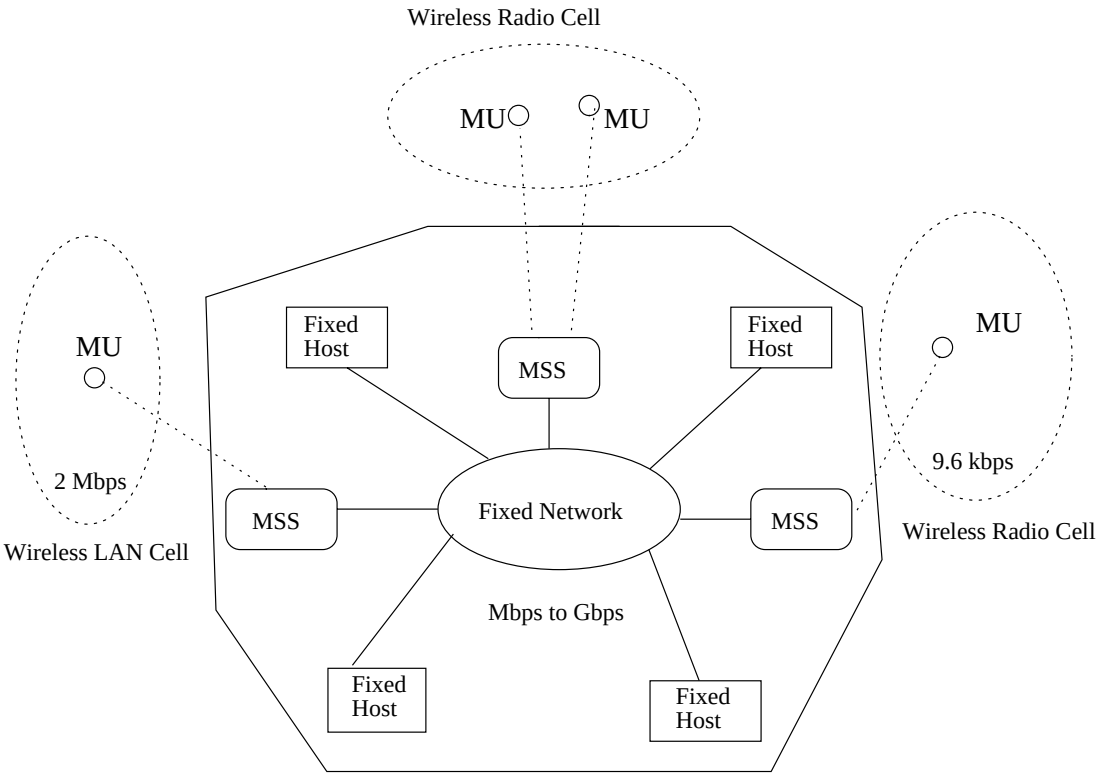

Fig. 1.1. Mobile Networks Architecture

For portability reasons, mobile units run on small batteries. The lifetime of a

battery is expected to increase only 20% over the next 10 years [5]. Thus power conservation is a key issue in designing access methods for wireless information services. Such battery power restrictions will cause MUs to be frequently disconnected, either voluntarily or forced. Mobile units can also reduce energy consumption by operating the hardware in *doze mode*, in which the clock speed is reduced and no user computation is performed. Mobile units return to normal operation upon receipt of a wakeup message. If a MU is totally disconnected, then any updates send by the server will not reach the MU during this period, and any update at the MU during this period is tentative. Hence, mechanisms are needed to allow MUs to propagate updates upon reconnection without any lost update. Also, mechanisms are needed to determine when MUs should reconnect to the server to save communication costs and reduce wireless channel contentions.

There are three models that are used currently in the design of a wireless web browsing system: client/server, peer-to-peer, and mobile-agent [2].

Client/Server Model

The simple client/server model (Figure 1.2a) only requires a web browser to be located at the MU and communicate directly with the web server located at the fixed network via wireless communications. Any browsing optimization requires changes to the client or server code.

The client/agent/server model (Figure 1.2b) implies that a *server proxy* (gateway) is created on the fixed network to communicate with the MU. The server proxy on the fixed network can serve multiple MUs, in which case each MU has to register with the proxy. All web related traffic flows to and from the MU go through the server proxy. The optimizations performed by the server proxy are mostly to

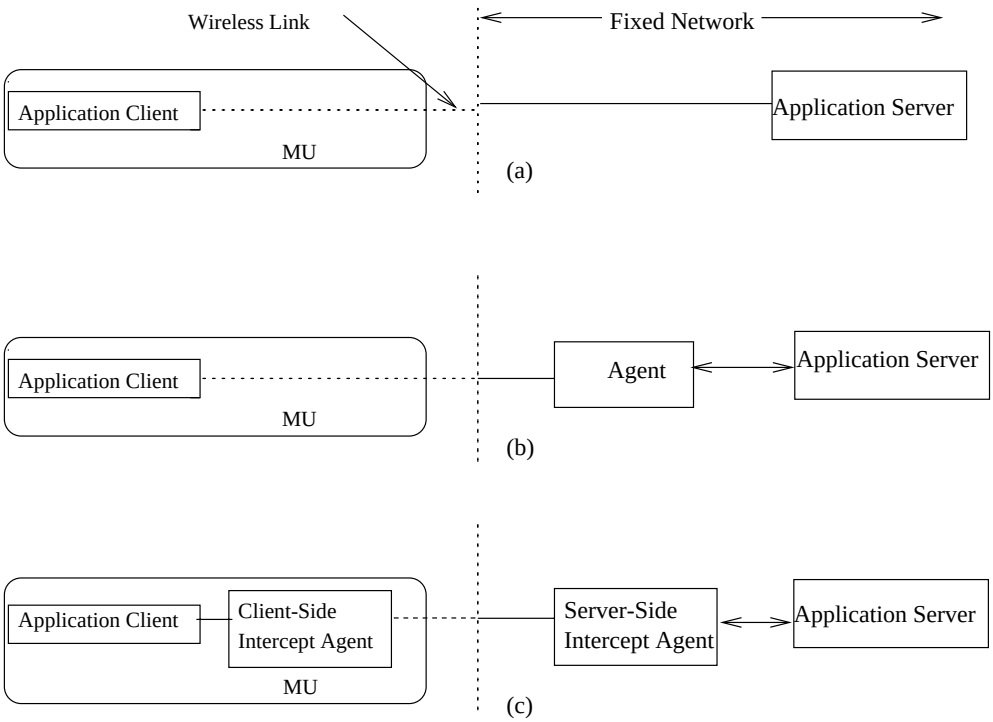

Fig. 1.2. Three Client-Server Based Models: (a) Simple Client/Server Model, (b) Client/Agent/Server Model, and (c) Client/Intercept/Server Model.

reduce the client or server computational load and to minimize the communication overhead from the server to the MU.

The client/intercept/server model (Figure 1.2c) introduces a web-specific *client-side intercept* agent residing at the MU, in addition to a web-specific *server-side intercept* agent at the fixed network. While the server-side agent at the fixed network serves multiple MUs, the client-side agent is unique to the MU. The agent pair cooperates to intercept and control communications over the wireless link for reducing the traffic volume. The pair also optimizes the communication protocol to reduce the latency.

Peer-to-Peer Model

In a peer-to-peer model, MUs are considered equal to the fixed hosts and capable of supporting both web browsers and web servers with locally stored HTML pages. To support such a model, the MU must be heavyweight with enough resources and computational power. Requests submitted by local browsers for URLs locally available are serviced without any network interaction and thus cause no performance problems. However, remote requests to the local server originated from browsers located at other sites or local requests for URLs that do not reside at the MU involve the wireless link. In the first case a server-side intercept agent needs to be placed at the MU to service requests originated from remote browsers. In the second case, a client-side intercept agent is required to facilitate browsing.

Mobile Agent Model

Mobile agents can be used along with the previous models. The main flexibility of mobile agents is the ability given to web agents to create and send a mobile agent around the net to collect information. The mobile agent proceeds asynchronously with minimum interaction with the web agents. Another way that mobile agents can be used is by making the server-side web agent mobile so that it follows its roaming clients. However benefits of such an approach are not clear, since the agent serves many clients.

1.3. Literature Survey

Over the past few years, various schemes have been proposed to support disconnected web access in mobile wireless environments in the three phases mentioned earlier in Section 1.1 [3, 4, 5, 6, 7, 12, 13, 14, 16, 18]. A general assumption for those schemes

is that web pages are updated relatively infrequently. Therefore, most existing schemes assume read-only data access during disconnection. However, with the rapidly increasing of distributed web authoring and form-based electronic commerce web applications [7], there is a need to support write operations. The rest of this section provides a survey of earlier work that has been done in the area of supporting mobile access to the web.

Liljeberg et al., in the Mowgli project [3], propose a proxy model to support disconnected web access. The model consists of two components, a Mowgli Agent and a Mowgli Proxy, located at the MU and the MSS, respectively. They communicate with each other through a private protocol that reduces the number of round-trip communications between the client and the server. A specialized transport service, the Mowgli Data Channel Service, is used to provide reliable communication between the MSS and the mobile host. The Mowgli architecture reduces the data transfer over the wireless link and the communication cost in three ways, i.e., data compression, caching, and intelligent filtering of the document, before sending it to the client. Disconnected mode for the web is supported by using a cache maintained by a Mowgli agent. The issue of cache coherency is discussed but no solution is provided. Also the Mowgli architecture supports read-only data during disconnection.

In [4], Joshi, Weerawarana and Houstis propose a system with two proxies – Moser and WebIQ – residing at the MSS to support disconnected browsing in a wireless environment. Moser is an active transcoding proxy that allows a mobile user to set his or her viewing preferences, based on the network connection and available resources, and performs active transcoding of data on both upstream and downstream traffic to present web information to the mobile user according to the QoS

parameters set by the user. Various transcoding approaches such as transcoding of HTTP requests, image files, video data, active contents, and HTML files are being used to save communication costs associated with retrieving web pages. WebIQ is a proxy that supports asynchronous operations. It manages disconnections and saves communication costs by providing the user the right information through information filtering and collaborative information retrieval techniques. WebIQ splits browsing into a set of asynchronous operations and maintains the state information while disconnections occur. These operations are requesting information, retrieving the results of the request and rating the URLs obtained as a result of the request. The rating information is the user feedback to the system which allows the system to infer a semantic match between the user's query and the information in the document. The state information consists of data about the user, the user's queries, the obtained URLs and the results of the queries not yet rated by the user. A connection is required only during the execution of these operations. Thus there are various points of disconnection in the system, which would allow the user to reconnect and proceed further. However, the issue of when the user can reconnect is not discussed. Also the system mainly deals with read-only data during disconnection. The issue of writing data during disconnection is not addressed.

Imielinski et al., in their Dataman project [5], discuss how a mobile user may access web servers and pages, whose contents depend on the user's location, such as yellow pages, traffic reports, shopping information, local advertising, directions, etc. The information service is organized as a collection of *pages*, with a hyperlink-based "Mosaic like" interface on the client. The set of locations in which the contents of a page are valid is termed its *scope*. The scope of a page can vary substantially

depending on its contents, and in many cases mirrors the geographic region in which it is valid. A tool is designed to allow a user to create new scopes and associate documents with those scopes. The user interface located on the mobile client displays the location-dependent information to keep track of whether the information is valid for the current location or not. If a document is not valid, the user interface will query the server for a corresponding document which is valid for that location. To support disconnection, the project has advocated a stateless solution in which the server does not maintain any information about the state of the clients but rather periodically multicasts an invalidation report which provides the timestamp at which each data item was last changed. When a mobile user reconnects, at some points it will get an invalidation report and can check to see if a cached web page is still valid or a fresh copy must be retrieved. However, this depends on the frequency of the broadcasting message and could result in some idle periods during which the mobile user has to wait for an invalidation message. This scheme also supports read-only web pages.

The eNetwork Web Express described by Floyd, Housel and Tait [6] is a dual-proxy architecture that supports disconnected wireless web browsing. It facilitates the use of web technology to run typical commercial transactions processing applications over wireless networks. The proxy model consists of two proxies: a client side interceptor that runs in the client's mobile device and a server side interceptor that runs within the wired network. While disconnected, the eNetwork Web Express relies solely on the cache at the mobile hosts. It uses a coherency interval (CI) associated with a web page to specify how often a cached page should be checked for changes, e.g., one day for text pages and no limit for graphics. The coherency

interval of a cached page is checked at the access time to see if the cached page should be used, or a fresh page is to be fetched from the server. Various differencing and digital signatures, protocol reduction and header reduction techniques have been used to reduce the bandwidth requirement of supplying fresh pages to the MU. These optimization schemes, however, were mainly used for supporting read-only web access by the MU without considering the possibility of write operations. Also, the concept of coherency interval applies to individual cached items. Thus, when a cached page's lifetime exceeds its coherency interval, the MU must reconnect to the web server to fetch a new copy. Consequently, the MU may be connected and then disconnected while it accesses a set of prefetched cached pages. Hence, the relationship between the length of the disconnection period and the coherency intervals of individual cached pages does not exist.

With the proliferation of distributed web authoring and form-based electronic commerce web applications, a recent trend is to support web page write/create operations during disconnection. Mazer and Brooks in their Caubweb project [7] addressed how to modify contemporary web browers/servers to support disconnected updates to web pages. The basic idea is to support disconnected updates via a HTTP (Hypertext Transfer Protocol) client proxy running on the MU side to cache staging updates while disconnected, and a PUT script running on the HTTP server to accept PUT requests from the HTTP client proxy upon reconnection [1]. A MU can update a page into its cache at any time while keeping two versions of the page in the cache to facilitate differencing, i.e., each MU keeps in the cache the original

[1]The PUT method in HTTP was intended to permit content to be associated with a URL on its origin server. PUT is not natively implemented by many web servers, but the method may be added by a server-specific CGI script.

version that is prefetched before disconnection and the latest modified version. Also, each modified page can be associated with an update policy to determine whether changes are propagated to the web server upon reconnection. Although the issue of conflict detection for avoiding parallel updates is noted in the Caubweb project, no solution is provided. Also, the assumption made in their work is that all updates will be reintegrated to the server upon reconnection. The issue of when the MU should be reconnected to the server is not addressed.

Recently, a web distributed authoring and versioning (WEBDAV) working group under the Internet Engineering Task Force (IETF) has been formed to specify HTTP enhancements to support collaborative authoring of web content [10]. Of particular interest in the context of supporting disconnected write operations is the notion of versioning through which each PUT request indicates the original version of the web page from which its new revision is derived. If the web page has been updated by the server, the update request is denied. The client can then retrieve a new page afresh and at the same time lock the web page to ensure no parallel updates. Locks can be issued for multiple web pages on the same web server to ensure an atomic update [2] be done on these web pages. Although WEBDAV provides a possible solution to prevent the lost update problem, it is not clear when update requests should be propagated to the web server in order to satisfy some design goals such as minimizing the bandwidth requirement for propagating updates, or maximizing the update acceptance ratio. This thesis specifically addresses the issue of when a mobile user should reconnect itself to the web server to propagate updates to minimize the communication cost.

[2]An atomic update is a unique update

The Coda file system [8] developed by Carnegie Mellon supports disconnected update operations. An important objective of Coda is to support mobility. Mobile users can run on Coda simply using the standard Unix file system interface. To support disconnected operations, Coda uses a cache manager called Venus that acts as a client-side agent. In the hoarding phase, Venus ensures that critical objects are cached. An optimistic replica control strategy is employed to allow updates to be done asynchronously. All updates made during the disconnection period are stored in an operation log implemented on top of a transactional facility. Upon reintegration, Venus resynchronizes its cache with the server. If Venus detects a divergence, an application specific resolver (ASR) is invoked to resolve the difference. If the ASR fails to resolve the difference, then a manual repair tool running on the client side is invoked. Our thesis also assumes optimistic replica control in that we allow mobile users to perform updates on the cached web pages during disconnection. Thus, our system model closely follows Coda. The goal of our thesis, however, is to address the issue of when the mobile unit should reconnect to the server so as to minimize the communication cost over the wireless link for update propagation.

1.4. Motivation

The recent trend in the mobile wireless environment is that the number of mobile users increases rapidly whereas the available bandwidth for wireless channels increases with a much lower speed and remains as a scarce resource. It is estimated that in 3-5 years, 20-50% of all Internet nodes may be wireless mobile terminals [20]. It is also anticipated that the future collaborative environment is characterized by mobility. A field study of a distributed product design team headed by Bellotti and

Bly [21] shows that most team members are rarely found at their individual desks. This indicates that mobility must be considered in resource sharing and communication. One possible solution concluded from this study is to use portable mobile units with wireless communications capability to facilitate collaboration among team members. However, this solution was not implemented as it was not supported by the existing software architecture. Hence, the future mobile environment must shield any data processing complexity and channel contention from these mobile clients, thereby supporting the notion of distributed collaboration among mobile and fixed users. Our motivation from the user perspective is to provide multiple mobile users the ability to do cooperative creative works simultaneously with the least restriction and communication cost. From the network perspective, our motivation is to optimize the channel allocation to mobile users and thus optimize the use of wireless resources.

1.4.1. Applicability of the Proposed Scheme

Next we discuss some application scenarios that could benefit from our proposed scheme which will be describe in chapter 2.

1.4.1.1. WEBDAV Applications

The most common application scenario is the scenario of two people maintaining a web site. The following scenario is drawn from the distributed web authoring domain WEBDAV [22]. It motivates our work.

Scenario: Two People Trying to Change the Same Document

A certain Web site is maintained by two people (we will call them "Jane" and

"Joe"), both of whom make changes on an ad hoc basis. Both people are equipped with mobile units. As is frequently the case, there are a few web pages that are hot points of congestion, even between these two people. Both people have a version-aware web authoring tool that interacts with their web server. Joe fetches and starts to work on a web page. Since the web page is not edited by Jane at that moment, the browser/authoring tool reports that the web server has acknowledged his edit operation, and also the web page he will edit is identical to that which he is viewing. Joe disconnects while working on the web page to save the battery power. Jane meanwhile fetches and begins viewing the same web page and realizes that it has some typos. Jane and Joe are working in a disconnected mode and are not communicating with each other. Therefore, instead of waiting for Joe, Jane decides to fix his typos. When Jane starts to edit the web page, the authoring tool in addition to the acknowledgment of her edit operation also reports that Joe is working on the same web page. Jane makes her changes, reconnects to the server and propagates her updates. The server informs Jane that her changes have resulted in a new, named revision of the web page. Joe, however, has not finished editing the web page and a week later decides to finish it off. He makes his final edits and reconnects to propagate his updates. Joe gets a message indicating that what he edited is no longer the latest version and it needs merging. The authoring tool has the merge mechanisms based on actual differences that allow Joe to resolve Jane's work with his. Having done so, Joe saves the web page again, and this time the new version of the web page is saved.

The scenario described above is supported by the working protocol defined by WEBDAV. However, it lacks the control of when Joe should reconnect to the server

to propagate his updates in order to minimize the bandwidth requirement. Also it does not consider a deadline when Joe should update his cached copies to reduce the rejection probability. This motivates us to develop a scheme which provides the same functionality for distributed web authoring with the goal of minimizing the communication costs or reducing the rejection probability.

Under our proposed algorithm in the thesis, we can determine an optimal disconnection period, after which Joe could reconnect to the server to propagate his updates with a minimal communication cost. This also reduces the chance that merging has to be done if Joe forgets what he was doing and a week later decides to finish the document when Jane during this time has propagated many changes to the web pages. In chapter 4 we will show the communication costs can be reduced by determining an optimal disconnection period in a case study.

1.4.1.2. Online Customer Support

There are application scenarios in which a web site is maintained by several people. Such online customer support applications could benefit from using the scheme proposed in this thesis.

For example, currently many companies, especially nationwide companies, are doing businesses over the Internet and for this reason they maintain a technical document web page that contains information about their products and technical support information. An example of this is the Cisco technical documents web page [23]. The information in this page is divided into subcategories such as Security Advisories, Hardware Technical Tips, Software Technical Tips, Troubleshooting Internetworking Systems, etc. Each of these subcategories contains a set of web pages.

A group of customer-support people are responsible for updating the pages of each subcategory. If a set of pages are maintained by multiple customer-support people and they often travel to different cities or countries they could prefetch these pages into their laptops to work in a disconnected mode when traveling. When sitting in the airplane, each customer support person could modify any subset of pages, and add his/her own tips or advice to solve a particular problem. We aim at determining an optimal disconnection period to minimize the communication cost for propagating such updates to the web server and also to reduce the rejection probability and channel contention.

1.4.1.3. Online Newspapers

Another example of an application scenario for our scheme is a group of journalists updating news reports on an online newspaper web site. Online newspaper services have become very popular today. According to the statistical number provided by Media Metrix, [24] the number of visitors in July of 1999 to top newspaper web sites are 2.7 millions for the New York Times, 2.5 millions for USA Today, 1.7 millions for the Washington Post, 1.0 million for the Los Angeles Times, and 0.98 million for the Wall Street Journal. Each section in an online newspaper web site contains a set of web pages written and modified by a group of journalists to report real-time events. A conventional practice of online journalism is that reporters email their reports to the web page maintainer who will combine the reports of an event and post the final report on the web. This approach puts a lot of work on the web page maintainer and also increases the delay of news to appear on the web. Using our approach, a group of journalists could prefetch a set of web pages to their laptops

and travel to the place where a newsworthy event occurs. At the place of the event, each journalist can work in a disconnected mode and modify a subset of web pages to report the latest news on the event. For example, a journalist could report his/her own observation from different corners of an event or add his/her own comments about some other events. After an optimal disconnection period determined by our algorithm, each journalist reconnects to the server to make the latest news available to the public. By doing this they can minimize the cost associated with using the wireless bandwidth. They also achieve the goal of collaborative authoring without causing too much contention to the wireless channel.

1.4.1.4. Web Advisor for Software Engineers

In education environments multiple simultaneous editors could benefit from using our scheme. We use the software engineering class CS 5704 currently offered in the CS graduate program at Virginia Tech as an example. Students is this class are assigned a group project of creating a *web advisor for software engineers* in a large organization [25]. The software products being developed by this organization are very large (using 100 - 1000 programmers), technically complex (hard real-time mission-critical services with high reliability, reusability, and maintainability) and must be developed at various locations in the United States. To facilitate efficient software development in this environment, this organization must be able to effectively coordinate training, disseminate policy and standards to the development team, and rapidly and securely exchange information regarding progress of the project. The *web advisor for software engineers* will provide a readily available resource repository for individuals within the organization who are engaged in requirements generation

and management, architectural definition, detailed design, design and execution of critical experiments, coding, testing, and verification/validation. It will support distributed engineering, development, testing, and fielding at each engineering site. An example system is as follows: Six software engineers are assigned to work in a team with a schedule limited to ten weeks, and team members are full-time working professionals at different organizations. To facilitate the developing of the system, some team members use mobile laptops to work when they are out of their office locations. The web site is set up with the main pages covering concept explorations, requirements, designs and implementations. Each member is assigned to write a portion of a page. However, each member should revise other people's work, and add or correct information if necessary. Under the traditional practice, information is exchanged by broadcasting email messages to team members. The system is maintained by one webmaster who posts the information on the web and updates the site any time an update message arrives. Obviously, this is not an efficient approach. Under our approach, each member is a webmaster himself. Mobile wireless members can disconnect while editing to save battery life. Desktop members can stay connected. After finishing editing a part of the web page or revising other parts, each member reconnects at the optimal reconnection time to propagate the update. The merge procedure is applied at the propagation time if there is an update conflict. Moreover, each member can be assigned a deadline for his part, before which he has to reconnect to the server to propagate updates. The approach proposed in this thesis can facilitate the efficient development of the web advisor system. Also, it supports the notion of collaborative work with least communication cost and contention.

1.4.1.5. Database Applications

As we have mentioned before, our proposed scheme is applicable to web pages as well as to generic data items such as files and database tables. Next we provide an application scenario of a group of retailers maintaining a set of database.

Many nationwide companies have retailers to distribute their products. Their products are divided into different categories and subcategories. Each category of products is distributed by a group of retailers who often travel. We use Cisco as an example. Cisco has numerous retailers to distribute different models of routers, switches, and other products. Using our approach, each retailer could prefetch a database of available models of a product and their prices into the mobile unit. While traveling, each retailer could modify any subset of the database in a disconnected mode. For example, he/she could change the available number of a product after a customer has tentatively committed to buy it, or tentatively change the price of a product model after negotiating with customers. After an optimal disconnection period as determined by our algorithm, a retailer can reconnect to the server to propagate the update. The benefit from this is obvious – paying a minimum amount on using wireless channels while achieving the goal of collaborative work.

1.5. Contribution of the Thesis

In this thesis we develop and analyze update propagation algorithms that combine the concept of coherency interval for supporting disconnected web browsing and the concept of versioning and locking for supporting disconnected write operations. We develop analytical models together with simulation validation methods with the goal of identifying the length of the disconnection period (after the prefetch phase) so that

the communication costs are minimized during the reintegration phase based on our algorithms. Our analysis method can be applied to a number of web applications, particularly those with real-time requirements.

From the network perspective, when using our algorithms a channel allocation manager could use this information to decide when a mobile user should be reconnected to the network so that the mobile user will stay connected the least amount of time to propagate updates, thus effectively maximizing the channel utilization as the system allocates channels to competing mobile users.

From the user perspective, the power saving achieved by shutting down the radio transceiver may be maximized by choosing the right disconnection interval.

1.6. Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 states the system model. Chapter 3 first discusses several algorithms for supporting disconnected write operations in wireless mobile environments. Then it derives analytical expressions for the communication time required to propagate updates under these algorithms. Chapter 4 analyzes the effect of various model parameters. Numerical examples are given to illustrate the use of our analysis methods and a simulation study is used to validate the results. Some physical interpretations of the analysis results are also discussed in Chapter 4. Finally, Chapter 5 concludes the thesis and outlines possible future research areas derived from this work.

CHAPTER 2

ASSUMPTIONS AND SYSTEM MODEL

2.1. System Model

We assume that a number of web pages will be prefetched and stored in the MU's cache during the prefetching phase. Some of these pages are read-only, while others may be updated by the MU during the disconnection phase if needed to facilitate distributed web content authoring. We assume that a prefetching policy exists to determine which pages are to be prefetched, e.g., based on a prediction algorithm, as proposed in [12].

We assume that at the same time of prefetching, for each web cached page the MU also obtains from the web server some update history information in terms of one general parameter, i.e., the update rate of that web page by all users of the system. This information can be collected by the web server by monitoring the update history of the web page. For a cached web page i, we denote this parameter as λ_i^w. In addition, we assume that the MU has some ideas of how often it is going to perform updates on each cached web page. This information depends on the

urgency level associated with the web page and is largely application dependent, e.g., there could be a deadline by which the MU must update the web page. For each cached web page i, we call this parameter as λ_i. For read-only cached web pages, $\lambda_i = 0$. As λ_i is included in λ_i^w, the inequality $\lambda_i^w \geq \lambda_i$ always holds.

We assume that the web server is located somewhere in the fixed network and is not moved during a web session. The MU communicates with the web server by using a client-side intercept agent via expensive wireless links (to the base station) and thus will voluntarily disconnect itself from the web server after the prefetch phase to avoid expensive communication costs and also to save battery power. When the MU reconnects to the web server, it enters into a reintegration phase during which it propagates all staging updates performed during the disconnection phase to the server. During the reintegration phase, we assume that for each modified page, the MU only submits the differences between the original web page prefetched from the server and the latest version which it modifies, as well as the version number associated with the original version, to the web server.

The web server checks to see if the page has been modified during the MU's disconnection period by comparing the version number of the web page it currently keeps with the version number submitted to it by the MU. If they are the same, the web server will accept the update request and modifies the web page accordingly based on the differences received from the MU; otherwise, the update request of the web page is rejected.

If an update request (by means of a PUT request) is rejected, we assume that the MU will stay connected and will issue a request to write lock the web page. A fresh web page will be retrieved from the web server and the MU will apply a merge

program such as the UNIX diff3 tool [26] to resolve the update differences. After the update is done, the MU will propagate the changes to the web server by means of differencing again and will then release the lock. It is assumed that the MU will stay on-line performing the merge operation and the update propagation in this protocol, i.e., the case in which the MU locks the page and then goes away will not occur. If the MU is forced to be disconnected because of environment changes (e.g., due to roaming), we assume that the lock will be broken by the server after a timeout period. This can be implemented by the server by attaching a timestamp to the lock and releasing the lock after a timeout period expires. The MU upon reconnection will discover that it does not own the lock anymore and will retry again by repeating part of the update propagation protocol. This extra time overhead involved in this extreme case is not considered in this thesis due to its small probability.

If the MU creates a new web page, the new page will always be accepted by the server. Therefore, the MU can propagate the new page any time it likes with the same communication cost for that page. Since the update propagation cost for new pages is the same regardless of when the MU reconnects to the server, the effect it brings to our cost model is a constant. Therefore, in the thesis we need only consider update propagation protocols for the case when the MU modifies existing cached pages.

We assume that a MU communicates with the web server via an intelligent server gateway (the server-side intercept agent) located on the fixed network, e.g., it can be just the base station. Further, the communication time on the fixed network is negligibly small compared with that on the wireless link. This assumption is justified for future high-speed wirelined networks. While it is possible that optimization

schemes based on caching, transcoding and differencing may be used by the server gateway to minimize the volume of data sent over the wireless network, we will assume that two general cost parameters, T_1 and T_2, suffice for our analysis: T_1 is the one-way communication cost of transmitting a packet carrying the differences along the wireless link between the MU and the server gateway; and T_2 is the one-way communication cost of transmitting a simple acknowledgement/reply packet. These two parameters can be estimated by knowing more specific parameter values of the wireless network under consideration. Let s_r be the average size of a simple acknowledgement/reply packet. Let s_o be the average size of a web page. Let p_m be the average fraction of any web page being modified. Let B be the bandwidth of the wireless channel through which the MU communicates with the fixed network. Assume that the bandwidth is the same for both up-link and down-link wireless channels. Assume that the communication time in the fixed network is relatively small compared with that in the wireless network. Then, T_1 and T_2 can be estimated as

$$T_1 = \frac{p_m s_o}{B} \tag{2.1}$$

$$T_2 = \frac{s_r}{B} \tag{2.2}$$

where T_1 accounts for the time for transmitting the PUT update request that carries the version number of the original cached page and the differences between the latest version and the original prefetched version; T_2 accounts for the time for transmitting a reply from the web server to the MU.

For notational conveniences, let $T = T_1 + T_2$, representing the round-trip communication cost for propagating a web page that has been updated by the MU but has not been updated by the server at the reconnection time. Normally, $T_1 >> T_2$,

so $T \approx T_1$.

2.1.1. Update Propagation Protocols

The content of a packet carries the information necessary for executing an HTTP-enhanced protocol between the MU and the web server. The overall communication cost for updating a modified cached web page depends on whether or not the page has been updated by other users, or equivalently stated, by the server.

2.1.1.1. Non-Forced Updates

We assume the following protocol is used when the MU propagates a modified web page to the web server during the reintegration phase. The MU sends the differences between the latest version and the original version, and also the version number of the original version in a PUT request packet. This step takes T_1 time. There are two possible cases following this step:

1. If the web page has not been updated by other users, the web server will send an acknowledgement reply packet (ACK) to the MU to accept the PUT request. In this case, the communication cost for the MU to update the modified web page to the server is exactly $T_1 + T_2$ or simply T. Note that here we assume that both the communication time in the fixed network and the server processing time are relatively small compared with the communication time in the wireless network.

2. If the web page has been updated by other users and thus two versions do not match, the web server sends a rejection reply packet to the MU (which takes T_2 time). When the MU is informed, it sends a lock request packet to the web

server to lock the web page (which takes another T_2 time). Here we assume that a lock request packet carries only a simple command to lock a page so it has the same size as an acknowledgment packet.

On receiving the lock request packet, the web server acts accordingly and also sends the new up-to-date version to the requesting MU via differencing (which take T_1 time). After the MU applies a merge algorithm possibly with some manual inspection to resolve the update conflict based on the new version received (which takes D_m time to be defined below), it sends another request packet (which takes T_1 time to transmit). This request packet carries the differences to the web server along with a command to release the lock, for which the web server acts accordingly and afterward sends a reply packet to the MU to complete the update process (which takes another T_2 time). The overall communication cost in this sequence is $3T_1 + 3T_2 + D_m = 3T + D_m$, where D_m stands for the average time taken to resolve the update conflict by the MU.

2.1.1.2. Forced Updates

Many real-time applications such as stock market and weather reporting systems necessitate *forced updates* when the MU is reconnected to the server, i.e., the MU must propagate its updates to the server at an appropriate reconnection time so as to meet the deadline requirement. This real-time requirement calls for a special handling protocol for those pages that have not been updated by the MU at the time of reconnection and also mandates that the MU be reconnected to the server at an appropriate time prior to the real-time deadline to account for the time needed

to propagate updates.

We assume that if the MU must perform a forced update on a cached web page which has not been updated by the MU at the reconnection time, it will first issue an inquiry packet carrying the web page's version number to the server to check if the cached page is still valid (an operation which takes T_2 time). Then, one of the following two cases will occur:

1. If the server has updated the web page, the server will send the differences to the MU so that the MU can perform its forced update operation based on the new version received. This conditional step takes T_1 time.

2. If the server has not updated the web page, the server will simply send a reply packet to the MU indicating that the MU's cached page is still valid. This conditional step takes T_2 time.

In either case, the particular web page will be locked by the MU to prevent other users from updating it. The MU must subsequently modify the web page (which takes D_m time) while it is on-line, and propagate the differences to the server (which takes T_1 time). The server then will send an ACK to the MU and release the lock (which takes T_2 time). Summarizing above, case 1 will take a total of $2T_1 + 2T_2 + D_m = 2T + D_m$, while case 2 will take a total of $T_1 + 3T_2 + D_m = T + 2T_2 + D_m$.

2.1.1.3. Performance Metric

One objective of our analysis is to identify the disconnection period of the MU when multiple data items exist in the cache in order to optimize the system performance. One possible performance metric is the total communication (or reconnection) time

between the MU and the web server in the reintegration phase during which all updates are propagated to the web server based on versioning and locking mechanisms supported by the web server. Another possible performance metric is the rejection probability, i.e., the percentage of modified web pages being rejected by the web server as the MU attempts to propagate updates to the web server. Conceptually, a short disconnection period will result in a low rejection rate since the probability of a cached web page being modified by other users will be low for a short disconnection period. This, however, may unnecessarily increase the precious battery consumption of the MU, especially if the web page is not modified often by other users in the system. Thus, there is a tradeoff between a low rejection rate and a short disconnection period. Moreover, for real-time applications demanding forced updates at the reconnection time, a high rejection rate implies a large reconnection time since all rejected pages must be forcibly updated by the MU and then all updates must be propagated to the server. This means that the MU must reconnect to the server early enough to avoid a high rejection rate so that it won't spend too much time propagating updates, and as a result missing the deadline. A main goal of this thesis is to find the longest disconnection period so that the total communication (or reconnection) time for propagating updates is minimized, thereby saving the power consumption of the MU. For real-time web applications, in addition to the above goal, we must also satisfy the requirement that the deadline not be missed. We will illustrate how our analysis result can be applied to real-time applications with such a requirement in Chapter 4. Table 2.1 summarizes the parameters which will be used in the thesis.

λ_i	update rate for web page i by the MU
λ_i^w	update rate for web page i by the world
s_o	average size of a cached web page
s_r	average size of an acknowledgement/reply packet
p_m	fraction of a web page being modified by the MU
B	bandwidth of a wireless channel
T	average round-trip communication time between the MU and its web server
D_m	average time to execute a merge algorithm to resolve update conflicts
L	length of the disconnection period
p_i	probability that page i is updated by the MU within a time period of L
r_i	probability that page i is updated by the server within a time period of L
C_i	total time for propagating updates to the web server upon reconnection

Table 2.1. Parameters Used in the Analysis.

CHAPTER 3

MODEL AND ANALYSIS

We first derive expressions for r_i and p_i as defined in Table 2.1. Suppose that the length of the disconnection period is L. Upon reconnection, the MU will attempt to propagate the update of a modified web page. The PUT request will be denied, however, if the web server has modified the web page during L. Thus, r_i, the probability that an update request for page i is rejected by the server upon reconnection after the MU has been disconnected for a period of L, is given by

$$r_i = 1 - e^{-(\lambda_i^w - \lambda_i)L} \tag{3.1}$$

where we assume that updates to page i arrive at the system as a Poisson process, with rates λ_i and λ_i^w by the MU and by the world, respectively.

Since we are interested in estimating the cost of propagating updates to the web server during the reintegration phase, we wish to know the probability that a web page has been modified by the MU at the end of the disconnection phase. Suppose that the length of the disconnection phase is L again. Then, p_i, the probability that page i is updated by the MU within a period of L, is simply given by

$$p_i = Prob\{update\ time\ \leq\ L\} = 1 - e^{-\lambda_i L} \tag{3.2}$$

Below we consider three performance models. The first is for a special case in which the MU caches only a single web page. The second considers the case where the MU has multiple cache items but it propagates one web page update at a time to the server. The third model also considers multiple cached web pages; however, the MU propagates all the updates to the web server in "batch" using as few message exchanges as possible in order to reduce the reconnection time. The third case requires the use of a protocol between the server and the MU so as to pack and unpack individual web page updates embedded in a message.

3.1. Single-Page Update Propagation

We first consider the case in which the MU prefetches only a single cached web page. Here, we apply the concept of coherency interval to determine the best disconnection period for the MU. Recall that under the coherency interval method for supporting read-only web browsing, an access to a cached page is allowed to proceed as long as the access time is within the interval. If the access time is out of the interval, then the MU must reconnect to the server and fetch a new copy if it is different from the cached copy. In the previous work [6], the length of the interval is heuristic in nature, defined arbitrarily either by the system or by the user.

We modify the above protocol to support write operations on a single web page, say page i, as follows:

1. If the time of modifying the cached page by the MU is less than L (relative to the beginning of the disconnection phase), we allow the modification to proceed by recording the differences between the latest version and the previous version in the MU's local cache/log without propagating the update to the server. All

subsequent updates to the cached page, if any, are processed locally until the time reaches L, at which point the MU is reconnected to the web server and updates are propagated by following the protocol described in Chapter 2. In this case, the probability that the PUT request is rejected by the server is r_i as derived earlier. Following the discussion in Section 2, the communication cost of update propagation is $3T + D_m$ with probability r_i, and T with probability $1 - r_i$.

2. If the time of modifying the cached page by the MU is greater than L, that is, the MU has not updated the page by the time L is reached, a forced update is performed at L, i.e., the MU is forced to perform an update at time L. The MU at time L does not know if the cached page is valid or not. To avoid performing the forced update operation on an out-of-date version, it sends a message carrying the version number of its cached page to the server, for which the server responds by locking the page and, if the page has been updated during L, also sends the difference to the MU.

   The communication time in this case thus depends on whether the cached page has been updated by the server or not at time L. It can be obtained by considering the following two subcases:

   (a) The cached page is still valid, i.e., not being updated by others during L. In this case, the communication time is $2T_2 + D_m + T$. The first term accounts for the time for the inquiry and reply messages; the 2nd term accounts for the time to perform the forced update; and the 3rd term accounts for the time for the MU to propagates the difference and to get

an acknowledgement from the server.

(b) The cached page has been updated by others during L and therefore a fresh copy must be retrieved from the server before a forced update by the MU can be performed. In this case, the communication time is $2T + D_m$.

Note that the probabilities of the above two subcases are $1 - r_i$, and r_i, respectively.

Summarizing all of the above, the average communication time involved in propagating the update from the MU to the server for a single web page upon reconnection is thus given by

$$C_i = p_i \left[r_i(3T + D_m) + (1 - r_i)T\right] + (1 - p_i) \left[r_i(2T + D_m) + (1 - r_i)(T + 2T_2 + D_m)\right] \tag{3.3}$$

The above equation can be used to determine the best L value (representing the disconnection period) under which the total reconnection time is minimized.

3.2. Multiple-Page Update Propagation: Algorithm 1

We now consider the case in which the MU prefetches a set of cached web pages, say, based on a prediction algorithm. All updates occurring before L are again staged and later propagated back to the server at time L relative to the beginning of the disconnection phase. Moreover, all web pages not updated during L are also forcibly updated, following the algorithm described in the previous subsection. Again, we are interested in identifying the best disconnection period that will optimize the system performance. In this subsection, we consider the case in which modified web

pages are propagated to the server one at a time. In the next subsection, we will consider the case in which all modified web pages are propagated back to the server in "batch" to further save the communication cost.

Without loss of generality, consider a particular web page, say i, in a set of N prefetched pages. Again, let λ_i and λ_i^w be the update rates on page i by the MU and by the world, respectively. Since the MU propagates one page at a time to the server upon reconnection at time L, the total reconnection time needed for the MU to propagate updates of all N pages to the server is simply given by

$$C_N^{alg\ 1} = \sum_{i=1}^{N} C_i \tag{3.4}$$

where C_i is given by Equation 3.3, standing for the reconnection time needed for propagating updates to page i.

3.3. Multiple-Page Update Propagation: Algorithm 2

Here we also deal with the case in which the MU prefetches a set of web pages before disconnection. However, when the MU reconnects at time L, the server and the MU execute the following algorithm to propagate updates in batch in order to shorten the reconnection time:

1. The MU sends to the server a bit vector A (carrying 0 or 1 values) indicating which cached pages have been updated by the MU, and also a value vector B (carrying integer values) indicating the original version numbers associated with the cached web pages prefetched into the MU before disconnection. This is done by sending a single message from the MU to the server. Here we

assume that for practical web applications the MU typically will only update a few web pages (on the order of 10's at most) so a regular ACK message is sufficient to carry these two vectors. Therefore, the time needed for this step is T_2.

2. The server decides accepting or rejecting page updates based on vectors A and B received from the MU, and the current version numbers associated with the requested pages stored in the server. It then sends a bit vector C indicating which pages can be accepted (for which the value is 0) and which pages are to be rejected (for which the value is 1). Those pages not updated by the MU have their corresponding values also marked with 1 in vector C. The server also sends a separate version number vector D indicating the current version numbers of the pages stored in the server. All the above information is embedded in a single message sent from the server to the MU for the same reason that discussed before. Therefore, the time for this step is also T_2.

3. For those pages not accepted by the server, the server also locks those pages to prevent them from being updated by other users in the system. Simultaneously, the server sends to the MU the differences of those pages rejected by the server (because the server has updated those pages) and also those pages that have not been updated by the MU but have been updated by the server. The time for this step is

$$T1 \sum_{i=1}^{N} r_i$$

because on average $\sum_{i=1}^{N} r_i$ pages (out of N) are updated by the server during [0, L], for which the server must send the differences to the MU so that the

MU can perform forced updates on the newest version.

4. When the MU receives vectors C and D, it knows immediately which pages are accepted by the server. Those accepted pages (i.e., those updated by the MU but not updated by the server during [0, L]) are then sent to the server by means of differencing. The time for this step is

$$T1 \sum_{i=1}^{N} p_i(1 - r_i)$$

because on average $\sum_{i=1}^{N} p_i(1 - r_i)$ pages are updated by the MU but not updated by the server (thus are accepted by the server) during [0,L].

5. The server sends an acknowledgement to the MU after it receives and processes the update propagation for those accepted pages. The time for this step is T_2.

6. The MU then performs forced updates for those pages not having been accepted by the server, including

   (a) pages updated by the MU and also updated by the server;

   (b) pages not updated by the MU but updated by the server; and

   (c) pages not updated by the MU and not updated by the server.

   After the MU receives differences for those pages updated by the server during [0,L], it performs merge/update operations on all of the above three types of pages. Then, it sends differences of the newest versions of all these pages to the server. The time for this step is given by

$$(D_m + T_1) \sum_{i=1}^{N} r_i \ + \ (D_m + T_1) \sum_{i=1}^{N} (1 - r_i)(1 - p_i)$$

where the first term accounts for the time taken to generate and propagate updates for the first two types of pages (i.e., those updated by the server during [0,L]) based on the versions received from the server; the 2nd term accounts for the time taken to generate and propagate updates for the third type of pages (i.e., those not updated by either the MU or the server during [0,L]).

7. The server sends an acknowledgement to the MU after it receives and processes the update propagation for those pages being forcibly updated by the MU in the previous step. All locks are also released in this step. The time for this step is T_2.

The total reconnection time to propagate updates of all cached pages is therefore equal to the sum of all the individual times in steps (1) to (7) above, i.e.,

$$C_N^{alg\ 2} = 4T_2 + \sum_{i=1}^{N} [r_i T_1 \ + \ p_i(1-r_i)T_1 \ + \ r_i(D_m + T_1) \ + \ (1-r_i)(1-p_i)(D_m + T_1)]\,. \tag{3.5}$$

This can be expressed in a simplier form as,

$$C_N^{alg\ 2} = 4T_2 \ + \ r_i T_1 \ + \ D_m \ + \ T_1 \ - \ p_i D_m \ + \ r_i p_i D_m. \tag{3.6}$$

CHAPTER 4

APPLICATION

In this chapter, we present numerical data to demonstrate the applicability of our analysis. We use simulation to validate the analytical results obtained in sections 3.3, 3.4, and 3.5. We study the effect of different parameters $(\lambda, \lambda^w, D_m, p_m, s_o)$ on the optimal disconnection period (L). Three cases are considered:

1. the effect of the ratio between the MU update rate and the world update rate (λ/λ^w)
2. the effect of the average time to execute a merge algorithm (D_m)
3. the effect of the size of differences between two versions $(p_m s_o)$

For each case, we show analytical and simulation results under three algorithms: The single-page update propagation (section 3.3), multiple-page update propagation under algorithm 1 (section 3.4) and multiple page update propagation under algorithm 2 (section 3.5). We provide a physical interpretation of these results in this chapter.

4.1. Analytical Results

4.1.1. Effect of λ/λ_w ratio

To demonstrate the effect of λ/λ_w ratio, we fix $T1 \approx 5$ seconds, $T2 \approx 0$ second, $D_m = 100$ seconds. We arbitrarily select a λ_w value, and vary the λ value from $0.2\lambda_w$ to λ_w.

For the single-page update propagation algorithm, we consider the case where the MU updates a web page for which the world will update it with a rate $\lambda_w = 10$ updates/hour (in Figure 4.1). For the multiple-page update propagation under both algorithms, we consider the case where the MU updates 5 web pages (an arbitrary selection) for which the corresponding λ_w values are arbitrarily selected in the range of [0, 15] updates/hour (in Figures 4.2 and 4.3, respectively).

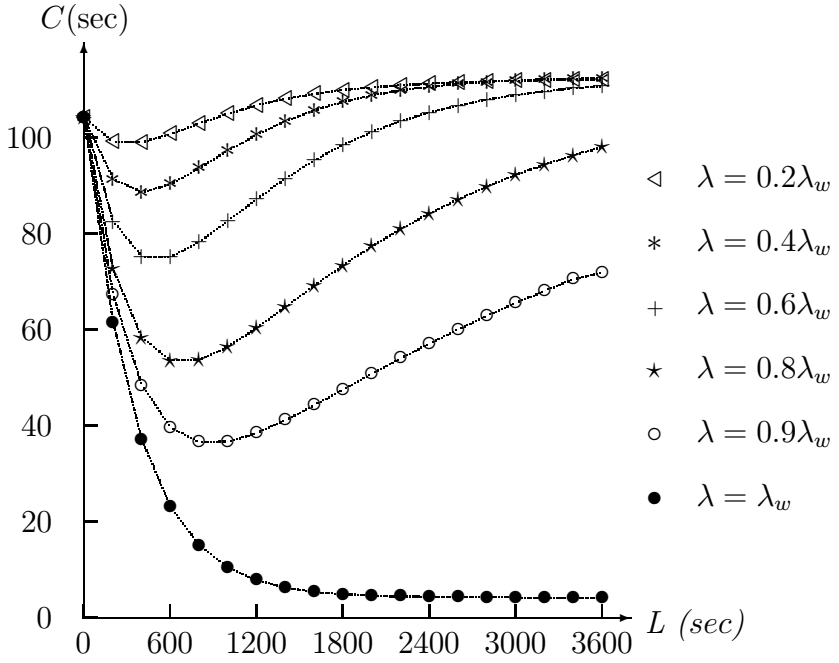

Fig. 4.1. Effect of λ/λ_w Ratio on Single-page Update Propagation Algorithm.

As the MU's update rate λ varies from $0.2\lambda_w$ (the top curve) to λ_w (the bottom curve), we observe that the curves get deeper. This means that when the MU's update rate is closer to the world update rate, the MU can save more communication cost by propagating updates at the optimal disconnection point L. An exception occurs when λ equals λ_w, which means that the page is updated only by the MU. In this extreme case there is no optimal disconnection period, i.e., the optimal disconnection period equals infinity. The optimal disconnection point also shifts to the right as we go from the top curve to the bottom curve. This indicates that the reconnection time interval (L) is shorter when the MU's update rate is far below the world update rate, and it is longer when the MU's update rate is close to the world update rate.

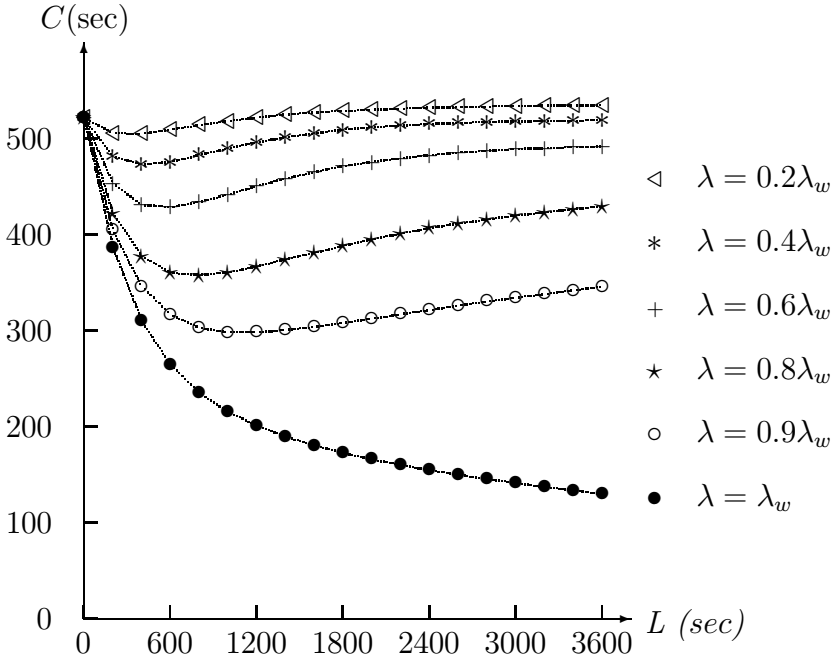

Fig. 4.2. Effect of λ/λ_w Ratio on Multiple-Page Update Propagation under Algorithm 1.

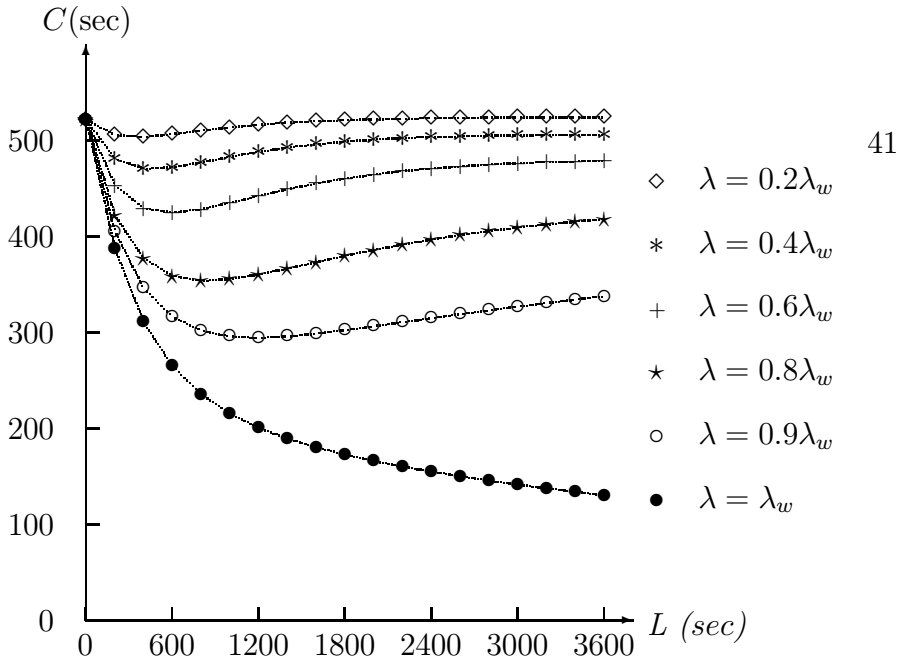

Fig. 4.3. Effect of λ/λ_w Ratio on Multiple-Page Update Propagation under Algorithm 2.

4.1.2. Effect of the Time to Perform Merge Operations

To demonstrate the effect of the time to perform the merge operation we selectively fix the λ/λ_w ratio to 0.8, $T1 \approx 5$ seconds, $T2 \approx 0.1$, and vary the D_m values in the range of 30 to 180 seconds.

For the single-page update propagation algorithm, we consider the case where the MU updates a web page for which the corresponding λ_w value equals 10 updates/hour (in Figure 4.4). For the multiple-page update propagation algorithms under both algorithms, we consider the case where the MU updates 5 web pages (an arbitrary selection) for which the corresponding λ_w values are arbitrarily selected in the range of [0, 15] updates/hour (in Figures 4.5 and 4.6, respectively).

As the time to perform merge operations varies from 30 seconds (the bottom curve) to 180 seconds (the top curve), we observe that for all algorithms the curves get deeper as the D_m value gets larger. This means that the MU can save more communication cost by propagating updates at the optimal disconnection point when the merge operation requires more time to perform.

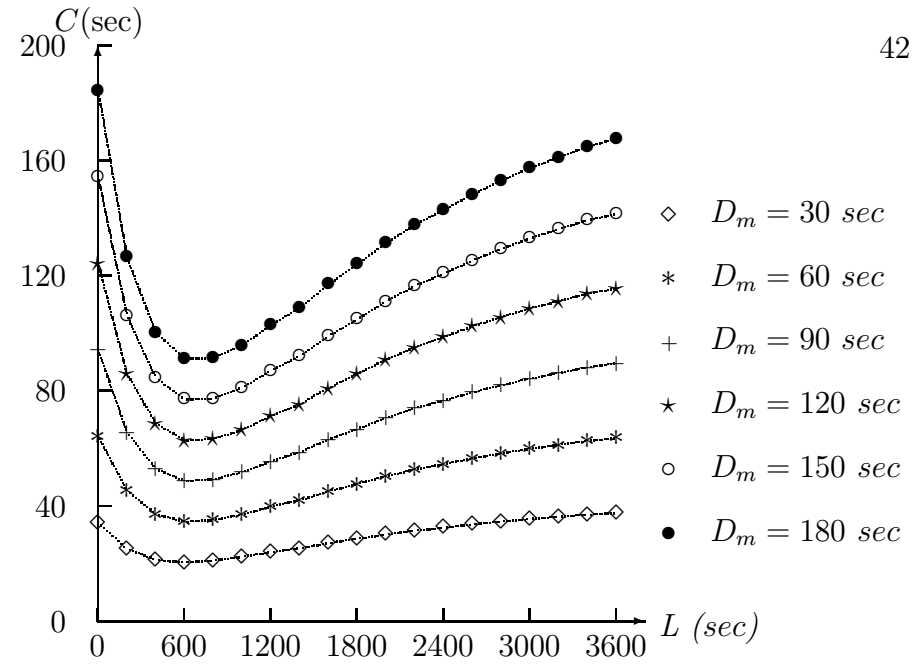

Fig. 4.4. Effect of D_m on Single-page Update Propagation Algorithm.

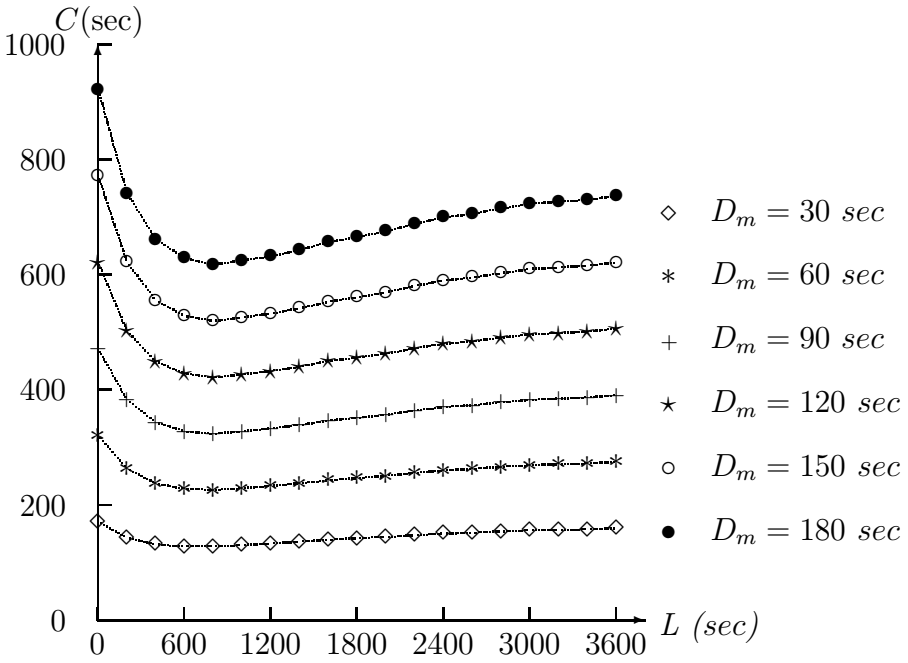

Fig. 4.5. Effect of D_m on Multiple-page Update Propagation under Algorithm 1.

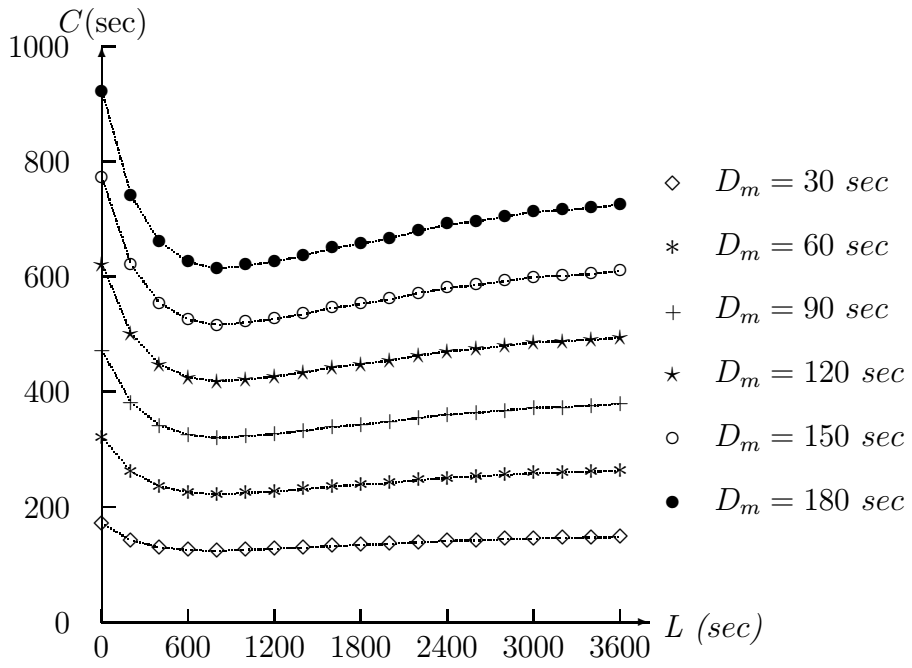

Fig. 4.6. Effect of D_m on Multiple-page Update Propagation under Algorithm 2

4.1.3. Effect of the Size of Differences Between Two Versions

To demonstrate the effect of the size of the differences between two versions, we selectively fix λ/λ_w to 0.8, and the D_m value to 100 seconds. We vary the fraction of the web page modified by the MU (p_m values) in the range of [10%, 50%], and the average size of a web page (s_o values) in the range of [50 KB, 250 KB]. In this case, the size of the version differences ($p_m s_o$ values) will vary from 5 KB to 25 KB.

For the single-page update propagation algorithm, we consider the case where the MU updates a web page for which the corresponding λ_w value equals 10 updates/hour (in Figure 4.7). For the multiple-page update propagation under both algorithms, we consider the case where the MU updates 5 web pages (an arbitrary selection) for which the corresponding λ_w values are arbitrarily selected in the range

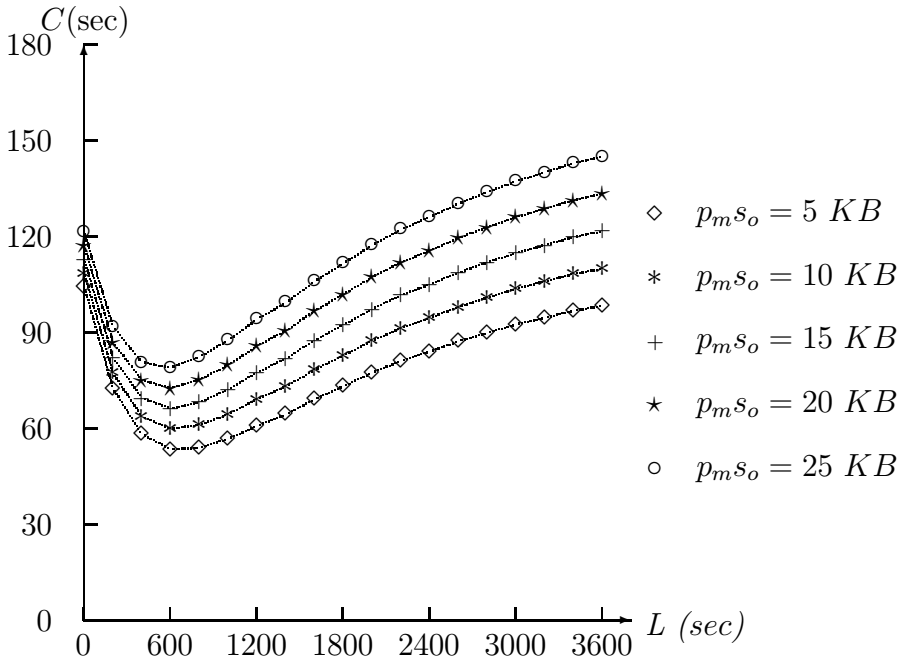

Fig. 4.7. Effect of $p_m s_o$ on Single-page Update Propagation.

of [0, 15] updates/hour (in Figures 4.8 and 4.9, respectively).

As the size of the version differences varies from 5 KB (the bottom curve) to 25 KB (the top curve), we observe that for all cases the curves get deeper as the product of p_m and s_o gets larger. This means that the MU can save more communication cost by propagating updates at the optimal disconnection point when it modifies a larger portion of the web pages or a larger web page.

When comparing updates under algorithm 1 and algorithm 2, we observe that the MU can save more communication cost by propagating updates under algorithm 2. Algorithm 2 is especially effective when the MU modifies a large portion of the web pages or a large web page.

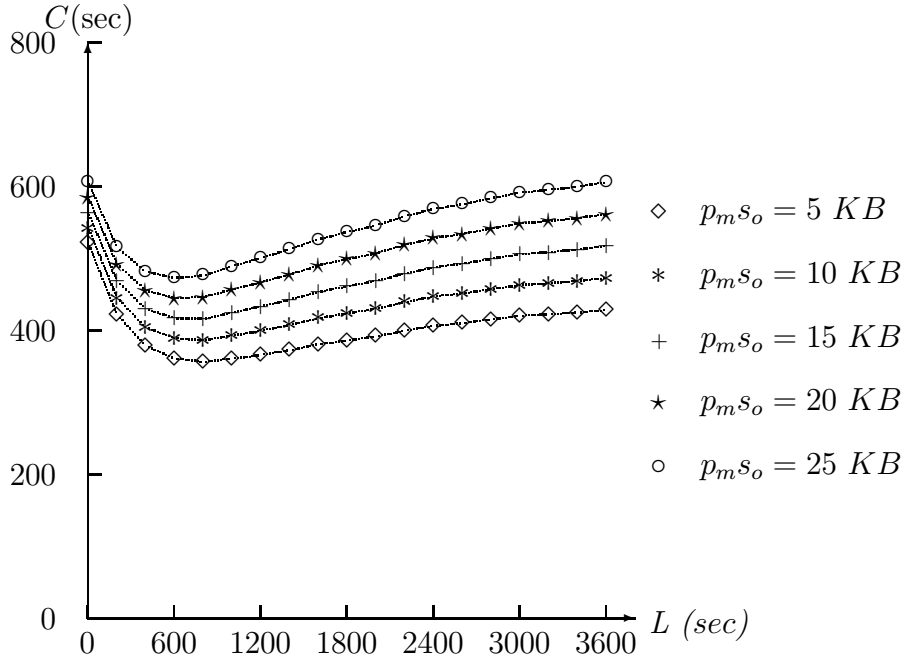

Fig. 4.8. Effect of $p_m s_o$ of Multiple-page Update Propagation under Algorithm 1.

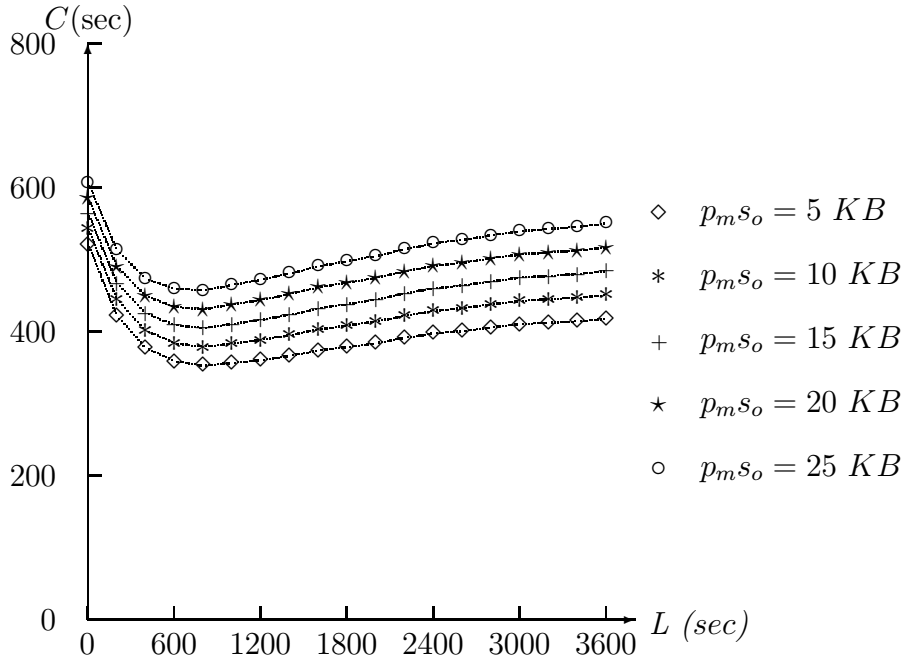

Fig. 4.9. Effect of $p_m s_o$ on Multiple-page Update Propagation under Algorithm 2.

4.2. Simulation Validation

To validate the analytical results presented in Section 4.1, we have developed a simulation model. To simulate update events, we randomly generate the time when an update will occur to page i by the MU (T_i^c) and by other users (T_i^s). We follow the assumption in Chapter 3 that the interarrival time between two updates to a web page is exponentially distributed, although the simulation model can consider any general distribution. Therefore, the average time interval an update will occur at the MU side is given by

$$T_i^c = \frac{-ln(1 - P_u)}{\lambda_i} \tag{4.1}$$

The average time interval an update will occur by other users is given by

$$T_i^s = \frac{-ln(1 - P_u)}{\lambda_i^w - \lambda_i} \tag{4.2}$$

where λ_i and λ_w are the update rates of the MU and the world, respectively, and P_u is a randomly generated real number between 0 and 1.

For each value of L in the x-coordinate, we generate a value of C in the y-coordinate as follows: We compare L with T_i^c and T_i^s and use the protocols described earlier in section 3.3, 3.4, and 3.5 to determine C. The number of states that page i could be in is given by

$$p(3, 2) = \frac{3!}{(3 - 2)!} = 6 \tag{4.3}$$

because there are 6 distinct ways to arrange L, T_i^c, T_i^s (see Figure 4.1 for illustration.)

However, according to our protocol when both the MU and the server (i.e., other users) have updated a web page before the disconnection period, the order in which who updates the page first does not affect the total communication cost.

State 1: $T_i^s > L > T_i^c$
State 2: $L > T_i^c > T_i^s$
State 3: $L > T_i^s > T_i^c$
State 4: $T_i^c > L > T_i^s$
State 5: $T_i^c > T_i^s > L$
State 6: $T_i^s > T_i^c > L$

Table 4.1. Possible States of a Web Page.

This also applies to the case when both the MU and the server have not updated a web page before the disconnection period. Therefore, for each web page there are four possible states:

State 1: The MU has updated the page but the server has not updated the page, i.e., $T_i^s > L > T_i^c$.

State 2: The MU has updated the page and the server has also updated the page, i.e., $L > T_i^c, T_i^s$.

State 3: The MU has not updated the page but the server has updated the page, i.e., $T_i^c > L > T_i^s$.

State 4: The MU has not updated the page and the server also has not updated the page, i.e., $T_i^c, T_i^s > L$.

If the MU updates N pages, the total number of possible states for these N pages altogether is given by 4^N. Table 4.2 shows the possible states and the associated communication costs (C) for the single-page update propagation algorithm. Table 4.3 and 4.4 show examples of calculating C for a 5-page update propagation case

under algorithm 1, and for a 5-page update propagation case under algorithm 2, respectively.

State	Cost
1	$T_1 + T_2$
2	$3T_1 + 3T_2 + D_m$
3	$2T_1 + 2T_2 + D_m$
4	$T_1 + 3T_2 + D_m$

Table 4.2. Communication Costs under Single Page Update Propagation

Web page number	State	Communication cost
1	2	$3T_1 + 3T_2 + D_m$
2	4	$T_1 + 3T_2 + D_m$
3	3	$2T_1 + 2T_2 + D_m$
4	4	$T_1 + T_2 + D_m$
5	1	$T_1 + T_2$
Total communication cost		$8T_1 + 10T_2 + 4D_m$

Table 4.3. An Example of Calculating Communication Costs under Algorithm 1 for a 5-Page Update Propagation Case

We use the batch means analysis method [30] to obtain the average communication cost (C) for a given L value with 5% accuracy level and 95% confidence level.

Step	Cost
1	T_2
2	T_2
3	$2T_1$
4	T_1
5	T_2
6	$4T_1 + 4D_m$
7	T_2
Total communication cost	$7T_1 + 4T_2 + 4D_m$

Table 4.4. An Example of Calculating Communication Costs under Algorithm 2 for a 5-Page Update Propagation Case

To collect data and get the average value of a batch run, we run 2000 simulation runs in one batch run, where each simulation run returns a C value. After k batch runs, we compute the sample mean and the sample variance to determine if the accuracy and confidence levels have been achieved. If not we run another batch run. We use $k = 10$ in the simulation. The sample mean is given by

$$\overline{C} = \frac{\sum_{i=1}^{k} C_i}{k}$$

The sample variance is given by:

$$S^2 = \frac{(\sum_{i=1}^{k} C_i^2) - k * \overline{C}^2}{k-1} \tag{4.4}$$

The confidence level is determined as

$$prob[\overline{C} - H \leq \mu \leq \overline{C} + H] = 1 - \alpha \tag{4.5}$$

where H is given by

$$H = t_{\frac{\alpha}{2};k-1} * \frac{S}{\sqrt[2]{k}} \tag{4.6}$$

The accuracy level is determined as

$$\frac{H}{C} \leq \beta \tag{4.7}$$

The parameters used in the simulation model are shown in Table 4.5.

Of all data generated by our simulation, we observed virtually the same curves shown earlier in figure 4.1 - 4.9. The difference between simulated and analytical data is less than 0.1%. We show an example of the simulation results for the effect of λ/λ_w on the single-page update propagation algorithm in Figure 4.10. This figure is virtually identical to Figure 4.1 obtained earlier from the analytical results

4.3. Applications to Real-time Web Applications

In this section, we apply the analysis result to real-time web applications which must update a set of their on-line web pages periodically. Examples include web pages for market update from Newsweek.com which must be updated about every hour [27]; top news from The Washington Post which must be updated every 3 hours [28]; and America On Line Business Headlines which must be updated every 10 minutes [29]. We consider the case in which these web pages have many fields to be updated by multiple users, some of which may be stationary while some of which may be mobile

T_i^c	a randomly generated average time interval over which an update will occur on the MU side
T_i^s	a randomly generated average time interval over which an update will occur on the server side
C_i	average value of the communication cost in a batch run
$\overline{C}$	sample mean of the communication cost in k batch runs
k	number of batch runs ($k = 10$)
μ	true mean of the communication cost
H	the difference between the true mean and the sample mean
$1 - \alpha$	confidence level (95%)
β	accuracy level (5%)
$t_{\frac{\alpha}{2};k-1}$	t-distribution based on $\alpha/2$ and $k-1$ [30]

Table 4.5. Parameters Used in the Simulation.

so as to collect the needed information on the go. Regardless of who is performing the update, there is a real-time deadline by which these web pages must be updated. We denote this deadline by t_R whose magnitude depends on the application. For stock web servicing applications, this deadline may be in the order of seconds. For weather broadcasting applications, this deadline may be in the order of minutes.

Our objective is to show numerical data indicating the optimal disconnection period, L_{opt}, as a function of model parameters. If this optimal disconnection time plus its associated update propagation time is less than the deadline, then the MU should reconnect to the server at L_{opt}, to reduce the reconnection time so as to reduce

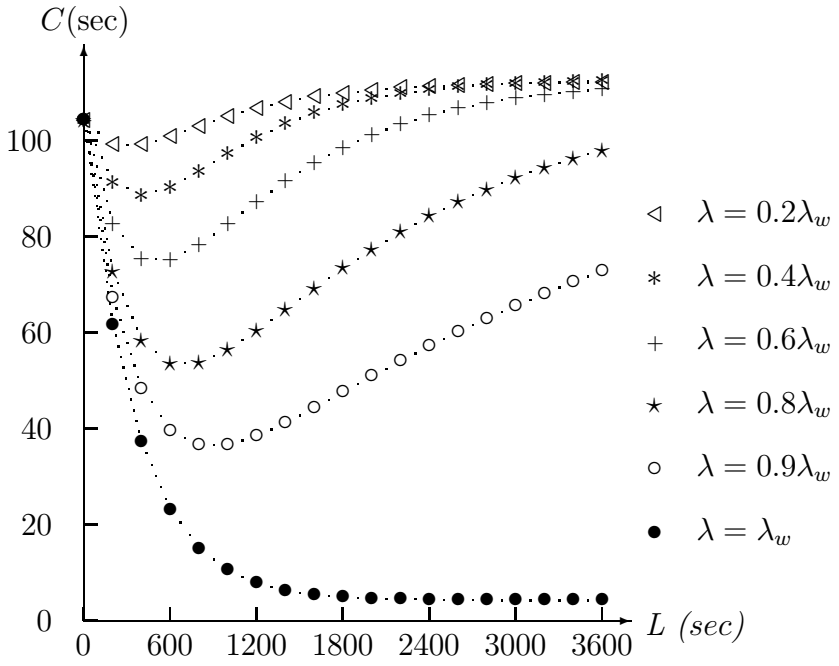

Fig. 4.10. Simulation Data for the Effect of λ/λ_w Ratio on Single-page Update Propagation Algorithm.

the communication cost and power consumption; otherwise, it should reconnect to the server at the time point less than L_{opt} such that the deadline is equal to the sum of the selected disconnection time (less than the optimal point) and the associated update propagation time (as a result of selecting the disconnection time) so as to avoid deadline violation. In all numerical results presented so far we have used x-y diagrams showing the relationship between the selected disconnection time period L (the x-coordinate) vs. the associated reconnection time C (the y-coordinate) needed for update propagation based on that selection, thus giving an estimate of the reconnection time needed for update propagation when given a disconnection period. Specifically, let (L, C) denote any point in a x-y diagram and let (L_{opt}, C_{min})

denote the optimal L point at which the cost is minimum. For real-time applications with a deadline of t_R, if $L_{opt} + C_{min} < t_R$ then L_{opt} is the disconnection time period of choice; otherwise, we select the largest $L < L_{opt}$ such that $L + C = t_R$. Of course, for non-real-time applications, we always select L_{opt} as the disconnection time period to reduce the reconnection time.

To illustrate how our result presented earlier can be applied to real-time web applications with a deadline, consider that the data displayed in Figure 4.11 are for a real-time web application for which $t_R = 25$ minutes and $\lambda = 0.5\lambda_w$ since two users are updating the same set of web pages simultaneously with virtually the same rate. Then, from the figure we see that the MU should propagate its updates at $L_{opt} = 850$ since $L_{opt} + C_{min} = 850 + 350$ seconds which is less than 25 minutes. However, if on the other hand, the real-time deadline is 15 minutes, then the MU should disconnect itself from the server (while it performs updates) for only approximately 540 seconds because with that disconnection time period, the anticipated update propagation time is 360 seconds as predicted from the diagram, i.e., the point has the coordinate (540,360) such that $360 + 540 = 900$ seconds $= 15$ minutes.

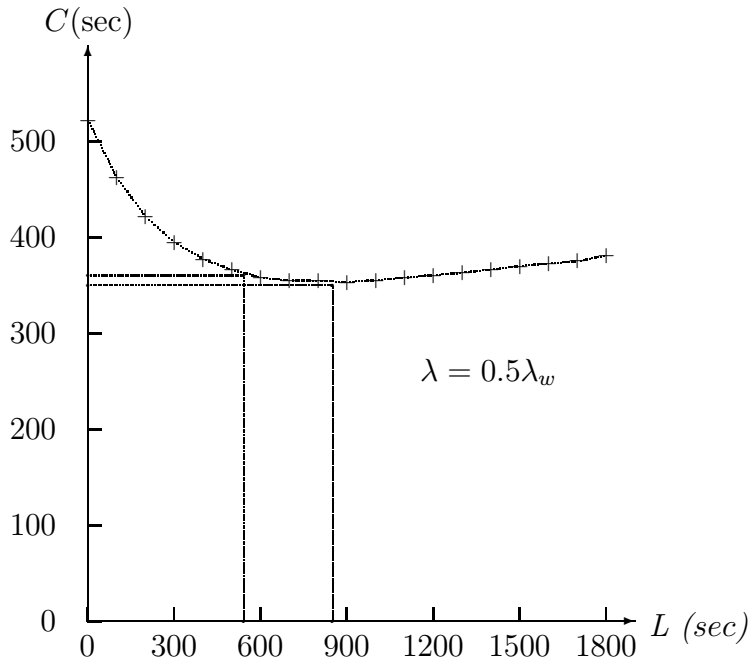

Fig. 4.11. Applying to Real-Time Web Applications with Deadline Requirements.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

In this thesis, we have discussed several algorithms for propagating updates made by the MU and analyzed their effect on performance in terms of the communication time needed to propagate updates, including the time needed to detect and resolve conflict if it happens. We have developed simple analytical arguments to relate the disconnection period of the MU with the associated communication time required based on the choice of disconnection period, taking into account some system conditions such as the update rates of the MU and the world, the time to communicate between the MU and server based on differencing/locking, and the time to resolve web page conflicts, etc. The end result is a simple $L - C$ diagram which may be applied at the run-time by the MU to determine the longest time to stay disconnected (L) so as to minimize the communication time (C) needed to propagate updates.

Our analysis shows that there exists an optimal L_{opt} value under which C is minimized. Moreover, this optimal disconnection period increases as the mobile user update rate increases relative to the world update rate. At one extreme where only the mobile user performs updates, the communication cost is minimum at time

equals ∞. At the other extreme where the world's update rate is much higher than that of the MU, the communication cost is minimum at time equals 0 since the world is going to update the web pages very often and the benefit of disconnecting from the web server is lost since the cost penalty for detecting and resolving conflicts dominates. In this case, it is better that the MU stays connected to the server, locks the web pages, and propagates updates to the server while it is connected.

For those cases where the MU's update rate is a fraction of that of the world, we observed there exists a finite value of L_{opt} under which the communication cost for update propagation is minimized. We also observed similar curves with other set of parameter values as long as the time to execute the merge operation to resolve update conflicts D_m is not too small, i.e., more than 10 seconds, with all other parameter values fixed. When D_m is small, the optimal disconnection time interval, i.e., L_{opt} approaches ∞ because the penalty of rejection is virtually zero. In practice, D_m is large relative to other parameter values since a manual inspection may be required even with the help of merging tools when the MU resolves update conflicts.

One application of the analysis result as we have demonstrated in this thesis is that for applications with a real-time deadline. We can use the result presented in the $L-C$ diagram to determine the time at which the MU should reconnect to the server so that the total time, including the disconnection time and the algorithm execution time for update propagation, is less than the deadline. The needed $L-C$ diagrams can be generated at the static time to cover a possible range of parameter values. Such results can be stored in a table in the MU which can then perform a table lookup at the run-time to adapt to environment changes at the run-time to reflect network conditions such as channel bandwidth. The (L, C) diagram generated from our

analysis also provides information regarding "anticipated" arrivals of disconnected mobile users. This information can be very helpful to wireless mobile networks as the system allocates wireless channels to users.

There are some possible future research areas which can be extended from this thesis work. We can investigate how a channel allocation manager in the mobile network could use the (L, C) diagram information to decide when a mobile user should be reconnected to the network so that the mobile user will stay connected with the least amount of time to propagate updates. We can further develop channel scheduling algorithms for optimal allocation of wireless channels taking into account both random and anticipated arrivals of mobile users. We can apply the result obtained from the thesis to building real-time web applications used in wireless mobile environments so that the MU can always select the longest disconnection time possible to reduce the communication cost without violating the real-time requirement and also to save it's valuable battery power.

Also, web applications which are not real-time do not require mobile users to stay connected for propagating updates. In this case, the mobile user may just send the request to the server and then disconnect immediately. Later it can reconnect to the server to check if the server has accepted/rejected the request. In the future, we can modify our analysis model to account for this possibility. Finally, as most web applications are read-only applications, the read/write probability ratio may affect the performance of the overall system because as web pages are locked for update propagation, read operations cannot be served. Therefore, it is possible to develop algorithms taking into account of the effect of the read/write ratio factor to optimize the overall system performance.

BIBLIOGRAPHY

[1] Palm Computing `http://palmpilot.3com/products/palmvii/index.html`

[2] E. Pitoura and G. Samaras, *Data Management for Mobile Computing,* Kluwer Academic Publishers, 1998.

[3] M. Liljeberg, T. Alanko, M. Kojo, H. Laamanen, K. Raatikainen, "Optimizing World Wide Web for weakly connected mobile workstations: An indirect approach," *Proceeing of the 2nd International Workshop on Services in Distributed and Networked Environments,* June 1995, pp.153-161.

[4] A. Joshi, S. Weerawarana, E. Houstis, "On Disconnected Browsing of Distributed Information," *Proceeding of the 7th IEEE Workshop on Research Issues in Data Engineering RIDE,* 1997, pp. 101-107.

[5] A. Acharya, B. Badrinath, T. Imielinski, J. Nasvas, "A WWW-based Location-Dependent Information Service for Mobile Clients,"

`http://www.cs.rutgers.edu/~navas/dataman/papers/loc_dep_mosaic/Overview.html.`

[6] R. Floyd, R. Housel and C. Tait, "Mobile web access using eNetwork Web Express," *IEEE Personal Communications*, Vol. 5, No. 5, Oct. 1998, pp. 47-52.

[7] M.S. Mazer and C.L. Brooks, "Writing the web while disconnected," *IEEE Personal Communications*, Vol. 5, No. 5, Oct. 1998, pp. 35-41.

[8] J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Transactions on Computer Systems*, Vol. 10, No. 1, Feb. 1992, pp. 3-25.

[9] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, D. Steere, "Coda: A highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, Vol. 39, No. 4, April 1990, pp. 447-451

[10] Y. Goland et al., "Extensions for distributed authoring and versioning on on the WWW,"

`http://www.ics.uci.edu/ejw/authoring/protocol/draft-ietf-webdav-protocol-0.5.html.`

[11] B.C. Housel, G. Samaras and D.B. Lindquist, "WebExpress: a client/intercept based system for optimizing web browsing in a wireless environment," *ACM/Baltzer Mobile Networking and Applications (MONET),* Vol. 3, No. 4, 1998, pp. 419-431.

[12] Z. Jiang and L. Kleinrock, "Web prefetching in a mobile environment," *IEEE Personal Communications*, Vol. 5, No. 5, Oct. 1998, pp. 25-34.

[13] M.F. Kaashoek, T. Pinckney, and J.A. Tauber, "Dynamic documents: mobile wireless access to the WWW," *IEEE Workshop on Mobile Computing Systems and Applications,* Santa Cruz, CA, Dec. 1994., pp. 179-184

[14] H. Bharadvaj, A. Joshi, S. Auephanwiriyakul, "An Active Transcoding Proxy to support Mobile Web Access," *Proceeding of The 17th IEEE Symposium on Reliable Distributed Systems,* 1998, pp. 218-224.

[15] A. Joshi, C. Punyapu, P. Karnam, "Personalization and Asyncronicity to support Mobile web Access,"

`http://www.cecs.missouri.edu/~joshi/dbrowse/mb_ref.html.`

[16] R. Kavasseri, T. Keating, M. Wittman, A. Joshi, S. Weerawarana, "Web Intelligent Query - Disconnected Web Browsing with Collaborative Techniques," *Proceeding of the 1st IFCIS Conference on Cooperative Information Systems,* 1996, pp. 167-174.

[17] H. Chang, C. Tait, N. Cohen, M. Shapiro, S. Mastrianni, R. Floyd, B. Housel, D. Lindquist, "Web Browsing in a Wireless Environment: Disconnected and Asyncronous Operation in ARTour Web Express," *Proceeding of the ACM/IEEE MobiCom'97,* Budapest, Hungary, Sept. 1997, pp. 260-269.

[18] J. Klingemann, T. Tesch, J. Wasch, "Enabling Cooperation Among Disconnected Mobile Users," *Proceeding of the 2nd IFCIS International Conference on Cooperative Information Systems (CoopIS'97),* S. Carolina, June 1997, pp. 141-154.

[19] J. Klingemann, T. Tesch, J. Wasch, "Cooperative Data management and its Application to Mobile Computing," *International Journal of Cooperative Information Systems,* Vol. 6, No. 4, pp. 341-365.

[20] A. Fasbender, et al., "Any network, any terminal, any where," *IEEE Personal Communications*, Vol. 6, No. 2, April 1999, pp.22-30

[21] V. Bellotti, S. Bly, "Walking Away From the Desktop Computer: Distributed Coolaboration and mobility in a Product Design Team" *Proceedings of the ACM 1996 conference on Computer Supported Cooperative Work,* pp. 209-218.

[22] Ora Lassila, "HTTP-based Distributed Content Editing Scenarios,"

`http://www.w3.org/TR/NOTE-http-edit-dist-scenarios.`

[23] Cisco Technical Documents,

`http://www.cisco.com.`

[24] Online Journalism, *The Washington Post,* Sep. 5, 1999.

[25] Web Advisor for Software Engineers,

`http://hokies.freeservers.com/project.htm.`

[26] J. Munson, P. Dewan, "A flexible Object Merging Framework" *Proceedings of ACM Conference on Computer Supported Cooperative Work*, October 1994, pp. 231-242.

[27] Wall Street Update

`http://www.newsweek.com/marketinc.aps.`

[28] Washington Post Top News

`http://www.washingtonpost.com/wp-srv/digest/digest.htm.`

[29] America Online Business Headlines

`http://www.aol.com/mynews/business/home.adp.`

[30] M.H. MacDougall *Simulating Computer System: Techniques and Tools,* The MIT Press, 1987

[31] A.S. Tanenbaum, *Computer Networks*, 3rd Edition, Prentice Hall, 1996.

APPENDIX A

SOURCE CODE FOR COMPUTING THE ANALYTICAL RESULTS

This Appendix contains source code for obtaining the analytical data described in Chapter 4. There are 3 .c files and one .h file. The meanings of these files are briefly explained as follows:

`thesis.h` This include file contains symbolic constants and structures to be use in all C programs

`analytical0.c` This file contains the program to obtain analytical data for the effect of $\lambda : \lambda_w$ ratio under single-page update propagation.

`analytical1.c` This file contains the program to obtain analytical data for the effect of $\lambda : \lambda_w$ ratio under algorithm 1 of multiple-page update propagation.

`analytical2.c` This file contains the program to obtain analytical data for the effect of $\lambda : \lambda_w$ ratio under algorithm 2 of multiple-page update propagation.

To create an executable program, for example, for analytical2.c, we use the following command:

```
cc analytical2.c -o analytical2
```

To execute a program, for example, for analytical2, we use the folowing command:

```
analytical2 ratio [output file]
```

```
/*********************************************************************/
/*                                                                   */
/*                          File "thesis.h"                          */
/*               Define Constants and Structures                     */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#ifndef _THESIS_H
#define _THESIS_H

typedef double real;
#define then

#define MAX_WEB_PAGE_PREFETCHED 5
#define MAX_POINT 19
#define TRUE 1
#define FALSE 0

struct WEB_PAGE_CLIENT_STATUS
{
    double lambda;
    real T_c;
    double p;
};

struct WEB_PAGE_SERVER_STATUS
```

```
{
    double lambda_w;
    real T_s;
    double r;
};

extern real Lq(), U(), B(), time();
extern char *fname(), *mname();
extern FILE *sendto();

extern real ranf(), uniform(), expntl(), erlang(), hyperx(), normal();
extern long seed();

extern init_bm(), civals();
extern double exprand();
extern int obs();

#endif
/************************************************************************/
/*                                                                      */
/*                          File "analytical0.c"                        */
/*    analytical analysis for single-page update propagation            */
/*    algorithm to study the effect of lambda:lambda_w ratio.           */
/*                                                                      */
/************************************************************************/

#include "thesis.h"

void main (int argc, char*argv[])
{
```

```
double T, B, pm;

double lambda_w, lambda, p, r;

int s0, Dm, L[MAX_POINT];

int ratio; //ratio between lambda and lambda_w

FILE  *out = NULL;

int j;

double totalC[MAX_POINT];


for(j=0; j<MAX_POINT; j++)

{

    L[j] = j*200;

}


//check argc and argv

if ( !(2 == argc || 3 == argc) )

{

    printf("Usage: thesis ratio [output_file]\n");

    return;

}


ratio = atoi(argv[1]);


//initial conditions

B = 9.6; //kbps

pm = 0.1;//fraction of a web page modified by the MU

s0 = 50; //KB

Dm = 100;//sec

lambda_w = 1.0/360;

lambda = lambda_w*ratio/100;
```

```
//average round-trip communication time
//T = (pm*s0*8*1024)/(B*1000);
T = 4.3;

for(j=0; j<MAX_POINT; j++)
{

        //probabilities

        p = (1-exp(-1.0*L[j]*lambda));
        r = (1-exp(-1.0*L[j]*lambda_w));

        //compute the total propagation time
        totalC[j] = p*r*(3*T+Dm) + T*(1-r)*p +
               (1-p)*r*(Dm+2*T) + (1-r)*(1-p)*(Dm+T);

 printf("The average communication cost is %f when L = %d, ratio = %d\n",
            totalC[j], L[j], ratio);

 }


//write to output file
if (3 == argc)
{
    out = fopen(argv[2], "w");
    if (NULL == out)
    {
        printf("The output file %s was not opened.  Program exits.\n", argv[2]);
        return;
```

```
        }
            for(j=0; j<MAX_POINT; j++)
            {
                fprintf(out, "\\plotpoint(%d,%f)\n", j+1,totalC[j]/10);
            }
        fclose(out);

    }
}


/*********************************************************************/
/*                                                                   */
/*                          File "analytical1.c"                     */
/*    analytical analysis for case 1 of multiple-page update         */
/*    propagation to study the effect of lambda:lambda_w ratio       */
/*                                                                   */
/*********************************************************************/

#include "thesis.h"

//array(1) -- record status of web pages at the MU side
struct WEB_PAGE_CLIENT_STATUS _web_page_client[MAX_WEB_PAGE_PREFETCHED];
//array(2) -- record status of web pages at the server side
struct WEB_PAGE_SERVER_STATUS _web_page_server[MAX_WEB_PAGE_PREFETCHED];

void main (int argc, char*argv[])
{
    double T, B, pm;
    double lambda_w1, lambda_w2, lambda_w3, lambda_w4, lambda_w5;
    double lambda1, lambda2, lambda3, lambda4, lambda5;
```

```
int s0, Dm, L[MAX_POINT];

int ratio; //ratio between lambda and lambda_w

FILE  *out = NULL;

double p[MAX_POINT], r[MAX_POINT];

int i, j;

int cont;

double totalC[MAX_POINT], C[MAX_WEB_PAGE_PREFETCHED];


for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)

{

    C[i] = 0.0;

}


for(j=0; j<MAX_POINT; j++)

{

    L[j] = j*200;

}


//check argc and argv

if ( !(2 == argc || 3 == argc) )

{

    printf("Usage: thesis ratio [output_file]\n");

    return;

}


ratio = atoi(argv[1]);


//initial conditions

B = 9.6; //kbps

pm = 0.1;//fraction of a web page modified by the MU
```

```
ratio = 80; //KB

Dm = 100;//sec

lambda_w1 = 1.0/360;

lambda_w2 = 1.0/10800;

lambda_w3 = 1.0/600;

lambda_w4 = 1.0/3600;

lambda_w5 = 1.0/240;

lambda1 = lambda_w1*ratio/100;

lambda2 = lambda_w2*ratio/100;

lambda3 = lambda_w3*ratio/100;

lambda4 = lambda_w4*ratio/100;

lambda5 = lambda_w5*ratio/100;


//average round-trip communication time

T = (pm*s0*8*1024)/(B*1000);


 // structures' initialization

_web_page_client[1].lambda = lambda1;

_web_page_client[2].lambda = lambda2;

_web_page_client[3].lambda = lambda3;

_web_page_client[4].lambda = lambda4;

_web_page_client[5].lambda = lambda5;

_web_page_server[1].lambda_w = lambda_w1-lambda1;

_web_page_server[2].lambda_w = lambda_w2-lambda2;

_web_page_server[3].lambda_w = lambda_w3-lambda3;

_web_page_server[4].lambda_w = lambda_w4-lambda4;

_web_page_server[5].lambda_w = lambda_w5-lambda5;


for(j=0; j<MAX_POINT; j++)

{
```

```
        //probabilities
        for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)
        {
            _web_page_client[i].p = (1-exp(-1.0*L[j]*_web_page_client[i].lambda));
            _web_page_server[i].r = (1-exp(-1.0*L[j]*_web_page_server[i].lambda_w));

        }

        //compute the total propagation time
        totalC[j] = 0.0;
        for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)
        {
         C[i] = _web_page_client[i].p*_web_page_server[i].r*(3*T+Dm) +
                T*(1-_web_page_server[i].r)*_web_page_client[i].p +
                (1-_web_page_client[i].p)*_web_page_server[i].r*(Dm+2*T) +
                (1-_web_page_server[i].r)*(1-_web_page_client[i].p)*(Dm+T);
         totalC[j] += C[i];
        }

        printf("The average communication cost is %f when L = %d, ratio = %d\n",
            totalC[j], L[j], ratio);

}

//write to output file
if (3 == argc)
{
    out = fopen(argv[2], "w");
    if (NULL == out)
    {
```

```
            printf("The output file %s was not opened.  Program exits.\n", argv[2]);
            return;
        }
            for(j=0; j<MAX_POINT; j++)
            {
                fprintf(out, "\\plotpoint(%d,%f)\n", j+1,totalC[j]/100);
            }
        fclose(out);

    }
}


/*********************************************************************/
/*                                                                   */
/*                          File "analytical2.c"                     */
/*    analytical analysis for case 2 of multiple-page update         */
/*    propagation to study the effect of lambda:lambda_w ratio.      */
/*                                                                   */
/*********************************************************************/

#include "thesis.h"

//array(1) -- record status of web pages at the MU side
struct WEB_PAGE_CLIENT_STATUS _web_page_client[MAX_WEB_PAGE_PREFETCHED];
//array(2) -- record status of web pages at the server side
struct WEB_PAGE_SERVER_STATUS _web_page_server[MAX_WEB_PAGE_PREFETCHED];

void main (int argc, char*argv[])
{
```

```
double T, B, pm;
double lambda_w1, lambda_w2, lambda_w3, lambda_w4, lambda_w5;
double lambda1, lambda2, lambda3, lambda4, lambda5;
int s0, Dm, L[MAX_POINT];
int ratio; //ratio between lambda and lambda_w
FILE  *out = NULL;
double p[MAX_POINT], r[MAX_POINT];
int i, j;
int cont;
double totalC[MAX_POINT], C[MAX_WEB_PAGE_PREFETCHED];

for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)
{
    C[i] = 0.0;
}

for(j=0; j<MAX_POINT; j++)
{
    L[j] = j*200;
}

//check argc and argv
if ( !(2 == argc || 3 == argc) )
{
    printf("Usage: thesis ratio [output_file]\n");
    return;
}

ratio = atoi(argv[1]);
```

```
//initial conditions

B = 9.6; //kbps

pm = 0.1;//fraction of a web page modified by the MU

s0 = 50; //KB

Dm = 100;//sec

lambda_w1 = 1.0/360;

lambda_w2 = 1.0/10800;

lambda_w3 = 1.0/600;

lambda_w4 = 1.0/3600;

lambda_w5 = 1.0/240;

lambda1 = lambda_w1*ratio/100;

lambda2 = lambda_w2*ratio/100;

lambda3 = lambda_w3*ratio/100;

lambda4 = lambda_w4*ratio/100;

lambda5 = lambda_w5*ratio/100;


//average round-trip communication time

T = (pm*s0*8*1024)/(B*1000);


 // structures' initialization

_web_page_client[1].lambda = lambda1;

_web_page_client[2].lambda = lambda2;

_web_page_client[3].lambda = lambda3;

_web_page_client[4].lambda = lambda4;

_web_page_client[5].lambda = lambda5;

_web_page_server[1].lambda_w = lambda_w1-lambda1;

_web_page_server[2].lambda_w = lambda_w2-lambda2;

_web_page_server[3].lambda_w = lambda_w3-lambda3;

_web_page_server[4].lambda_w = lambda_w4-lambda4;

_web_page_server[5].lambda_w = lambda_w5-lambda5;
```

```
for(j=0; j<MAX_POINT; j++)
{
        //probabilities
        for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)
        {
            _web_page_client[i].p = (1-exp(-1.0*L[j]*_web_page_client[i].lambda));
            _web_page_server[i].r = (1-exp(-1.0*L[j]*_web_page_server[i].lambda_w));

        }

        //compute the total propagation time
        totalC[j] = 0.0;
        for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)
        {
         C[i] = _web_page_client[i].p*_web_page_server[i].r*(2*T+Dm) +
                T*(1-_web_page_server[i].r)*_web_page_client[i].p +
                (1-_web_page_client[i].p)*_web_page_server[i].r*(2*T+Dm) +
                (1-_web_page_server[i].r)*(1-_web_page_client[i].p)*(Dm+T);
         totalC[j] += C[i];
        }

        printf("The average communication cost is %f when L = %d, ratio = %d\n",
            totalC[j], L[j], ratio);

}

//write to output file
if (3 == argc)
{
```

```
        out = fopen(argv[2], "w");
        if (NULL == out)
        {
            printf("The output file %s was not opened.  Program exits.\n", argv[2]);
            return;
        }
            for(j=0; j<MAX_POINT; j++)
            {
                fprintf(out, "\\plotpoint(%d,%f)\n", j+1,totalC[j]/100);
            }
        fclose(out);

    }
}
```

APPENDIX B

SOURCE CODE FOR COMPUTING SIMULATION RESULTS

This Appendix contains source code for obtaining the simulation data described in Chapter 4. There are 3 .c files. The meanings of these files are briefly explained as follows:

`simulation0.c` This file contains the program to obtain simulation data for the effect of $\lambda : \lambda_w$ ratio under single-page update propagation.

`simulation1.c` This file contains the program to obtain simulation data for the effect of $\lambda : \lambda_w$ ratio under algorithm 1 of multiple-page update propagation.

`simulation2.c` This file contains the program to obtain simulation data for the effect of $\lambda : \lambda_w$ ratio under algorithm 2 of multiple-page update propagation.

These source-code files are to be linked with three files called `bmeans.c`, `stat.c`, `random.c` which come with the SMPL simulation package provided in [30].

To create an executable program, for example, for simulation2.c, we use the following command:

```
cc simulation2.c bmeans.c random.c -lm -o simulation2
```

To execute a program, for example, for simulation2, we use the following command:

```
simulation2 ratio [output file]
```

```
/*********************************************************************/
/*                                                                   */
/*                  File "simulation0.c"                             */
/*    A simulation for single-page update propagation algorithm      */
/*    to study the effect of lambda:lambda_w ratio.                  */
/*    The average communication cost is measured to 5% accuracy      */
/*    and 95% confidence levels.                                     */
/*                                                                   */
/*********************************************************************/

#include "thesis.h"

void main (int argc, char*argv[])
{
    double T, B, pm;
    double lambda_w, lambda;
    real T_c, T_s;
    int s0, Dm, L[MAX_POINT];
    int ratio; //ratio between lambda and lambda_w
    FILE  *out = NULL;
    real mean[MAX_POINT], hw[MAX_POINT];
    int i, j, nb[MAX_POINT];
    int cont;
    double totalC;

    for(j=0; j<MAX_POINT; j++)
    {
        L[j] = j*200;
    }
```

```
//check argc and argv
if ( !(2 == argc || 3 == argc) )
{
    printf("Usage: thesis ratio [output_file]\n");
    return;
}
ratio = atoi(argv[1]);

//initial conditions
B = 9.6; //kbps
pm = 0.1;//fraction of a web page modified by the MU
s0 = 50; //KB
Dm = 100;//sec
lambda_w = 1.0/360;
lambda = lambda_w*ratio/100;

//average round-trip communication time
T = (pm*s0*8*1024)/(B*1000);

for(j=0; j<MAX_POINT; j++)
{
    cont = TRUE;
    init_bm(200,2000); // let m0 be 200 and mb be 2000 observations
    while (cont)
    {
        //randomly generate an average time interval that an update will occure at ..,
        //...the client and the server

            T_c = expntl(1.0/lambda);
            T_s = expntl(1.0/lambda_w);
```

```
    //compute the total propagation time
    totalC = 0.0;
    //(1) the MU has updated the page and the server has not updated the page
    if ((T_c < L[j]) && (T_s > L[j]))
    {
        totalC = T;
    }
    //(2) the MU has updated the page and the server has updated the page
    if ((T_c < L[j]) && (T_s < L[j]))
    {
        totalC = (3*T + Dm);
    }
    //(3) the MU has not update the page and the server has updated the page
    if ((T_c > L[j]) &&
        (T_s < L[j]))
    {
        totalC = (2*T + Dm);
    }
    //(4) the MU has not updated the page and the server has not updated the page
    if ((T_c > L[j]) &&
        (T_s > L[j]))
    {
        totalC = T+Dm;
    }

if (obs(totalC) == 1)
{
    cont = FALSE;
}
```

```
        }


        //get mean value
        civals(&mean[j], &hw[j], &nb[j]);
        printf("Mean is %f and half width is %f after %d batches\n", mean[j], hw[j], nb[j]);
        printf("The average communication cost is %f when L = %d, ratio = %d\n", mean[j], L[j], ratio);
    }


    //write to output file
    if (3 == argc)
    {
        out = fopen(argv[2], "w");
        if (NULL == out)
        {
            printf("The output file %s was not opened.  Program exits.\n", argv[2]);
            return;
        }
            for(j=0; j<MAX_POINT; j++)
            {
                fprintf(out, "\\plotpoint(%d,%f)\n", j+1,mean[j]/10);
            }
        fclose(out);
    }
}



/*********************************************************************/
/*                                                                   */
/*                          File "simulation2.c"                     */
/*    A simulation for algorithm 2 of multiple-page update           */
/*    propagation to study the effect of lambda:lambda_w ratio.      */
```

```
/*    The average communication cost is measured to 5% accuracy      */
/*    and 95% confidence levels                                       */
/*                                                                    */
/********************************************************************/

#include "thesis.h"

//array(1) -- record status of web pages at the MU side
struct WEB_PAGE_CLIENT_STATUS _web_page_client[MAX_WEB_PAGE_PREFETCHED];
//array(2) -- record status of web pages at the server side
struct WEB_PAGE_SERVER_STATUS _web_page_server[MAX_WEB_PAGE_PREFETCHED];

void main (int argc, char*argv[])
{
    double T, B, pm;
    double lambda_w1, lambda_w2, lambda_w3, lambda_w4, lambda_w5;
    double lambda1, lambda2, lambda3, lambda4, lambda5;
    int s0, Dm, L[MAX_POINT];
    int ratio; //ratio between lambda and lambda_w
    FILE  *out = NULL;
    real mean[MAX_POINT], hw[MAX_POINT];
    int i, j, nb[MAX_POINT];
    int cont;
    double totalC, C[MAX_WEB_PAGE_PREFETCHED];

    for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)
    {
        C[i] = 0.0;
    }
```

```
for(j=0; j<MAX_POINT; j++)
{
    L[j] = j*200;
}

//check argc and argv
if ( !(2 == argc || 3 == argc) )
{
    printf("Usage: thesis ratio [output_file]\n");
    return;
}

ratio = atoi(argv[1]);

//initial conditions
B = 9.6; //kbps
pm = 0.1;//fraction of a web page modified by the MU
s0 = 50; //KB
Dm = 100;//sec
lambda_w1 = 1.0/360;
lambda_w2 = 1.0/10800;
lambda_w3 = 1.0/600;
lambda_w4 = 1.0/3600;
lambda_w5 = 1.0/240;
lambda1 = lambda_w1*ratio/100;
lambda2 = lambda_w2*ratio/100;
lambda3 = lambda_w3*ratio/100;
lambda4 = lambda_w4*ratio/100;
lambda5 = lambda_w5*ratio/100;
```

```
//average round-trip communication time

T = (pm*s0*8*1024)/(B*1000);


 // structures' initialization

_web_page_client[1].lambda = lambda1;

_web_page_client[2].lambda = lambda2;

_web_page_client[3].lambda = lambda3;

_web_page_client[4].lambda = lambda4;

_web_page_client[5].lambda = lambda5;

_web_page_server[1].lambda_w = lambda_w1-lambda1;

_web_page_server[2].lambda_w = lambda_w2-lambda2;

_web_page_server[3].lambda_w = lambda_w3-lambda3;

_web_page_server[4].lambda_w = lambda_w4-lambda4;

_web_page_server[5].lambda_w = lambda_w5-lambda5;


for(j=0; j<MAX_POINT; j++)

{

    cont = TRUE;

    init_bm(200,2000); // let m0 be 200 and mb be 2000 observations

    while (cont)

    {

        //randomly generate an average time interval that an update will occure at ..,

        //...the client and the server


        for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)

        {

            _web_page_client[i].T_c = expntl(1.0/_web_page_client[i].lambda);

            _web_page_server[i].T_s = expntl(1.0/_web_page_server[i].lambda_w);

        }
```

```
//compute the total propagation time
totalC = 0.0;
for(i=1; i<=MAX_WEB_PAGE_PREFETCHED; i++)
{
    //(1) the MU has updated the page and the server has not updated the page
    if ((_web_page_client[i].T_c < L[j]) && (_web_page_server[i].T_s > L[j]))
    {
        C[i] = T;
    }
    //(2) the MU has updated the page and the server has updated the page
    if ((_web_page_client[i].T_c < L[j]) && (_web_page_server[i].T_s < L[j]))
    {
        C[i] = (2*T + Dm);
    }
    //(3) the MU has not update the page and the server has updated the page
    if ((_web_page_client[i].T_c > L[j]) &&
        (_web_page_server[i].T_s < L[j]))
    {
        C[i] = (Dm + 2*T);
    }
    //(4) the MU has not updated the page and the server has not updated the page
    if ((_web_page_client[i].T_c > L[j]) &&
        (_web_page_server[i].T_s > L[j]))
    {
        C[i] = T+Dm;
    }
    totalC += C[i];
    }
```

```
        if (obs(totalC) == 1)
        {
            cont = FALSE;
        }
    }

    //get mean value
    civals(&mean[j], &hw[j], &nb[j]);
    printf("Mean is %f and half width is %f after %d batches\n", mean[j], hw[j], nb[j]);

    printf("The average communication cost is %f when L = %d, ratio = %d\n",
            mean[j], L[j], ratio);
}

//write to output file
if (3 == argc)
{
    out = fopen(argv[2], "w");
    if (NULL == out)
    {
        printf("The output file %s was not opened.  Program exits.\n", argv[2]);
        return;
    }
        for(j=0; j<MAX_POINT; j++)
        {
            fprintf(out, "\\plotpoint(%d,%f)\n",j+1,mean[j]/100);
        }
    fclose(out);

}
```

```
}

/*********************************************************************/
/*                                                                   */
/*                          File "simulation3.c"                     */
/*    A simulation for single-page update propagation                */
/*    to study the effect of the time to perform the merge operation. */
/*    The average communication cost is measured to                  */
/*    5% of accuracy and 95% of confidence levels                    */
/*                                                                   */
/*********************************************************************/

#include "thesis.h"

void main (int argc, char*argv[])
{
    double T, B, pm;
    double lambda_w, lambda, T_c, T_s;
    int s0, ratio, L[SIMULATION_POINT];
    int Dm; //time to perform merge operation
    FILE  *out = NULL;
    real mean[SIMULATION_POINT], hw[SIMULATION_POINT];
    int j, nb[SIMULATION_POINT];
    int cont;
    double totalC;

    for(j=0; j<SIMULATION_POINT; j++)
    {
        L[j] = j*200;
    }
```

```
//check argc and argv
if ( !(2 == argc || 3 == argc) )
{
    printf("Usage: thesis Dm [output_file]\n");
    return;
}

Dm = atoi(argv[1]);

//initial conditions
B = 9.6; //kbps
pm = 0.1;//fraction of a web page modified by the MU
s0 = 50; //KB
ratio = 80; //ratio is fixed to 80
lambda_w = 1.0/360;// 10 updates/hr
lambda = lambda_w*ratio/100;

//average round-trip communication time
T = (pm*s0*8*1024)/(B*1000);

for(j=0; j<SIMULATION_POINT; j++)
{
    cont = TRUE;
    init_bm(200,2000); // let m0 be 200 and mb be 2000 observations
    while (cont)
    {
        //randomly generate an average time interval that an update will occure at ..,
        //...the client and the server
```

```
    T_c = expntl(1.0/lambda);
    T_s = expntl(1.0/lambda_w);

//compute the total propagation time
    totalC = 0.0;
    //(1) the MU has updated the page and the server has not updated the page
    if ((T_c < L[j]) && (T_s > L[j]))
    {
        totalC = T;
    }
    //(2) the MU has updated the page and the server has updated the page
    if ((T_c < L[j]) && (T_s < L[j]))
    {
        totalC = (3*T + Dm);
    }
    //(3) the MU has not update the page and the server has updated the page
    if ((T_c > L[j]) &&
        (T_s < L[j]))
    {
        totalC = (2*T + Dm);
    }
    //(4) the MU has not updated the page and the server has not updated the page
    if ((T_c > L[j]) &&
        (T_s > L[j]))
    {
        totalC = T+Dm;
    }

if (obs(totalC) == 1)
{
```

```
                cont = FALSE;
            }
        }

        //get mean value
        civals(&mean[j], &hw[j], &nb[j]);
        printf("Mean is %f and half width is %f after %d batches\n", mean[j], hw[j], nb[j]);

        printf("The average communication cost is %f when L = %d, Dm = %d\n",
                mean[j], L[j], Dm);
    }

    //write to output file
    if (3 == argc)
    {
        out = fopen(argv[2], "w");
        if (NULL == out)
        {
            printf("The output file %s was not opened.  Program exits.\n", argv[2]);
            return;
        }
            for(j=0; j<SIMULATION_POINT; j++)
            {
                fprintf(out, "\\plotpoint(%d,%f)\n", j+1,mean[j]/10);
            }
        fclose(out);

    }
}
```

BIOGRAPHICAL SKETCH OF THE AUTHOR

Ngoc Anh Phan was born in Vietnam on April 17, 1973 and lived there until 1992. She graduated from Hanoi University of Technologies in 1992. She then earned a B.S. degree in Telecommunication Management from Moscow Technical University of Communications and Computer Science in 1997. In 1997 she moved to the United States to pursue an M.S. degree in Computer Science at Virginia Polytechnic Institute and State University, where she held a research and teaching assistant position in the Department of Computer Science. She graduated in December 1999.

Her career interests are in software development and network design. Her research interests include wireless communications and mobile computing.

She is fluent in English, Russian and Vietnamese. Her hobbies include landscape and abstract painting, biking, hiking and sailing.