Resource Management for Delivery of Dynamic Information

by

David Evans

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Computer Science

Waterloo, Ontario, Canada, 2005

Author's Declaration for Electronic Submission of a Thesis

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Information delivery via the web has become very popular. Along with a growing user population, systems increasingly are supporting content that changes frequently, personalised information, and differentiation and choice. This thesis is concerned with the design and evaluation of resource management strategies for such systems. An architecture that provides scalability through caching proxies is considered. When a cached page is updated at the server, the cached copy may become stale if the server is not able to transmit the update to the proxies immediately. From the perspective of the server, resources are required to transmit updates for cached pages and to process requests for pages that are not cached. Analytic results on how the available resources should be managed in order to minimise staleness-related cost are presented. An efficient algorithm that the server can use to determine the set of pages that should be cached and a policy for transmitting updates for these pages are also presented. We then apply these results to page fragments, a technique that can provide increased efficiency for delivery of personalised pages.

Acknowledgements

Completing a Ph.D. is a bit like being a teenager. You feel utterly alone, are plagued with doubts, and think that you are the only person suffering these problems. That's not typically true, of course. All graduate students—well, the honest ones—experience the same feelings of loneliness, inadequacy, and anxiety. It would have been impossible for me to survive this on my own and so I have many people to thank for as many different reasons.

My supervisor, Professor Johnny Wong, has provided invaluable advice, knowledge, and experience. This thesis is the better for his insightful comments and criticisms. I also wish to thank the members of my examining committee, Professors Jay Black, Ken Salem, Sherman Shen, and Michael Bauer. Lindsay Chen implemented the simulator used in Chapter 6. Terry Lau, Don Bourne, Darl Crick, Weidong Kou, and everyone else I worked with at the IBM Toronto Lab have given me insight into the sorts of scalability problems faced in industry and the approaches taken to solving them. My research was financially supported in part by the Canadian Institute for Telecommunications Research under the Networks of Centres of Excellence programme of the Government of Canada, the IBM Centre for Advanced Studies at the IBM Toronto Lab, and the IBM Ph.D. Fellowship Programme.

I thank my parents for, long ago, helping me choose the path that has lead me to where I am today.

And, finally, I thank my friends. Without their cleverness, companionship, and love I would have been lost long ago. Such success as I have I owe to you: Ambles, Andrew, Billy, Carlos, Chris, Claus, Dana, Doug, Ellen, Fei, Gabe, Hugh, Jacob, Joanna, Jerry, Keith, Kelly, Lisa, Michael, Nicky, Niel, Olga, Paul, Rick, Rob, Ruth, Scott, Seb, Stefanus, Steve, Tim, Wendy, Will.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Access to information via the world wide web is extremely popular, spanning a wide variety of application areas including electronic commerce, news, and delivery of financial and weather information. In these applications, users submit requests and the system responds with the requested pages. From the user's perspective, an important requirement is good response time.

The system's response time has two main components: delay at the transport network (the Internet, in the case of the web) and delay at the web servers. The Internet today provides best-effort service; its delay performance is good when the traffic is light. However, its delay characteristics are poor when the level of traffic

increases or when the number of users is large. It is possible that future routers will support service guarantees so that the transport network component of the overall response time is more predictable.

This thesis will focus on the server delay. As the number of users accessing the pages stored at a server increases, the aggregate rate of requests directed to that server will increase. For a server with a given capacity, an elementary queueing analysis tells us that an increase in request arrival rate will lead to an increase in response time. The server will not be able to scale to large user populations while maintaining acceptable response time performance.

Scalability can sometimes be achieved by increasing the rate at which the server can process requests. This approach has been used from the early stages of web development. Techniques that have been employed include the use of faster machines, multi-processors, and improvements to the HTTP protocol (such as fetching multiple items using the same TCP connection). However, the first two techniques are constrained by hardware availability and all three do little to alleviate the increased traffic that must be carried by the network.

An alternate approach to reducing response time is to use proxy servers and distribute the user requests among them such that each proxy is able to process its share of requests while delivering satisfactory response time. This reduces the load on the server because fewer requests are received for processing. If the proxies

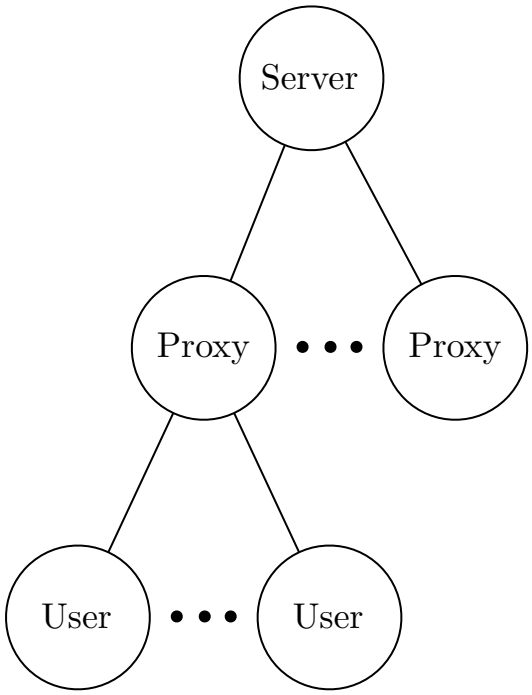

Figure 1.1: A schematic view of web caching

are placed at judicious points in the network, a reduction in traffic can also be realised. By directing a user's requests to a proxy that is close in terms of the network topology, these requests and the information sent in reply will consume fewer network resources than if the requests were sent directly to the server.

In the context of the web, proxies are commonly used to implement caching [1–4]. This is illustrated in Figure 1.1. All pages are stored at the web server. Proxy servers are placed in the network and a subset of the web pages is cached at each proxy. User requests are directed to these proxies rather than to the web server.

Each proxy can be viewed as a site-wide cache for a community of users. These communities may be organised according to geographic location, position within the network topology, or commonality of desired content. As the number of users in a community increases, its proxy might experience heavy load and become a bottleneck. Typically, this problem is resolved by splitting the community and distributing the load to more proxies. One can further organise proxies in a hierarchical manner, where proxies share cached information with those that are nearby.

Caching at proxy servers brings with it the problem of update consistency. Suppose that all page updates are made at the web server. When a page is updated, cached copies at the proxies become stale or out-dated. Strong consistency—that is, an assurance that users are always provided with the most up-to-date version of a page—can be provided if the proxies verify the validity of a cached page for every access. Such an approach requires server resources for the processing of every request and is not a scalable solution. Scalability can be improved if the requirement of strong consistency is relaxed. This is done by techniques such as time-to-live, polling, and invalidation [1, 5, 6]. For these techniques, there is a tradeoff between resource consumption and the staleness of the information delivered to the users. For example, the time-to-live parameter should be kept small if one wishes to reduce the frequency of delivering outdated information. However, a small time-to-live also means that more requests will be forwarded to the server. Similarly, in polling, if

the poll is performed more frequently, it is more likely that the information in the cache will be up-to-date.

With web caching, pages can be classified according to whether a page is frequently requested or not and whether a page is frequently updated or not. Each class might favour a different strategy for delivery to the users. Our classification is described below, together with delivery strategies appropriate to the classes.

Class 1: Page is infrequently requested If pages are infrequently requested, they may be served without consuming significant resources. This suggests that it is not necessary to cache pages in this class.

Class 2: Page is frequently requested but changes rarely For this type of page, cache refresh ceases to be an issue. We may cache all such pages and select a time to live such that the impact on the web server is not significant.

Class 3: Page is frequently requested and changes frequently Popularity makes caching appealing but frequent page updates imply the need for frequent cache refreshes if the cached page copies are to be kept up to date.

In this thesis, we focus on pages in Class 3. Our emphasis is on the questions of which pages should be cached and how page updates should be transmitted in order to achieve the best page timeliness for a given resource availability.

In our analysis, we consider an approach where updates to cached pages are transmitted (or "pushed") to the proxies by the server. This approach allows the server to maintain a high degree of consistency if resources are sufficient to transmit each update without delay. For pages in Class 3, this will be resource intensive as these pages are updated frequently and it may not even be possible if the updates occur at a rate faster than the server can transmit them. In case the server does not have the resources (or does not wish to expend the resources required) to transmit all updates immediately, the cached pages may become stale. Our study focuses on this scenario. Using push for web page delivery is not a new idea [7–10]. What is novel is our mechanism for managing the resources available for delivery such that the pages delivered to the users are as timely as possible.

It has been suggested that another way to enhance the scalability of web caching is by organising pages as a number of fragments [11–15]. The same fragment may be used in a number of pages. Scalability is improved because, when a fragment common to many pages changes, only the fragment in question is affected. Consequently, if the fragment is cached, the server only needs to transmit the update of this fragment. On the other hand, if fragments are not used, each page containing the updated information will become out of date. If these pages are cached, considerable resources are required to transmit updates for each page when compared to those required to transmit the one fragment update. We will investigate the

conditions under which the use of fragments will lead to improved performance.

1.2 Contributions

This thesis represents a significant step in the understanding of the resource management issues inherent in designing information delivery systems for pages that are popular and that change frequently. Specifically, the main contributions of this thesis are as follows:

Characterisation of staleness and resource consumption

We present a new measure of staleness, taking the position that the server is responsible for keeping the pages up to date. Any staleness is caused by the server not being able to transmit updates promptly. This approach is different from those provided by Cho and Garcia-Molina [16, 17], Dingle and Partl [18], Wolf *et al.* [19], Coffman *et al.* [20], and Edwards *et al.* [21].

Optimal strategy for transmission of updates

We consider an architecture where pages are cached at proxy servers and page updates are pushed to these proxies using a mechanism called "transmission attempts". A staleness-related cost is defined, based on our measure of staleness. We develop analytic results that describe how the available resources can best be used to transmit page updates. Specifically, we show that the

staleness-related cost is minimised when transmission attempts are made at regular intervals, regardless of the values of the page request rate and page update rate.

Optimal strategy for caching pages

We further show that for a given page, the staleness-related cost is minimised if the page is either cached at all of the proxies or at none of the proxies, depending on the amount of resources available.

Heuristic algorithms for page delivery

We focus on the case where the page update rates are at least two times faster than the transmission attempt rate. Two heuristic algorithms are developed that determine, for a given resource availability, which pages should be cached (at all the proxies) and which pages are to be retrieved from the server, along with the transmission attempt rate for each cached page, such that the resulting staleness-related cost is close to optimal. We are not aware of any prior algorithms that accomplish this task in the context of web caching.

Page delivery using fragments

Fragments have been suggested as a means for improving system scalability. We extend our system architecture to include fragments and obtain simulation results that characterise the conditions under which the use of fragments

is beneficial in terms of reducing the staleness-related cost. These results significantly enhance understanding of the benefits from using fragments.

1.3 Thesis Organisation

The remainder of this thesis is organised as follows.

Chapter 2 examines relevant work that has been done in information delivery system design, including analysis of broadcast-only systems, push for the world wide web, and web caching. This provides a context for the contributions described in the following chapters.

Chapter 3 presents our proposed architecture, including the details of server–proxy interaction. It also introduces our performance model and defines what we mean by staleness. This definition is different from those of others and, we feel, more appropriate when the system attempts to maintain consistency by pushing page updates to proxies where copies of these pages are cached.

In Chapter 4 we begin our investigation of optimal page delivery by considering the case of a single page. We define a staleness-related cost which is used in our analysis of optimal page delivery strategies. Following this, we focus on solving the following two problems:

1. For a given limit on resource availability, determine the strategy that the

server should use to transmit page updates in order to minimise the staleness-related cost.

2. Determine the set of proxies such that caching the page at these proxies would lead to minimum staleness-related cost.

In Chapter 5 we extend our investigation to the general case of multiple pages and develop heuristic algorithms to determine which pages should be cached at which proxies. We also address the question of how the available capacity should be expended to transmit updates of the various cached pages. We thus end up with a complete picture of how the server should manage its resources so as to attain the minimum staleness-related cost for a given resource availability.

Chapter 6 extends our architecture to the case where pages are divided into fragments. We extend our performance model to describe the use of fragments and use simulation to study, in the context of frequent updates, the conditions under which the use of fragments would lead to a reduction in staleness-related cost.

Conclusions and suggestions for possible future work are summarised in Chapter 7.

Chapter 2

Related Work

In this chapter we review some of the work that has been done in designing information delivery systems. We first examine early approaches, some of which use one-way broadcast to deliver pages and some of which have two-way communication with the users. We then summarise efforts to analyse user behaviour in the context of caching and prefetching. This is followed by an overview of research concerning the web, including using push for web page delivery, web caching, and management of web page timeliness.

2.1 Early Information Delivery Systems

2.1.1 One-way Broadcast

Early examples of information delivery systems are television and radio. Servers (stations) provide information in the form of programmes to users by means of one-way broadcast. These programmes are transmitted at pre-determined times and multiple programmes may be transmitted simultaneously if more than one broadcast channel is available. The users learn of the transmission schedule, defining what channels are used for what programmes at what time, by means of an out-of-band directory. Users' interaction with the system consists of tuning in to the channel carrying the desired programme at the appropriate time. Such systems are said to be *push*-based, relying on server-initiated delivery of content.

It is somewhat problematic to define interesting performance metrics for radio and television. The systems make continual use of a broadcast channel, making resource consumption highly predictable. The response time for the system could be viewed as the length of time between the user's wish to watch a programme and the time that the broadcast of the programme commences. As there is no interaction between the users and servers the resource consumption and response time are very predictable and are independent of the number of users. This provides excellent scalability when faced with user population growth. However the

system does not perform as gracefully when required to deliver large numbers of programmes, assuming that the capacity of the channel is fixed. The number of programmes transmitted can be increased by using more broadcast channels. This would result in an increase in the resource requirements of the system. If the addition of broadcast channels is not possible, the transmission schedule must be adjusted to accommodate more programmes, resulting in a longer wait before a desired programme is transmitted.

One-way broadcast can also be used to deliver pages of information. The definition of what constitutes a page is application-specific but examples include pages of a book, images, sections of a map, etc. Suppose that a server possesses N pages, numbered 1 through N. They can be repeatedly transmitted over a broadcast channel using a transmission schedule. When a user desires one of the pages, his or her client can simply monitor the channel until the page of interest is received. Systems like this are referred to as *teletext* [22]. They share with television and radio the property that their behaviour is independent of the number of users. However, teletext user requests are far better defined than those for television and radio since a user interaction results in the need for a particular page. This allows us to define response time as the time from when a user requires a page until the time that the page's next transmission has completed. A thorough characterisation of this response time may be found in [23]. These results show that, if page requests arrive

according to a Poisson process, transmitting pages in a cyclic manner is optimal with respect to minimising the mean response time over all users. In [23–25] a scheme is presented that can produce such cycles yielding optimal or near-optimal mean response time. While this scheme designs the cycle off-line, Su and Tassiulas advocate an approach where the next page to transmit is selected at each transmission opportunity [26, 27]. This approach has the advantage of requiring no off-line computation and can produce transmission schedules that provide a response time very close to the minimum predicted in [24]. Bar-Noy *et al.* propose an efficient scheme to implement periodic transmission schedules like those generated by the algorithms above [28].

Clearly a server that follows the one-way broadcast model must transmit all of the pages in its possession as there is no way of knowing which pages are required and which are not. This means that teletext faces the same response time/channel use tradeoff experienced by radio and television. Increasing the number of pages reduces the frequency of each page's transmission, resulting in longer mean response times. Response time can be improved by using a faster channel or a larger number of channels. Datacycle, a relational database, explored the use of increased channel capacity to solve this problem [29, 30]. The Datacycle server operates by repeatedly transmitting its database over one or more broadcast channels. Applications pass database queries to a "record access manager" (at the application's host computer)

which translates these queries into filtering specifications. These specifications are presented to custom VLSI devices that passively and independently observe the broadcast channels, extracting records that match the filtering criteria. In terms of performance, the authors observe that reducing the response time requires either a shorter transmission cycle (requiring faster transmission) or additional transmission channels.

2.1.2 Two-way Schemes

The lack of upstream communication in one-way schemes means that the system must anticipate all users' page requirements, resulting in poor scalability as the number of pages becomes large. The availability of a reverse channel allows users to notify servers of the pages that they require. This is a *pull* model of information delivery, in contrast to the push model used by one-way systems. An early example of this type of system is *videotex*, described by Gecsei in [22]. In this system a reverse channel allows users to deliver page requests to the server, whereupon the server prepares the pages in question for transmission. Gecsei describes a range of systems, some of which (like INDAX) broadcast responses to all users. Others (such as PANDA, the Bildschirmtext commercial network, the Télétel network, and CAPTAIN) transmit a page to its intended recipient only. An analysis of

the performance of videotex can be found in [23, 31], user response time being the metric of interest. That study focused on broadcast-only videotex as well as systems that use broadcast for information pages and unicast for customised pages (such as when placing an order with a retailer, retrieving private information such as bank balances, etc.). Hybrid systems using both broadcast and unicast were shown to be able to handle higher traffic intensities than systems using only unicast. Further consideration is given to the performance of these systems in [32, 33] where the idea of on-demand multicast is introduced. It operates as follows. The server maintains a queue of pages to transmit. If an arriving request is for a page that is already queued for transmission then the sender of the request is added as a recipient for the transmission. Otherwise an entry for the page in question is placed in the queue. The scheduling algorithm used to select pages from the transmission queue may influence the response time of the system [33–35]. When the request rate is low, the scheduling algorithm has little impact on the response time. As the load increases, the Most Requests First algorithm yields the best response time for equal page request probabilities. If the request probabilities are unequal then the Longest Wait First algorithm is preferred.

An early example of a system that uses the hybrid broadcast-unicast technique is the Boston Community Information System [36]. A set of servers manages a group of databases; query routers make the servers appear as a single system image. User

terminals, the clients in the system, possess small local databases and are equipped with two-way channels to communicate with the servers. They also possess a radio receiver capable of capturing broadcast transmissions from the servers. Users begin their use of the system by specifying a set of queries, known as the "filter list", that is used to populate their local databases. Queries from the user may be processed locally if the relevant information resides in the local database. Otherwise the query will be forwarded to the appropriate server, which responds thereafter. Servers broadcast new data via the radio link. Users monitor these broadcasts for records matching their filter list and may update their local database with such records. The work is focused on the system's architecture rather than on performance evaluation.

The Boston Community Information System uses a wireless network for information delivery. More recent work has been done to support portable wireless devices which frequently depend on exhaustible power sources [37]. Reception of information is costly because the incoming packets must be examined by the CPU or equivalent hardware, placing a high demand on the device's energy supply. Minimising the frequency and extent of communication is therefore crucial to power conservation. These problems are outlined in detail in [37–39] and mechanisms for designing wireless broadcast schedules are presented in [39]. The idea is to intersperse the data itself with index information. The transmission is structured such that retrieving a given page requires a device to listen to the channel (known as

"tuning") for a short length of time. Indices may be complex in structure but all consist of pointers to either more refined indices or pages. The pointers can be used to determine the length of time until the indicated information will appear in the broadcast. Based on these indices the user only needs to tune in for several short sessions, each time retrieving either an index or the required data. The parameters of these indices can be adjusted to provide an appropriate balance between user response time and energy consumption.

2.2 User Behaviour: Caching and Prefetching

Initial studies on optimising the response time of teletext and videotex systems used the mean response time over all user requests as their performance metric. It is reasonable to expect that the performance experienced by individual users may differ from this average. Furthermore the inability to evaluate an individual user's response time makes it difficult to assess user-based optimisations. These issues were addressed for teletext in [40]. The request pattern of an individual user is formulated as a discrete state Markov chain. Analytic results for the mean response time of each user under a given broadcast cycle were obtained. These results are used to evaluate the benefits of prefetching, whereby the teletext terminal, under no instruction from the user, retrieves certain pages from the broadcast and stores

them locally. The pages to store are selected based on the belief that they will be required by the user in the near future. The scheme proposed in [40], called "linked pages", assumes that each page carries with it a list of the pages that are most likely to be requested next. The terminal will fetch and store these pages up to the limit of its local storage. Clearly this operation must be performed on a user-by-user basis as the decisions made depend on the particular pages the user retrieves. The reduction in response time using this scheme is evaluated for various capacities of local storage and is found to be substantial.

Per-user behaviour has also been considered in the context of "broadcast disks" [41–45]. The design goal of broadcast disks is similar to that of teletext. A one-way broadcast channel is used to disseminate a set of pages to a (potentially large) number of users. The composite access probabilities for each page, over all users, are known and are used to construct a broadcast cycle emulating a set of disks. The deviation of per-user request probabilities from the average is investigated and user-based solutions in the form of prefetching and caching are considered to be the best tools for reducing individual user response time. In [46] it is reported that cache maintenance based on techniques like the least-recently-used (LRU) algorithm may not be attractive as the time penalty for cache misses is not uniform for all pages. If the broadcast cycle was tuned to a particular user then that user could simply cache its hottest pages (*i.e.*, those that the user requires most frequently). However

this is not possible for every user when the user population is large.

Franklin *et al.* claim that, under certain conditions, the optimal cache replacement strategy is one that replaces the page with the lowest ratio of probability of access (P) to its frequency of broadcast (X), *i.e.*, the page with the lowest value of P/X [41]. This strategy is referred to as $\mathcal{PIX}$ and, while it performs well, it is not practical to implement. It depends on keeping cached pages sorted by P/X value and, furthermore, requires perfect knowledge of request probabilities. Approximations to $\mathcal{PIX}$ are presented that exhibit only a slight degradation in performance [41]. In these studies, the cost function of a particular schedule is the expected waiting time for the users. Bar-Noy *et al.* examine the case where the cost is a polynomial function of the waiting time [47]; this can be used to model a page transmission cost that changes over time.

The above page replacement algorithms were initially studied in the context of clients that perform no prefetching. In this context pages are only inserted into the cache when they are requested. We have seen that prefetching of pages is beneficial for teletext users, so it is no surprise that this technique is also applicable to broadcast disks. In [44] the "value" of a page is defined as the product of its probability of access (P) and the amount of time until it will next be seen (T) in the broadcast cycle. The $\mathcal{PT}$ algorithm is proposed where the value of the page being broadcast is compared with the least valuable page residing in the

cache. If the page being broadcast is more valuable, it will be captured, replacing the page in the cache. $\mathcal{PT}$ is found to provide lower latencies than systems that populate their cache solely on requests and use an approximation to $\mathcal{PIX}$ as a replacement strategy. Prefetching is therefore considered a valuable addition to a one-way broadcast delivery system, a conclusion that agrees with those made in [40].

2.3 The Web

The World Wide Web's core architecture is client/server in nature, with web browsers acting as clients that forward requests over an IP-based network to web servers [48]. As with the information delivery systems we have seen, information is organised into pages, forming the basic unit of addressability. Web servers face the scalability problems common to all client/server systems. An elementary queueing analysis shows that user response time increases dramatically as the request arrival rate approaches the request processing rate. The finite space allocated to the request queue can also lead to rejection of incoming requests when the server is very busy.

There are several solutions to this scalability problem. The most obvious is to use faster web servers, including the use of multi-processors. This increases

the request service rate, thereby supporting a higher request arrival rate before saturation takes place, but processing capacity is still limited by the service rate of the server. Multiple servers can also improve scalability if they are supported by an effective load sharing mechanism. Another approach to improving scalability is web caching, where proxy servers are placed at judicious points in the network and user requests are directed to these proxies rather than to the web server. The following discussion will not examine load sharing techniques. Rather it will concentrate on work done to increase scalability via the use of push and the management of caches.

2.3.1 Push for Web Page Delivery

Most of the schemes that have been proposed to deliver web pages via push have made use of IP Multicast. The original Internet Protocol was not designed to deliver packets to more than one recipient [49]. This capability was later added [50, 51]. The functionality, though not necessarily the mechanism, is as follows. Hosts subscribe to *multicast group addresses*. IP datagrams destined for such a group address will then be delivered to the hosts that have subscribed to the group. With proper group management, multicast groups can be used in similar ways to the broadcast channels discussed so far.

The implications of using push for web pages are discussed in [7]. The authors

claim that the lack of acceptance of push delivery for the web is caused by the conception of the web as a pull-based system and the worry that mechanisms to implement push entail too much overhead to attain any advantage. The authors proceed to present an architecture that is conceptually similar to videotex systems equipped with point-to-point and multicast facilities [7]. Pages are divided into three categories. Those that are extremely popular are delivered via cyclic multicast push where the pages are continually transmitted to a multicast group address. Those that are not frequently requested are transmitted using traditional unicast means. The remaining pages that are "reasonably popular" are delivered via on-demand multicast. Performance results suggest that multicast can be very effective in reducing the network resources consumed by a demanding user population as well as reducing server processing time. Similar conclusions are reached in [8]. In [9], a method that hashes URLs into multicast addresses is proposed, providing users with a straightforward method of subscribing to the multicast addresses appropriate to the pages that they have requested. Attempts are made to quantify the minimum number of recipients for which multicast may provide a savings of resources. Experiments suggest that groups of size greater than 3 may benefit. Literature argues that if multicast-like schemes are to be effective, the server should consider the link dependencies among pages [52].

Some of the above schemes require that the available pages be classified accord-

ing to their popularity. This can be done via user profiles or request rate estimates or, as in [53], by monitoring the request stream of infrequently requested pages and promoting a page as its popularity increases. A quantitative analysis of these types of approaches is given in [54].

Page classification is also key to the design of "Asynchronous Multicast Push" or AMP [10]. As in [7], data are classified by popularity and delivered by unicast pull, on-demand multicast, or cyclic multicast. The bandwidth reduction that multicast can provide is quantified, resulting from eliminating redundant transmissions along a given network link ("spatial" overlap) and by aggregating requests for the same page that arrive over a short interval ("temporal" overlap). In [55], Chuang and Sirbu claim that the ratio of the length of a multicast distribution tree to the mean length of a unicast routing path is approximately equal to $N^{0.8}$ where N is the number of members of the multicast group. This can be used to estimate the expected degree of spatial overlap. Nonnenmacher and Biersack evaluate AMP's potential gain from temporal overlap by constructing a probabilistic model of the number of requests for a popular page issued per minute throughout the day. For busy periods the potential gain can be large even if requests are aggregated over a short period of time. As expected, when the aggregation period is lengthened the gain increases.

The reliable delivery of pages is critical to the successful use of multicast for

information dissemination on the Internet. Core IP multicast does not ensure delivery of datagrams and many schemes designed for continuous media, such as RLM [56], only attempt to reduce packet loss. A summary of the difficulties inherent in reliable multicast and indications why traditional reliable unicast approaches are not appropriate can be found in [57]. For example, straightforward application of acknowledgements as used in protocols like TCP is not feasible because of the potentially large number of recipients and the resulting feedback implosion problem. This may be alleviated by relying on a small set of nodes to represent those nodes that are experiencing packet loss due to congestion [58, 59]. Other approaches to reducing the amount of acknowledgement information have been proposed. Concast allows aggregation of responses sent towards a common destination [60]. The continuous-media monitoring scheme implemented in the IVS conferencing system [61] uses probabilistic probing to solicit feedback information and to gauge the size of the multicast group. In [62, 63] Grossglauser attempts to carefully set NACK timeouts such that, in the face of bounded jitter, the volume of feedback is also bounded. This idea is taken further in [64] where the nodes cooperate to maintain state information and manage error recovery. Birk and Crupnicoff propose a scheme that uses erasure-correcting codes to facilitate reliable transmission of bulk data via multicast while avoiding feedback implosion [65].

Once client feedback has been summarised and aggregated a mechanism is

needed to deliver the missing packets to the appropriate recipients. "Scalable Reliable Multicast", as presented in [57], relies on recipients to transmit packets to others that have experienced losses. Repair requests are transmitted to the entire multicast group; any node capable of responding may do so. In order to avoid implosion, requests and responses are randomly delayed. Nodes suppress their request or response if they see a similar message during the delay.

Cyclic transmission provides its own resilience to packet loss at the expense of latency—users can simply wait until the corrupt data are once again transmitted. The effectiveness of this approach is analysed in [66] where simulation results show that for 50 users and a packet loss probability of 0.2, over 95% of the users can be assured of correctly receiving a page after 8 complete transmissions of the packets that make up the page. Note that on-demand multicast cannot benefit from this. Another approach to reliability uses parity packets, allowing complete reconstruction of a group of packets at a receiver if the fraction of packets received correctly is greater than or equal to a given value (such as k out of n) [10]. This is significant because different recipients may lose different packets. As long as each recipient correctly captures k or more packets, each can reconstruct the page. This can result in bandwidth savings over retransmitting the entire page if data are lost. A similar approach using forward error correcting codes is described by Rizzo and Vicisano [67].

2.3.2 Web Caching

Web caching has been a very popular technique to reduce response time. Hosts other than web servers, called *proxy caches*, maintain copies of popular web pages. Users then submit their requests to these proxies rather than to the web servers. Such hosts can collectively serve a larger number of users than could the server on its own, assuming that the users' requests are sensibly distributed amongst the proxies. A tremendous amount of work has been done in this area; what follows is not an exhaustive survey.

The central issues in cache design are cache placement (where the proxies should be placed within the network topology), cache population (which pages are placed in the cache and how they are obtained), page replacement (which pages should be evicted when more space is required in the cache), and cache consistency (how pages in the cache are managed when they are updated at their server).

A hierarchical arrangement of caches is generally suggested. This allows multiple users at one location to share the use of a cache, nearby locations to share a higher-level cache, and so on. Gwertzman *et al.* suggest that web servers be modified to monitor the popularity and access history of their documents [3]. When a document's popularity increases beyond a certain threshold, a copy of it is sent to a cache (caches are populated individually). The cache is chosen based on the

document request history with the goal of minimising the bandwidth spent serving requests for the page in question. Geographical proximity is used as a predictor of network proximity. These ideas are further explored in a quantitative manner in [68]. Hierarchical caching is also advocated in [8]. No guidelines are provided for automated cache placement, however; institution- and ISP-centric hierarchies are suggested. Furthermore it is recommended that these hierarchies be no more than three levels deep [8, 69].

It is generally believed that caches should contain popular pages. A straightforward approach to cache population is to simply place pages in the cache as they are requested and, as cache space is finite, remove pages whose request frequencies are low. This is the Least Frequently Used, or LFU, algorithm. Alternatively, when the cache becomes full, we can remove pages that have not been requested for a long time. This is the Least Recently Used, or LRU, algorithm. A cache is more effective, however, if it can be "pre-loaded" with pages in anticipation of users' requests. It is crucial to accurately predict the pages that will be required. Cohen *et al.* propose a solution to this problem that makes use of scheduled broadcasts from the server to the proxy caches [70].

An important observation about web traffic is that page requests from a fixed population (such as the users of a single cache) follow a "Zipf-like" distribution [71]. Zipf's law suggests that the number of requests for the ith most popular page

is proportional to $1/i$, meaning that the probability that a given request is for page i is G/i, where G is a normalisation constant [72]. Almeida *et al.* indicate that page selection probabilities follow this distribution precisely [73]. However, they also found that synthetic workloads generated according to a Zipf distribution yield higher cache hit ratios than those observed in reality. They conclude that this is caused by the Zipf distribution's failure to capture temporal and spatial locality of reference and proceed to construct a model capable of capturing these characteristics.

Breslau *et al.* found that the frequency of requests for the ith most popular web page is proportional to $1/i^{\alpha}$, where α is a constant that varies from observation to observation [71]. Specifically, they found α values ranging from 0.64 to 0.83, representing a nontrivial deviation from the Zipf distribution (indeed Marshall and Roadknight show that α varies considerably from one user to another [74]). A related observation is that less than a quarter of web documents are accessed multiple times but that requests for such documents can make up half of the requests [75]. Breslau *et al.* also conclude that there is little correlation between a document's popularity and its size and that there is a small but noticeable correlation between a document's popularity and the frequency of updates to that document; Douglis *et al.* suggest that these correlations depend on the type of pages being observed [75]. Cho and Garcia-Molina agree with this observation [76].

In addition to the page request probability, the temporal pattern of page requests has also been investigated. The request stream produced by a single user can be modelled as a two state process [77]. While in the "on" state the user requests pages with an inter-arrival time that follows a Weibull distribution; no requests are made while in the "off" state that represents the user think time. The times spent in these two states are described by heavy-tailed Weibull and Pareto distributions, respectively. These distributions are explained by the fact that many page requests are not initiated by the user. In-line images and other embedded objects are fetched by the web browser with no human intervention, resulting in bursts of requests following a user's selection of a page. The superposition of many such "on"/"off" processes with heavy-tailed periods results in self-similar traffic [78, 79]. The implication of this self-similarity is that burstiness is observed in the traffic at many different time scales—no smoothing is observed by averaging over a sufficiently long period. Self-similar traffic is also long-range dependent, meaning that observed behaviour at any time instant is typically correlated with all future behaviour. The degree of self-similarity of a series of observations may be captured by the *Hurst parameter*, H, $1/2 < H < 1$ (the degree increases as H approaches 1). Estimates of H for various web traces were found to be significantly different from $1/2$, suggesting self-similar behaviour [77].

Marshall and Roadknight plotted histograms of users' request rates observed at

a popular web cache and determined that there were few cache users who rarely made requests, suggesting that cache analysis can concentrate on those users who produce higher request arrival rates [74]. Barford *et al.* examined web traces from 1998 and 1995 and found that the effectiveness of caching had dropped over time and that size-based cache replacement policies were becoming less effective [80].

It is also important to note that packets making up the request stream from a given user are not independent; their inter-arrival time depends on the performance of the server [81]. Khaunte and Limb analyse the packet output of individual web users with the goal of modelling the utilisation of the upstream channel from a user. Results include characterisation of connection setup time, HTTP request packet size, parsing time, upstream IP packet size, and user think time. These results were then used to construct a simple finite state machine capable of simulating a web browsing session. The knowledge gained from user behaviour characterisation has been used to construct synthetic workloads for the evaluation of caching proxies, including SURGE [82], the Wisconsin Proxy Benchmark [83], the SPE architecture [84], and TPC-W [85].

2.3.3 Page Timeliness

The timeliness of cached pages is also of concern to cache users. Wessels recommended server-initiated call-back invalidations as a solution [2]; a survey of other early approaches can be found in [5]. Three common mechanisms used to ensure cache consistency are Time-to-Live (TTL) fields, client polling (or "proxy polling"), and invalidation protocols. TTL fields are used by web servers to instruct a proxy to dispose of pages after a pre-set timer has expired. When this happens, the next request for that page that arrives at the proxy will result in a query to the web server and thus the retrieval of the most recent version of the page. This technique is most suitable for pages that change on a regular basis and where the periods of the changes are known *a priori*. The performance of TTL has been studied in detail by Jung *et al.* [86].

With proxy polling a caching proxy periodically consults the web server to determine whether the cached content is up-to-date. Stale cache entries are removed. If polling is done when a page is requested, the user will have to wait for the proxy cache to hear from the server before the request can be processed. Cohen and Kaplan explore methods whereby the cache can perform such polling pro-actively [87, 88]. Gwertzman and Selezer recommend a version of client polling where the frequency of polls is determined by a threshold parameter and the length

of time the page has been in the cache [5]. Invalidation protocols, such as the Web Content Distribution Protocol [6], require servers to keep track of cache contents, *i.e.*, which caches hold copies of their pages, and transmit updates to these caches when required. Hybrid systems are also possible; Fei suggests that the server should select between invalidation and update-propagation for each document [89]. HTTP supports a complex caching architecture that makes use of many of the ideas that we have discussed so far [1].

Reddy and Pletcher argue that caching schemes should attempt to predict the future value of stored pages [90], such as by estimating the time until the next request for each page. Krishnamurthy and Wills attempt to provide cache coherency using "piggyback validation" [91]. Each time a cache needs to communicate with a server, it includes a list of pages that it thinks may be stale. The server, in conjunction with satisfying the request, indicates to the proxy which pages on the list have actually been changed. Krishnamurthy and Wills subsequently find that, for large caches, the cache coherency scheme has a strong influence on resource use (defined as a function of the number of requests that reach the server, network bandwidth, and latency) while the cache replacement policy has a larger effect for caches that are small [92]. Yu and Breslau provide a good overview of three basic consistency methods (TTL fields, invalidation messages, and leases) and then present a rather complex consistency scheme based on multicast invalidation messages and hierarchi-

cal caches [4]. Yin *et al.* explore the use of invalidation messages when a significant fraction of the web pages delivered by the server are generated dynamically [93], while Li and Cheriton construct a system using invalidation messages delivered via multicast [94]. Deolasee *et al.* have designed protocols that attempt to satisfy user timeliness requirements [95]. Some of these techniques are evaluated by Mikhailov and Wills [96].

Very few of the above techniques purport to offer strong consistency—indeed providing this is difficult without the use of a locking mechanism [4] or leases [97, 98]. It may also be impossible to ensure that cached pages are completely up-to-date even when the server transmits updates of cached pages to the proxies immediately after the updates are made. To see why this is so, consider the situation shown in Figure 2.1. Here a page, say page i, is updated at time t_0. At this time the server begins the process of sending the update to the proxies. At time t_2 the proxies receive the updated page and make it available to their users. Queueing and processing at the server, transmission time in the network, and processing at the proxies may all contribute to the delay between t_0 and t_2. If page i is updated between t_0 and t_2, such as at t_1, then the proxies will hold a version of the page that is not up-to-date, even though the server has transmitted the update at t_0 promptly.

Measures in connection with the degree of consistency that a system can provide

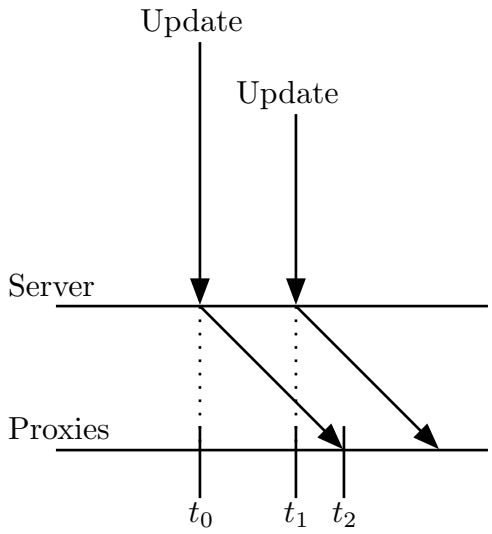

Figure 2.1: Why timeliness may be unobtainable

have been investigated. Our early research, described in [99], advocates measuring the length of time that a cached copy of a page differs from the copy stored at the server. Cho and Garcia-Molina have undertaken a study of web crawlers. Like caching proxies, crawlers store copies of pages at a location other than the server at which the pages are maintained [16, 17]. These copies are typically indexed, allowing users to perform searches over the pages that have been collected. It is important to ensure that searches return results that are currently valid. Cho and Garcia-Molina define two metrics, "freshness" and "age." A copy of a page is said to be "fresh" (and its freshness is equal to 1) if it holds the same content as the corresponding page stored at the server. Otherwise the page's freshness equals 0.

"Age" represents the time since the page was last fresh, or 0 if the page is currently fresh—this is essentially the same as the measure defined in [99]. Averages of these quantities over a set of stored pages can determine the aggregate freshness or age of the set.

In [16], update schedules suitable for minimising age or maximising expected freshness are presented. A similar staleness metric is suggested by Wolf *et al.* and is used as the basis for an optimisation problem to determine when pages should be crawled [19]. Coffman *et al.* consider a measure similar to Cho and Garcia-Molina's average freshness. They conclude that if page inter-update times are exponentially distributed then the times between successive crawler visits to a given page should be as equal as possible [20]. Edwards *et al.* construct a web crawler by formulating the minimisation of the number of "obsolete" pages as a nonlinear minimisation problem [21]. Yu and Vahdat have defined a metric similar to "age" but in the context of generic replicated systems [100]. Labrinidis and Roussopoulos similarly define staleness in the context of rendering web pages from content stored in a database, but they include the processing time at the server in the staleness computation [101, 102]. Dingle and Partl also use a similar definition of staleness, measuring it as the time elapsed since the "last-good time", or the last time that the page's content was identical at the server and at the cache in question [18]. Brewington and Cybenko argue for page inter-update times being

exponentially distributed and devise a metric they call "(α, β)-currency" that may be used to guide web crawler sampling processes [103].

Some other results concerning caching are worth mentioning. Liu *et al.* have found that at least 25% of the response time for retrieving a web page is comprised of connection setup and, furthermore, poor proxy cache design can increase the response time experienced by users [104]. An early study performed in 1995 used client instrumentation and found that 52% of all document requests were retrieved via hyperlinks (presumably including those embedded in a page and fetched automatically by the browser), 41% via the browser's "back" command, and the remaining small number by direct keyboard entry of URLs [105]. This suggests that users rarely know of the location of a document beforehand. Furthermore, only 2% of the documents requested were saved or printed. In [106] several web traces are used to evaluate cache performance with the conclusions that caches with sizes of 2 GB to 10 GB can yield hit ratios of between 24% and 45% with 85% of these hits being due to sharing amongst different users. We have not emphasised cache replacement algorithms, such as those described in [107–109]. While the necessarily finite size of caches makes this an important research area, it is beyond the scope of this thesis.

2.4 Conclusions

From our investigation we conclude that little attention has been paid to the delivery of popular pages that change frequently. The high request rates for such pages make caching attractive while their volatility compels the design of a robust staleness management architecture. Such an architecture is required because server and network resources must be expended to update pages stored by proxy caches. While staleness has been defined in several contexts, little attention has been paid to its impact on the quality of proxy responses. It is therefore appropriate to define the cost associated with staleness and use this definition to investigate the tradeoff between staleness-related cost and resource consumption.

Chapter 3

System Model

In this chapter we present in detail the architecture of the information delivery system under study. Sections 3.1 and 3.2 explain how the system operates, including the algorithms used by the system's components as well as the messages the components use to exchange control information and page data. This will be followed in Section 3.3 by a description of our performance model. We also include in Section 3.4 our definition of staleness.

3.1 System Architecture

The logical architecture of our system for the case of a single web server is shown

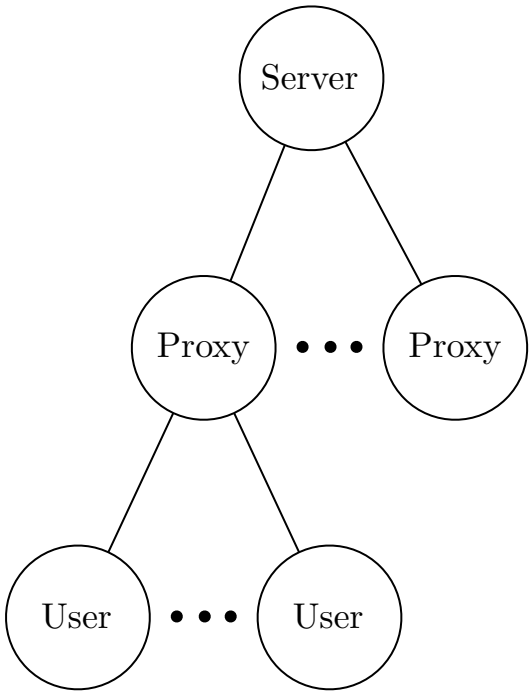

Figure 3.1: The system's logical architecture

in Figure 3.1. (Our architecture can easily be extended to systems with multiple servers.) The server provides content and is where pages are updated. A proxy can be viewed as a site-wide cache for a community of users. Each proxy caches a subset of the pages. Requests for cached pages are served from the copies at the proxy's cache. Pages that are not cached must be retrieved from the server.

For cached pages, updates made at the server are "pushed" to the proxies. Each proxy captures the updates that are transmitted to it, using them to refresh the copies of pages residing in its cache. We consider applications where a proxy

requires only the most recent update to a page in order for its copy of that page to be up-to-date. In other words, if a page is updated twice before the server has a chance to transmit the update, only the second update needs to be sent. Examples of applications where this approach can be used include delivery of current weather conditions and traffic information. Decisions on which pages are to be cached at a given proxy are made by the server. Specifically, the server may instruct the proxy to start caching a page that is not currently cached. Once the proxy has cached this page, the page will remain unchanged in the cache until the reception of the next update or until the server instructs the proxy to cease caching the page. We assume that proxies have enough space to cache all pages that may be required. We do not consider page replacement strategies in our study.

Each proxy keeps track of the rate of requests it receives for each page and periodically notifies the server of these request rates. The server monitors the update rate for each page. These update rates along with the request rate information received from the proxies will be used to determine which pages should be cached at which proxies and how the available resources should be used to transmit updates to the cached pages.

3.2 Server/Proxy Interaction

We now describe the interaction between the proxies and the server. We assume that each proxy has a *proxy ID*, allowing that proxy to be addressed individually. Each user is likewise assumed to have a unique identifier, called a *user ID*. Furthermore, each page is assumed to be assigned a unique and permanent *page ID*. This allows users, proxies, and the server to unambiguously and consistently refer to a particular page. Finally, we assume that the server, the proxies, and the users have a reliable means of communication.

A summary of the messages used by the proxies and the server is presented in Figure 3.2. REQ_RATES messages are sent by proxies to the server at regular intervals. Each such message contains a proxy's estimate of page request rates for a set of pages. A proxy will send a PAGE_REQ message when it needs to retrieve from the server a page that is not cached. The server will respond to such a request with a PAGE_REPLY message. The server transmits a PAGE_UPDATE message to a proxy in order to update the copy of a page that is in the proxy's cache. Each such message contains a page ID and updated content for the page. A PAGE_ADD or PAGE_REMOVE message is sent by the server to a proxy to instruct it to start or cease caching the specified page, respectively.

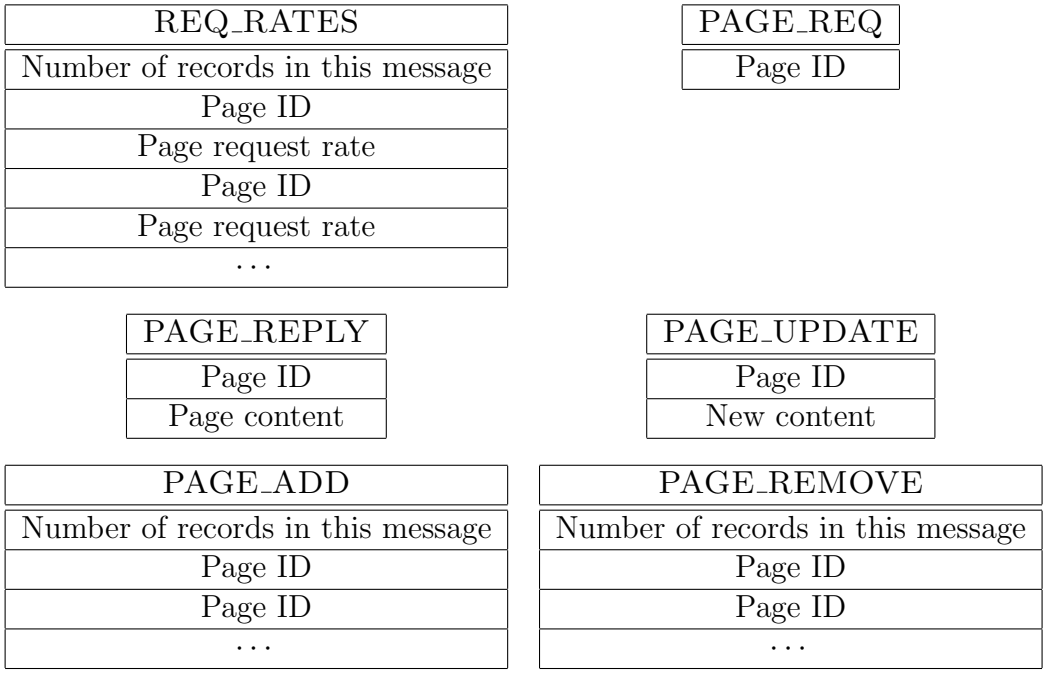

Figure 3.2: Messages used in the server–proxy protocol

3.2.1 Proxy Operation

At initialisation, a proxy creates and initialises the following data structures:

- The page cache, which is an array that maps page IDs to the contents of the cached pages;
- an array of counters, indexed by page ID, used to count page requests; and
- an array l of lists of user IDs, indexed by page ID, used to store the intended recipients of non-cached pages while waiting for them to be retrieved from

the server.

The proxy also starts a timer, referred to as the *request timer*, by setting its value to REQ_TIMER_INTERVAL. This timer is used to collect page request counts.

The proxy's actions upon receiving a page request are as follows. Suppose this request is for page i and is from user j. The proxy first increments the counter associated with page i. If this page is cached, the cached copy is transmitted to user j. Otherwise, the proxy checks the list of users contained in $l[i]$. If this list is empty, it means that the proxy is not currently waiting for the server to respond to a previous request for page i. In this case, a PAGE_REQ message is constructed and sent to the server and the user ID j is appended to $l[i]$. If, on the other hand, the proxy finds $l[i]$ to be non-empty, then the proxy is already waiting for page i to be transmitted by the server. In this case, user ID j is simply appended to $l[i]$.

When a PAGE_REPLY message for page i is received from the server, the proxy will send the retrieved page to each user in $l[i]$. Following this, $l[i]$ will be set to empty.

From time to time, the proxy receives PAGE_UPDATE messages from the server. When such a message is received, the proxy uses the supplied page ID and new content to update the cached copy of that page.

Also from time to time, the proxy receives PAGE_ADD or PAGE_REMOVE messages from the server. When such a message is received, the pages listed in the message are added to or removed from the cache as appropriate.

The proxy is responsible for the collection and reporting of page request rates. This is done whenever the request timer expires, which triggers the following operations. The proxy sends a REQ_RATES message to the server; this message reports the request rates for all pages that have been requested since the last report. The request rate for a given page, say page i, is given by the request counter value for the page divided by the value of REQ_TIMER_INTERVAL. The request counters are reset to zero and the request timer is restarted.

REQUEST_TIMER_INTERVAL is a tunable parameter. It should be large enough to accumulate a sufficient number of requests for the page request rates to be estimated with accuracy. However, if the timer interval is too large, the proxy may report the rates to the server too infrequently to reflect changing user behaviour.

3.2.2 Server Operation

At initialisation, the server creates and initialises the following data structures:

- An array of counters, indexed by page ID, to count page updates;

- an array u, indexed by page ID, to store the computed page update rates;
- an array to store, for each page, the proxies that cache the page;
- an array a, indexed by page ID and proxy ID, to store the page request rates received from the proxies.

The server also starts a timer, referred to as the *update timer*, and sets its value to UPDATE_TIMER_INTERVAL.

Recall that when a proxy receives a request for a page that is not cached, it will send a PAGE_REQ message to the server to retrieve that page. Upon receiving such a message, the server transmits the required page to the requesting proxy in the form of a PAGE_REPLY message.

When the server receives page request rates from a proxy in the form of a PAGE_UPDATE message, it updates the array a using the page IDs and page request rates contained in the message. More specifically, suppose the message is from proxy j. For each page ID i contained in the message, the content of $a[i, j]$ is updated using the page request rate for page i. For those pages that are cached at proxy j but not mentioned in the message, the corresponding $a[i, j]$s are set to zero.

Each time a page is updated, the update counter associated with that page is incremented by one. When the update timer expires, the server examines each page

i in turn. If the reported count for page i is greater than zero, the update rate of this page is computed by dividing the counter value by UPDATE_TIMER_INTERVAL; otherwise, the update rate is zero. The update rate is stored in $u[i]$ and the counter for page i is reset to zero. Once the page update rate of every page has been computed, the update timer is restarted.

The server uses the data in the arrays a and u to determine (i) the set of pages that should be cached at each proxy and (ii) the strategy for transmitting updates to these pages. How to do this will be explained in Chapters 4 and 5. For a given proxy, the server compares the set of pages that should be cached with the set of pages that are currently cached. This yields the pages that should be added to or removed from the cache and the corresponding PAGE_ADD and PAGE_REMOVE messages are sent to the proxy. When a page update is to be transmitted the server sends a PAGE_UPDATE message with the new page content to all proxies at which the page is cached.

It is worth noting that in this section we have described a simple method for computing the page request rates and page update rates. Specifically, each of these rates is computed using observations made during the last measurement interval. The system architecture is flexible and other approaches are possible. For example, the server could maintain a moving average for each request and update rate, thereby applying smoothing to the measured values. The proxies and the server

may also perform pre-processing of the raw measurement data. Exploration of the these approaches is beyond the scope of our study.

3.3 Performance Model

In this section, we describe the performance model used in our investigation. The features of the model are as follows:

System size We consider a system with one server providing N pages that are frequently updated. There are M proxies, each supporting its own community of users. The pages are assigned IDs $1, 2, \ldots, N$ while the proxies have IDs $1, 2, \ldots, M$.

Page updates We assume that the time between updates to page i follows an exponential distribution with mean $1/u_i$. This is consistent with previous measurements [16]. It follows that the update rate for page i is u_i.

Page requests We use a_{ij} to denote the rate at which requests for page i arrive at proxy j. This represents the combined rate of requests for page i from all users served by proxy j.

Processing of requests for non-cached pages In this investigation, we focus on resource requirements at the server; resource consumption within the net-

work and at the proxies is not taken into consideration. We assume that at the server, the processing of each request for a non-cached page requires one unit of resources.

Transmission of Updates The processing required to transmit an update to a cached page is assumed to consume P resource units.

Resource availability We assume that the server resources available to deliver frequently-updated pages amount to R resource units per second. These resources will be used to process requests for non-cached pages and to transmit updates to cached pages.

Note that we do not model the server's processing of the server-proxy messages described in Section 3.2. Instead, we assume that the capacity required for such processing has been deducted from the total system capacity when R is defined. Furthermore, we do not consider the resources required for proxies to process user requests. This is because more proxies can be added if the user request load becomes too high.

System performance will be evaluated based on the following metrics:

Cost of page delivery We assume that users would like to receive data that are current but that they may tolerate pages being somewhat out-of-date. We

provide a definition of page staleness in Section 3.4. Our focus, however, is on the cost resulting from the delivery of pages that are out of date. This cost will be defined in Section 4.3.

Resource usage In developing strategies for selecting pages to cache and for transmitting page updates, R resource units per second are available.

3.4 A Definition of Staleness

We now present our definition of staleness. Consider the example shown in Figure 3.3. At time t_0 an update to a page, say page i, is transmitted to the proxies. This page is not stale at time t_0 because the update has just been transmitted by the server. Suppose page i is updated at times t_1 and t_2, but an update is not transmitted until t_3. Page i is considered to be *stale* from t_1 to t_3. The *staleness* of page i, denoted by S_i, is defined to be the fraction of time that page i is stale [110]. Our definition is different from others [16, 18–21, 99] in the sense that the server is viewed as being responsible for keeping the pages up to date. Any staleness is caused by the server not being able to transmit updates promptly.

If a page is not cached, it is retrieved from the server on-demand; as a result, the page transmitted by the server is always up-to-date. In this case, we say the staleness of the page is zero.

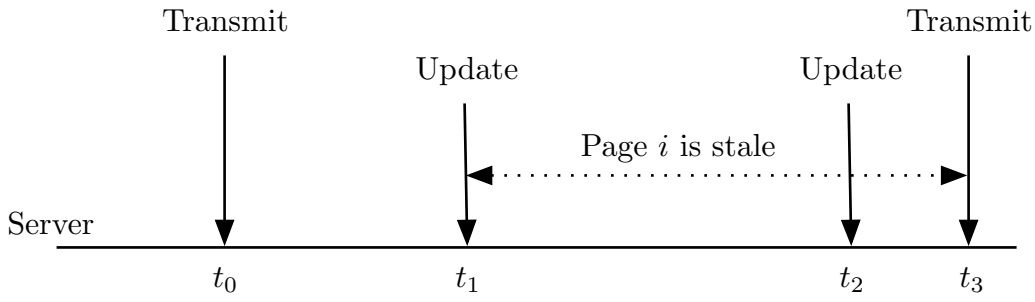

Figure 3.3: Definition of staleness

3.5 Concluding Remarks

In this chapter we have described the architecture of the system under study and discussed how such a system might operate. We have also presented the performance model that will guide our subsequent explorations. From this point onwards we will concentrate on answering the questions of which pages should be cached at which proxies and how the available resources should be used to transmit updates to cached pages.

Chapter 4

Optimal Page Delivery: Single Page Case

In this chapter we consider the delivery of a single page. This consists of determining when the server should transmit page updates and deciding which proxies should cache the page. The results presented in this chapter will be used in Chapter 5 to show how the server should manage multiple pages. In the discussion that follows, we will refer to a page that is not cached as a page delivered via "pull" because such a page is retrieved from the server when required.

The chapter is organised as follows. Section 4.1 introduces the conditions under which page staleness will be incurred, while Section 4.2 introduces the notion of transmission attempts and defines a staleness-related cost. Section 4.3 shows that

cost is minimised if the transmission attempts for a cached page are equally spaced. This is followed in Section 4.4 by an illustration of the cost/resource consumption tradeoff. Finally, in Section 4.5, we show that the page should be cached at either all of the proxies or at none of the proxies.

4.1 Preliminary Observations

Consider the delivery of a single page, say page i. Let R_i be the number of resource units per second devoted to page i. Also let $b_i = \sum_{j=1}^{M} a_{ij}$ be the total arrival rate of requests for page i among all the proxies. It is obvious that if $b_i \leq R_i$, then page i should be delivered via pull for all proxies because there is sufficient capacity to handle the requests. The resulting staleness will be zero. On the other hand, if $b_i > R_i$, some proxies must cache page i. Let G_i be the set of these proxies. It follows that the rate of requests for page i that are served via pull is

$$y_i = b_i - \sum_{j \in G_i} a_{ij} \tag{4.1}$$

If $R_i - y_i \geq Pu_i$ there are sufficient resources to transmit all updates to page i immediately and the resulting staleness is also zero. Otherwise, immediate transmission of all page updates is not possible and the cached copies of the page may

become stale.

4.2 Staleness-Related Cost

In this section, we focus on how updates should be transmitted when it is not possible to transmit them immediately. A staleness-related cost is also defined.

We consider an approach where the status of page i is checked at selected time instants. If this page has been updated since it was last checked, then the update is transmitted to the proxies in G_i; otherwise, no transmission is made. We refer to such an action as a *transmission attempt*. Note that the rate of transmission attempts is an upper bound on the rate at which updates are transmitted. The resources required to transmit updates can therefore be lowered by reducing the rate at which transmission attempts are made.

Consider a time interval of length T during which L transmission attempts for page i are made. Without loss of generality we assume that at the beginning of this interval, or at time 0, page i is not stale. Let x_1 be the time until the first transmission attempt and x_k be the time between the $(k-1)$th and kth transmission attempts, $k = 2, 3, \ldots, L$ (see Figure 4.1). We refer to the time between the $(k-1)$th and kth transmission attempts as "interval k." Within interval k, let w_k be the

amount of time that page i is stale. The staleness of page i is then given by

$$S_i = \frac{1}{T} \sum_{k=1}^{L} w_k$$

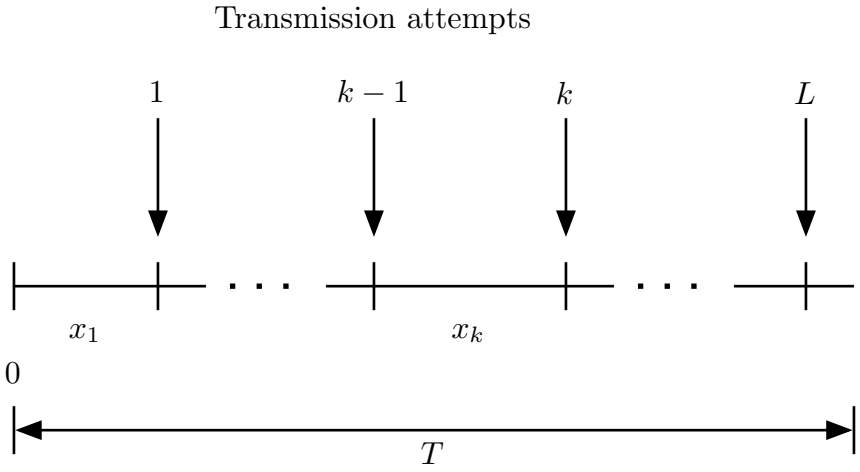

Figure 4.1: L transmission attempts in time T

In our study, the cost of page i being stale (denoted by q_i) is defined to be

$$q_i = (b_i - y_i) S_i \left(\frac{T}{L} \right) \tag{4.2}$$

q_i can be interpreted as follows. $b_i - y_i$ is the total arrival rate of requests for page i at those proxies where page i is cached (see Equation (4.1)) and S_i is an estimate of the probability that such requests will find that page i is stale. $(b_i - y_i)S_i$ is therefore an estimate of the rate at which requests for page i are served with stale

copies of the page. T/L is the mean time between transmission attempts for page i, and a larger mean implies that page i could be stale for a longer period of time. Our cost function q_i captures the combined effect of these two factors.

To compute q_i, we first rewrite Equation 4.2 as follows.

$$q_i = \frac{b_i - y_i}{L} \sum_{k=1}^{L} w_k$$

To obtain w_k, we condition on t, the time after the $(k-1)$th transmission attempt at which the first update to page i occurs. We thus have

$$w_k \mid t = \begin{cases} 0 & \text{if } t > x_k \\ x_k - t & \text{if } 0 \leq t \leq x_k \end{cases}$$

Recall that the time between updates is exponentially distributed. We remove the condition on t to yield

$$\begin{aligned} w_k &= \int_0^{x_k} (x_k - t) u_i e^{-u_i t} dt \\ &= \frac{e^{-u_i x_k} + u_i x_k - 1}{u_i} \end{aligned}$$

The cost q_i is therefore given by

$$q_i = \frac{b_i - y_i}{L} \sum_{k=1}^{L} \frac{e^{-u_i x_k} + u_i x_k - 1}{u_i}$$

4.3 Minimisation of Cost

Suppose that the server performs, on average, n_i transmission attempts per second for page i. n_i is given by L/T. We thus minimise

$$q_i = \frac{b_i - y_i}{n_i T} \sum_{k=1}^{n_i T} \frac{e^{-u_i x_k} + u_i x_k - 1}{u_i} \tag{4.3}$$

subject to

$$\sum_{k=1}^{n_i T} x_k = T$$

This problem can be solved by using the technique of Lagrangian multipliers as illustrated in Appendix A. The solution is

$$x_k = \frac{1}{n_i} \tag{4.4}$$

for all k. This result implies that, over the long term, transmission attempts for a given page should be equally spaced in order to minimise cost. This property is

not affected by the rate at which the page is updated. (Coffman *et al.* have arrived at a similar conclusion in the context of web crawler design [20].) Substituting Equation (4.4) into Equation (4.3), the optimal cost for page i (denoted by C_i) is given by:

$$C_i = \frac{b_i - y_i}{u_i}\left(e^{-\frac{u_i}{n_i}} + \frac{u_i}{n_i} - 1\right) \tag{4.5}$$

We next determine the relationship between n_i and R_i, the resources available to deliver page i. Based on our solution in Equation (4.4), a transmission attempt for page i will be made every $1/n_i$ seconds. Each such attempt will result in the transmission of an update with probability $1 - e^{-\frac{u_i}{n_i}}$. This is because the time between updates to page i is exponentially distributed. We thus have:

$$R_i = y_i + Pn_i\left(1 - e^{-\frac{u_i}{n_i}}\right)$$

This equation allows us to determine n_i such that the available server capacity will be fully used to transmit updates.

4.4 Cost/Resource Consumption Tradeoff

We now present a numerical example that illustrates the tradeoff between resource consumption and cost for the single page (page i). The parameters used in this

M	Number of proxies	1
b_i	Rate of requests for page i	1 request per second
P	Cost to transmit one page update to the proxy	1 resource unit

Table 4.1: Parameter values used to illustrate the cost/resource consumption trade-off

example are shown in Table 4.1. Note that because the page is cached at the proxy, $y_i = 0$.

In Figure 4.2, C_i is plotted against n_i for $u_i = 1, 2, 5, 10$, and 100. We observe that cost is a decreasing function of n_i. This is as expected—more transmission attempts per second should lead to lower cost. The rate of decrease, however, is a decreasing function of n_i. There is therefore a diminishing rate of return if we try to further reduce cost. For a given value of n_i, higher values of u_i result in higher cost. More frequent updates means there is a greater chance that the cached page has become stale since the last transmission attempt.

In Figure 4.3 n_i is plotted against R_i, the available resources for the delivery of page i. As expected, more frequent transmission attempts require higher resource expenditure. The additional resources required is more dramatic at $u_i = 5$ or $u_i = 10$ when compared to $u_i = 1$ or $u_i = 2$. This is due to the fact that at high update rates an update transmission is more likely at each transmission attempt. When $u_i = 100$, the update rate is so high that every transmission attempt results

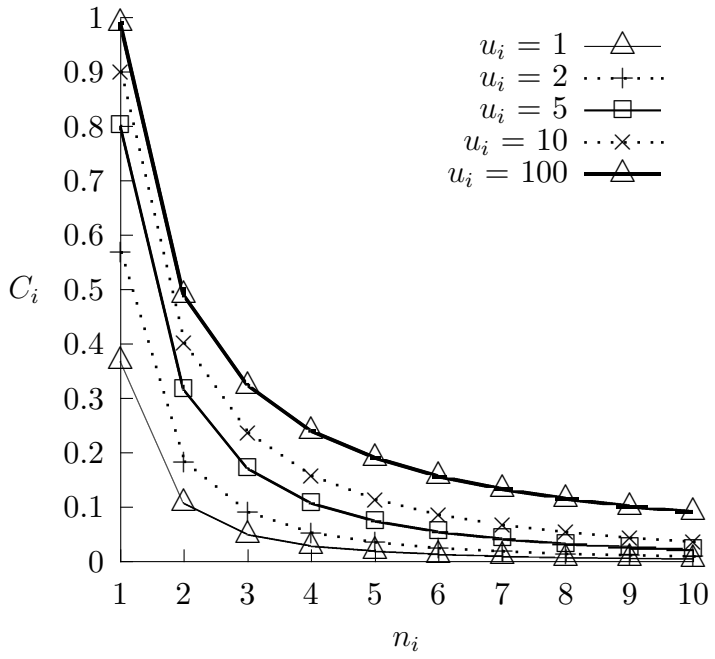

Figure 4.2: Optimal cost

in a transmission. This results in a plot of n_i vs. R_i that is essentially linear.

Figure 4.4 shows a plot of C_i vs. R_i using the results from Figures 4.2 and 4.3. It provides valuable insight into the tradeoff between resource consumption and cost. Once again, we observe that there is a diminishing rate of return in terms of decreased cost as more resources are made available to transmitting page updates. Also, with a higher update rate, more resources are required if we want to maintain the same cost.

It is worth noting that there is a simple relationship between cost and b_i, the arrival rate of requests for page i. From Equation (4.5) we can see that C_i is a linear

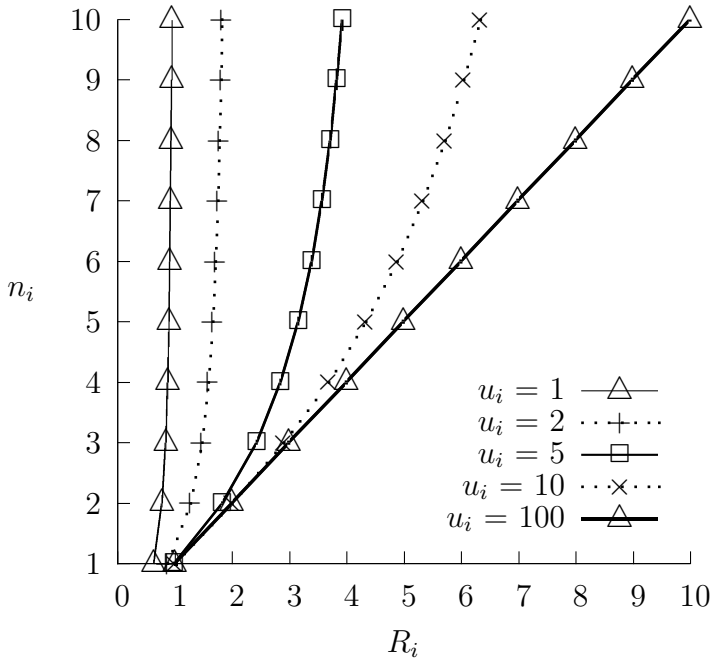

Figure 4.3: Resource consumption

function of b_i. An increase in b_i leads to a corresponding increase in cost for the same resource availability. Compared to Figure 4.4, the results in Figure 4.5 show a five-fold increase in cost when b_i is set to 5 while keeping the other parameters unchanged.

4.5 Where to Cache the Page

In this section, we investigate the problem of determining G_i (the set of proxies where page i is cached) such that the cost is minimised. As discussed previously,

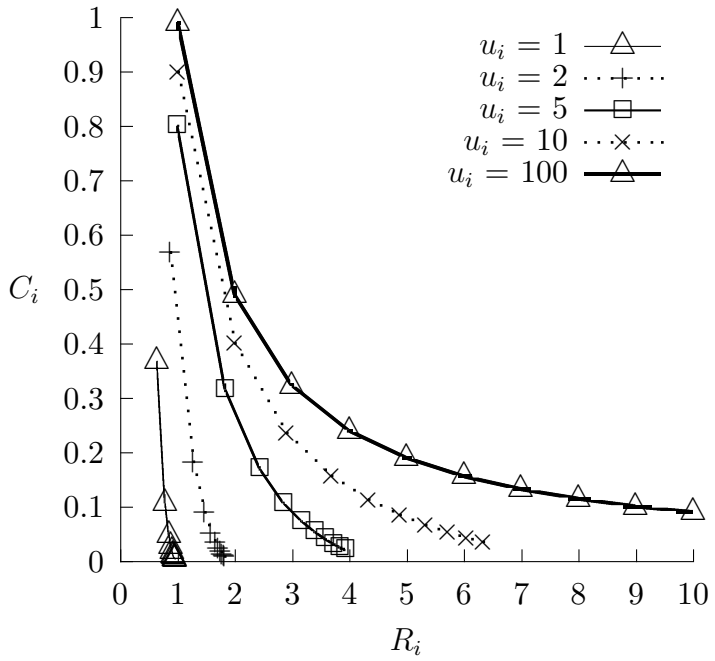

Figure 4.4: The tradeoff between resource consumption and cost

if $b_i \leq R_i$ then setting $G_i = \emptyset$ will yield zero staleness and therefore zero cost. We also noted that if $R_i - y_i \geq Pu_i$, there is sufficient resources to transmit all updates immediately. We can therefore attain zero cost by setting $G_i = \{1, 2, \ldots, M\}$. In what follows, we consider the case where $b_i > R_i$ and

$$R_i - y_i < Pu_i \tag{4.6}$$

We also require $R_i > y_i$; otherwise, the server does not have sufficient capacity for the pulled pages.

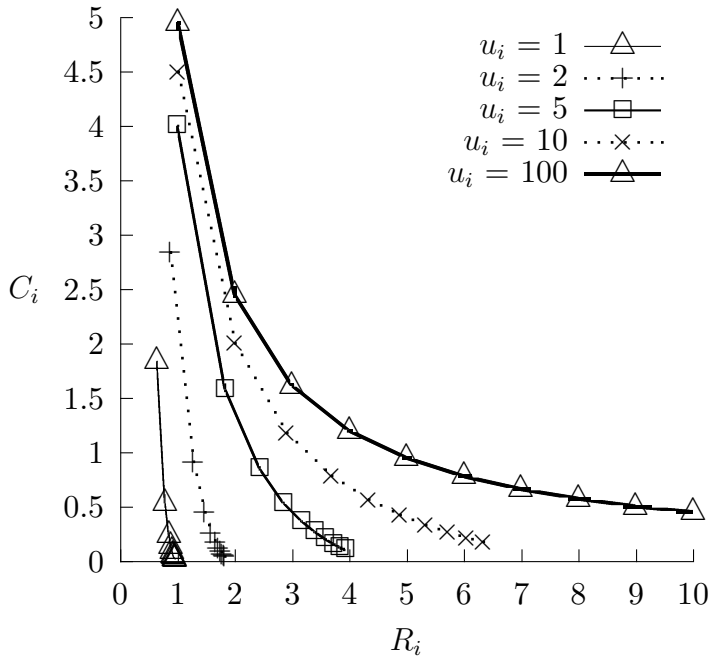

Figure 4.5: The tradeoff between resource consumption and cost, $b_i = 5$

We now show that the cost is minimised when page i is cached at all proxies. This is substantiated by the following theorem which implies that C_i is minimised when $y_i = 0$, *i.e.*, if we select $G_i = \{1, 2, \ldots, M\}$:

Theorem For P constant, we have $\frac{dC_i}{dy_i} > 0$ for all $y_i \in [0, R_i)$.

To prove this theorem, we first prove the following lemma.

Lemma Let $f(x) = x + 1 - e^x$. If $x > 0$ then $f(x) < 0$.

Proof: Using the Taylor expansion of e^x, $f(x)$ can be rewritten as

$$\begin{aligned} f(x) &= x + 1 - \left(1 + x + \sum_{i=2}^{\infty} \frac{x^i}{i!}\right) \\ &= -\sum_{i=2}^{\infty} \frac{x^i}{i!} \end{aligned}$$

From this we can clearly see that $f(x) < 0$ if $x > 0$. □

Proof of theorem: For ease of exposition, we drop the index i in our proof. Rewriting Equation (4.5) with the index i removed, we have

$$C = \frac{b-y}{u}\left(e^{-\frac{u}{n}} + \frac{u}{n} - 1\right) \tag{4.7}$$

Since the time between updates to the page is exponentially distributed, the rate at which updates are transmitted is $n\left(1 - e^{-\frac{u}{n}}\right)$. We select n such that all of the available capacity for the page is used, and thus have

$$Pn\left(1 - e^{-\frac{u}{n}}\right) = R - y \tag{4.8}$$

From Equation (4.7) and Equation (4.8) we get

$$C = \frac{b-y}{n}\left(\frac{y-R}{Pu}+1\right) \tag{4.9}$$

Differentiating both sides of Equation (4.9) with respect to y yields

$$\frac{dC}{dy} = \left(-\frac{1}{n}-\frac{b-y}{n^2}\frac{dn}{dy}\right)\left(\frac{y-R}{Pu}+1\right)+\frac{b-y}{n}\left(\frac{1}{Pu}-\frac{y-R}{P^2u}\frac{dP}{dy}\right) \tag{4.10}$$

Since P is constant, $\frac{dP}{dy}=0$ and Equation (4.10) is reduced to

$$\frac{dC}{dy} = \left(-\frac{1}{n}-\frac{b-y}{n^2}\frac{dn}{dy}\right)\left(\frac{y-R}{Pu}+1\right)+\frac{b-y}{Pun} \tag{4.11}$$

After some algebra, Equation (4.11) can be rewritten as

$$\begin{aligned}\frac{dC}{dy} &= \frac{1}{n}\left[\frac{b-y}{Pu}+\frac{R-y}{Pu}-\frac{b-y}{n}\frac{dn}{dy}\left(1-\frac{R-y}{Pu}\right)-1\right]\\ &= \frac{1}{n}\left[\left(\frac{R-y}{Pu}-1\right)\left(1+\frac{b-y}{n}\frac{dn}{dy}\right)+\frac{b-y}{Pu}\right]\end{aligned}$$

or

$$\frac{dC}{dy} = \frac{1}{n}\left[\left(\frac{R-y}{Pu}-1\right)g+\frac{b-y}{Pu}\right] \tag{4.12}$$

where

$$g = 1 + \frac{b-y}{n}\frac{dn}{dy} \tag{4.13}$$

Note that $\frac{b-y}{Pu} > 0$ because $b > R > y$. Also, from Equation (4.6), we have $\frac{R-y}{Pu} \leq 1$. We therefore conclude that $\frac{dC}{dy} > 0$ if $g < 0$.

We next show that $g < 0$. Differentiating both sides of Equation (4.8) with respect to y, and recognising that $\frac{dP}{dy} = 0$, we get

$$P\frac{dn}{dy}\left(1 - e^{-\frac{u}{n}} - \frac{u}{n}e^{-\frac{u}{n}}\right) = -1$$

Solving for $\frac{dn}{dy}$ and substituting the result into Equation (4.13) we have

$$\begin{aligned} g &= 1 + \frac{b-y}{Pn\left(e^{-\frac{u}{n}}\left(1+\frac{u}{n}\right)-1\right)} \\ &= 1 + \frac{b-Y}{Pue^{-\frac{u}{n}} - Pn\left(1-e^{-\frac{u}{n}}\right)} \end{aligned} \tag{4.14}$$

Substituting Equation (4.8) into Equation (4.14), we get

$$\begin{aligned} g &= 1 + \frac{b-y}{Pue^{-\frac{u}{n}} - (R-y)} \\ &= \frac{Pue^{-\frac{u}{n}} + b - R}{Pue^{-\frac{u}{n}} + y - R} \end{aligned} \tag{4.15}$$

The numerator is positive because $b > R$. We can therefore conclude that g is negative if the denominator of Equation (4.15) is negative. Let

$$h = Pue^{-\frac{u}{n}} + y - R$$

Using Equation (4.8), h can be written as

$$h = Pu\left[e^{-\frac{u}{n}} + \frac{n}{u}\left(e^{-\frac{u}{n}} - 1\right)\right]$$

Let $x = \frac{u}{n}$. We have,

$$\begin{aligned} h &= Pu\left[e^{-x} + \frac{1}{x}\left(e^{-x} - 1\right)\right] \\ &= Pu\left[\frac{e^{-x}}{x}\left(x + 1 - e^{x}\right)\right]. \end{aligned}$$

Applying our lemma we can see that $h < 0$ since $x > 0$. By Equation (4.15) this means that $g < 0$ and our theorem holds. □

Consider page i again. Our theorem tells us that when $b_i > R_i$, C_i is an increasing function of y_i for y_i in the interval $[0, R_i)$. This means that the cost of page i is minimised if $y_i = 0$, *i.e.*, $G_i = \{1, 2, \ldots, M\}$. In other words, the lowest cost is achieved if page i is cached at every proxy. On the other hand, if $b_i \leq R_i$ then

all requests for page i can be serviced directly by the server using pull, resulting in zero cost. We conclude that any page should either be cached at all proxies or at none of the proxies, depending on the resources available.

4.6 Concluding Remarks

This chapter has established the following important results regarding the delivery of a single page:

- If there is sufficient capacity, the page should not be cached at any of the proxies.
- If resources are insufficient for the server to process all requests via pull, staleness-related cost can be minimised as follows:
    1. the page should be cached at all proxies; and
    2. transmission attempts for the page should be equally-spaced, where the rate of transmission attempts depends on the resources available.

Although our study is focused on pages that are frequently updated, the above results are rather general in the sense that they are accurate regardless of the value of the page update rate.

Chapter 5

Optimal Page Delivery: Multiple Page Case

In this chapter we consider the case of multiple pages and show how the server can arrive at a near-optimal strategy for page delivery.

We assume that the system has a total capacity of R which is used to service all pages. From our results presented in Section 4.5, we know that each page should be either cached at all proxies or at none of the proxies. Let A be the set of pages that are cached and B be the set of pages that are not cached. We have $A \cup B = \{1, 2, \ldots, N\}$ where N is the total number of pages. Obviously, if $R \geq \sum_{i=1}^{N} b_i$, then all pages should be retrieved directly from the server (*i.e.*, $A = \emptyset$) because the resulting staleness, and therefore the cost, would be 0. However, when

$R < \sum_{i=1}^{N} b_i$, some pages must be cached. If $R \geq P \sum_{i=1}^{N} u_i$ then the server has sufficient capacity to transmit updates to all pages immediately. In this situation, all pages should be cached (*i.e.*, $A = \{1, 2, \ldots, N\}$) and the resulting cost will be 0. On the other hand, if $R < P \sum_{i=1}^{N} u_i$, then we need to determine the pages that should be cached and the rate of transmission attempts for each of the cached pages.

In this chapter we formulate an optimisation problem that can be used to determine the optimal page delivery strategy, *i.e.*, the strategy that would lead to minimum staleness-related cost. In general, an exact solution to this problem is very difficult to obtain. Our approach is to construct a near-optimal solution in two steps. In step 1, we determine, for a given set of cached pages, the rate of transmission attempts for each of these pages. Step 2 involves the development of heuristic algorithms to find the set of pages that should be cached.

In Section 5.1 we present the formulation of our optimisation problem. Section 5.2 is concerned with step 1 of our solution method where we use an approximation technique to obtain the rate of transmission attempts for each cached page. Section 5.3 describes our heuristics for determining which pages should be cached.

5.1 Optimisation Problem

Recall that A is used to denote the set of cached pages. Let

C_A = staleness-related cost associated with the cached pages

R_A = resources available for servicing the cached pages

When transmission attempts for page i are made regularly at rate n_i we have, from Equation (4.5),

$$C_A = \sum_{i \in A} \frac{b_i}{u_i} \left(e^{-\frac{u_i}{n_i}} + \frac{u_i}{n_i} - 1 \right)$$

where u_i and b_i are the update rate and total request arrival rate for page i, respectively. We have also shown that the resources required for a cached page, say page i, is $Pn_i \left(1 - e^{-\frac{u_i}{n_i}}\right)$. R_A is therefore given by

$$R_A = P \sum_{i \in A} n_i \left(1 - e^{-\frac{u_i}{n_i}}\right) \tag{5.1}$$

For the pages that are serviced via pull (denoted by the set B), the staleness-related cost is zero and the resource requirement is given by

$$R_B = \sum_{i \in B} b_i \tag{5.2}$$

We now present our optimisation problem which can be used to determine the optimal page delivery strategy.

Given R, P, u_i, and b_i, $i = 1, 2, \ldots, N$

Determine A and n_i for each $i \in A$ such that

$$C_A = \sum_{i \in A} \frac{b_i}{u_i} \left(e^{-\frac{u_i}{n_i}} + \frac{u_i}{n_i} - 1 \right)$$

is minimised subject to

$$R = P \sum_{i \in A} n_i \left(1 - e^{-\frac{u_i}{n_i}} \right) + \sum_{i \in B} b_i$$

An exact solution to this problem is difficult to obtain. We therefore focus on an approximate solution method that yields near-optimal results.

5.2 Determining Transmission Attempt Rates

In this section we describe step 1 of our solution method. We determine for a given set A the best value of n_i for each page i in A. The corresponding optimisation problem is

Given A, R_A, P, u_i, and b_i, $i \in A$

Determine n_i for each $i \in A$ such that

$$C_A = \sum_{i \in A} \frac{b_i}{u_i} \left(e^{-\frac{u_i}{n_i}} + \frac{u_i}{n_i} - 1 \right) \tag{5.3}$$

is minimised subject to

$$R_A = P \sum_{i \in A} n_i \left(1 - e^{-\frac{u_i}{n_i}} \right)$$

Determining the n_i's exactly is made difficult by the presence of $e^{-\frac{u_i}{n_i}}$ in Equation (5.3). However, given that the pages in A are updated frequently relative to the available server resources R, $\frac{u_i}{n_i}$ is likely to be large. In what follows, we present an approximate analysis based on the assumption that updates to pages in A are sufficiently frequent that $e^{-\frac{u_i}{n_i}}$ is negligible for all $i \in A$. To assess the accuracy of this assumption, consider Figure 5.1 which shows plots of $e^{-\frac{u_i}{n_i}} + \frac{u_i}{n_i} - 1$ and $\frac{u_i}{n_i} - 1$ versus $\frac{u_i}{n_i}$. The error introduced by our approximation when $\frac{u_i}{n_i} = 3$ is 2.4%. We can see that as $\frac{u_i}{n_i}$ increases, the error is reduced further.

Using the assumption that $e^{-\frac{u_i}{n_i}} = 0$, the optimisation problem is reduced to

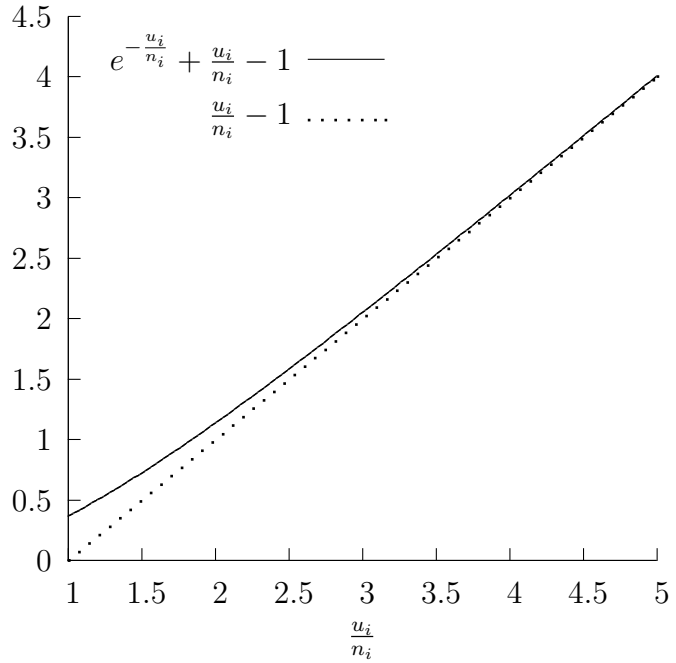

Figure 5.1: The effectiveness of the approximation used to determine n_i

Minimise

$$C_A = \sum_{i \in A} \left(\frac{b_i}{n_i} - \frac{b_i}{u_i} \right) \tag{5.4}$$

Subject to

$$R_A = P \sum_{i \in A} n_i \tag{5.5}$$

Noting that b_i/u_i is constant, this optimisation problem is similar to that investigated by Ammar and Wong [24]. Using the results from [24], the total cost is minimised when

$$n_i = K \frac{\sqrt{b_i}}{\sum_{k \in A} \sqrt{b_k}} \tag{5.6}$$

$\|A\|$	Number of pages in A	10
M	Number of proxies	1
P	Cost to transmit one page update to the proxy	1 resource unit

Table 5.1: Parameter values used to compute the numerical results

where

$$K = \sum_{i \in A} n_i = \frac{R_A}{P} \tag{5.7}$$

This result indicates that the rate at which transmission attempts for page i are made should be proportional to the square root of b_i. This relationship will be used in our heuristic for selecting pages to be included in A.

We next present numerical results that show the tradeoff between resource availability and cost when multiple pages are handled by the server. These results are based on the parameter values shown in Table 5.1. For each page in A, say page i, the total request arrival rate b_i was selected randomly between 1 and 100. The b_i's obtained are shown in Table 5.2. The corresponding transmission attempt rate n_i for $R_A = 50$ was then computed using Equation (5.6) and the results are also shown in Table 5.2.

As to the page update rate, we assume its value is the same for all pages in A and is given by u. For the above selection of parameters and $u = 25$, $\frac{u_i}{n_i} > 3.97$ for all i, meaning that our approximation method introduces little error. Note that

i	1	2	3	4	5	6	7	8	9	10
b_i	58	75	17	8	87	67	74	75	74	92
n_i	5.00	5.68	2.71	1.86	6.12	5.37	5.64	5.68	5.64	6.29

Table 5.2: Page request rates and corresponding transmission attempt rates for $R_A = 50$

smaller values of R_A or larger values of u will result in an even larger value for $\frac{u_i}{n_i}$. In Figure 5.2 we plot the staleness-related cost C_A against the resource availability R_A for different values of u. In this figure $u \geq 25$ and the range of R_A considered is from 10 to 50. As expected, when the system has few resources available, the cost C_A is relatively high. As we allocate more resources, the staleness-related cost decreases.

5.3 Selecting Cached Pages

In this section, we consider the problem of finding the set of pages to place in A that leads to minimum cost. This is step 2 of our solution method. One possibility is to perform an exhaustive search of all elements of the power set of $\{1, 2, \ldots, N\}$ but this is difficult to compute when N is large. Our approach is to obtain heuristic, near-optimal solutions that are computationally efficient.

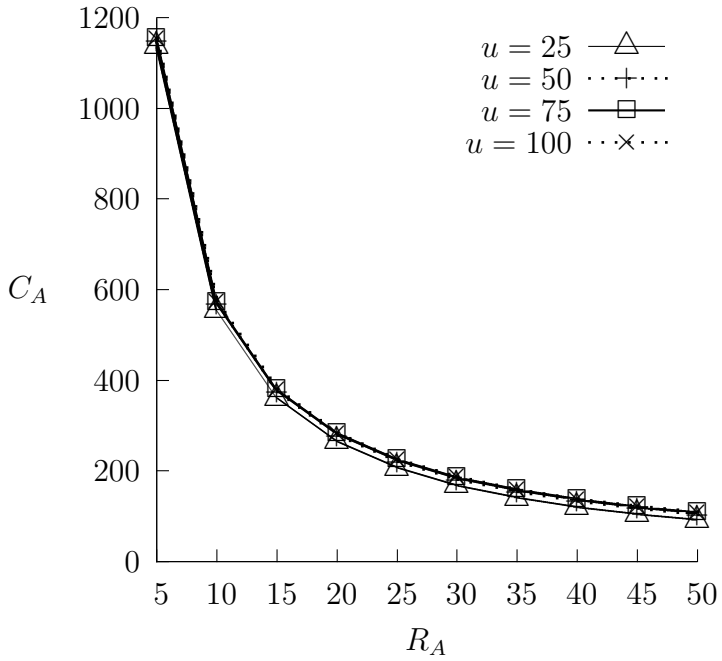

Figure 5.2: Cost versus resource consumption

5.3.1 Cost Analysis

We first analyse the impact on cost when a page, say page j, is moved from set A to set B. $A - \{j\}$ and $B \cup \{j\}$ are the sets resulting from this move. Let f be the difference in cost. Since pages in B are served via pull, they do not incur any cost. f is therefore given by

$$f = C_A - C_{A-\{j\}} \tag{5.8}$$

If $f > 0$ then moving page j from A to B would lead to a reduction in cost.

To determine f, we again assume that $e^{-\frac{u_i}{n_i}}$ is negligible. Suppose the rate of

transmission attempts for page i is n_i when the set of cached pages is A and m_i when the set of cached pages is $A - \{j\}$. Substituting Equation (5.4) into Equation (5.8) and simplifying, we get the following expression for f:

$$f = \sum_{i \in A - \{j\}} b_i \left(\frac{1}{n_i} - \frac{1}{m_i} \right) + b_j \left(\frac{1}{n_j} - \frac{1}{u_j} \right) \tag{5.9}$$

We have from Equation (5.6) that n_i (or m_i) should be proportional to the square root of b_i. We can therefore write:

$$n_i = \frac{K\sqrt{b_i}}{X} \tag{5.10}$$

where $K = \sum_{i \in A} n_i$ and $X = \sum_{i \in A} \sqrt{b_i}$. Similarly,

$$m_i = \frac{K_j\sqrt{b_i}}{X_j} \tag{5.11}$$

where $K_j = \sum_{i \in A - \{j\}} m_i$ and $X_j = \sum_{i \in A - \{j\}} \sqrt{b_i}$.

Substituting Equations (5.10) and (5.11) into Equation (5.9) and simplifying, we get

$$f = \sum_{i \in A - \{j\}} \sqrt{b_i} \left(\frac{X}{K} - \frac{X_j}{K_j} \right) + \frac{X\sqrt{b_j}}{K} - \frac{b_j}{u_j}$$

Since $X = X_j + \sqrt{b_j}$ we have

$$\begin{aligned} f &= X_j \left(\frac{X_j}{K} + \frac{\sqrt{b_j}}{K} - \frac{X_j}{K_j} \right) + \frac{X_j \sqrt{b_j}}{X} + \frac{b_j}{K} - \frac{b_j}{u_j} \\ &= X_j \left(\frac{2\sqrt{b_j}}{K} + \frac{X_j}{K} - \frac{X_j}{K_j} \right) + \frac{b_j}{K} - \frac{b_j}{u_j} \\ &= \frac{2X_j \sqrt{b_j} + X_j^2 + b_j}{K} - \frac{X_j^2}{K_j} - \frac{b_j}{u_j} \end{aligned} \tag{5.12}$$

We now determine expressions for K and K_j. Recall that the resource requirements for the pages in A and B are $R_A = P \sum_{i \in A} n_i$ and $R_B = \sum_{i \in B} b_i$, respectively. (see Equations (5.5) and (5.2)). Since the system has total capacity R, we have $R_A + R_B = R$. Using Equation (5.7), it follows that

$$K = \frac{R - R_B}{P} \tag{5.13}$$

Similarly,

$$K_j = \frac{R - R_B - b_j}{P} \tag{5.14}$$

Substituting Equations (5.13) and (5.14) into Equation (5.12) yields

$$f = P \left(\frac{2X_j \sqrt{b_j} + X_j^2 + b_j}{R - R_B} \right) - \frac{P X_j^2}{R - R_B - b_j} - \frac{b_j}{u_j} \tag{5.15}$$

Note that the cost savings f, as given by Equation (5.15) above, can be computed efficiently. It will be used as the basis for our first heuristic.

5.3.2 Heuristic Algorithm I

Our first heuristic algorithm makes use of the cost analysis presented in the previous subsection. We begin by placing all pages in A. A page in A is called a "candidate" page if (i) there are cost savings in moving this page to B, and (ii) the total resource requirement after the move is not more than R. We then find the candidate page that yields the largest cost savings and move that page to B. This step is repeated until no candidate pages remain in A. Pseudocode for this algorithm is shown in Figure 5.3.

We next evaluate the merit of heuristic algorithm I. Our evaluation is based on (i) accuracy and (ii) efficiency in terms of the amount of computation required. We first note that the optimal solution for the set A can be obtained by exhaustive search. In our investigation, the accuracy of heuristic algorithm I is assessed by comparing its result to this optimal solution.

We ran experiments for a range of values of the number of pages N. For a given value of N, the input parameters are determined as summarised in Table 5.3. These parameter values were selected such that $e^{-\frac{u_i}{n_i}}$ is negligible for all i, meaning that

```
HEURISTIC I(a, u, P, R)
  A ← {1, 2, ..., N}
  B ← ∅
  repeat
        candidatefound ← FALSE
        fbest ← 0
        for each j in A
        do if R − Σ_{i∈B} b[i] − b[j] > 0 and f(A − {j}, B, j) > fbest
              then fbest ← f(A − {j}, B, j)
                   jbest ← j
                   candidatefound ← TRUE
        if candidatefound
          then A ← A − {jbest}
               B ← B ∪ {jbest}
    until candidatefound = FALSE
  return A
```

Figure 5.3: Heuristic algorithm I

our approximation introduces little error. In each experiment, the best cost found by heuristic algorithm I is compared with the optimal cost found by exhaustive search. The difference between the two, expressed as a percentage of the optimal cost, is then calculated. The above procedure was replicated 10000 times and the

b_i	Randomly selected between 1 and 100, $i = 1, 2, \ldots, N$
u_i	Randomly selected between 1000 and 10000, $i = 1, 2, \ldots, N$
P	Randomly selected between 1 and N
R	Randomly selected between 1 and $50N$

Table 5.3: Parameter values used in our experiments

N	p	d
10	0.19%	0.5%
15	0.27%	0.3%
20	0.18%	0.4%
25	0.17%	0.2%

Table 5.4: Results for heuristic I

results are summarised in Table 5.4.[1] In this table, p is the percentage of replications where heuristic algorithm I did not find the optimal cost. Of those replications, d is the mean percentage difference between the two costs. We observe that heuristic algorithm I yields good results. In most cases, it selects the best pages to place in A and when it does not, the increase in cost is low.

As to efficiency, heuristic algorithm I is expected to examine $O(N^2)$ possible sets A. The computational requirement may be excessive if the number of pages N is large.

5.3.3 Heuristic Algorithm II

In this subsection we develop a second heuristic algorithm, referred to as heuristic algorithm II, that has lower computational requirements than those of heuristic algorithm I. This heuristic is motivated by the observation that for most solutions

[1]Because of computational complexity, we were only able to obtain results for exhaustive search for $N \leq 25$.

found by heuristic algorithm I, the pages in A are requested more frequently than those in B. (This is true for the results shown in Table 5.4 and for other values of $N > 25$ where it is not possible to obtain results using exhaustive search.) This implies that the smallest request rate among the pages in A is greater than the largest request rate among the pages in B. This observation suggests the existence of a threshold value H such that $b_i > H$ for all $i \in A$ and $b_i < H$ for all $i \in B$.

We note that analytic results for H are difficult to obtain. Heuristic algorithm II is designed to use this threshold property to determine A efficiently as follows. All pages are initially placed in A. Let E_k $(k = 1, 2, \ldots, N)$ be the cost, as given by Equation (5.4), when the k least popular pages are moved to B. Let $E_{k_{\min}}$ be the smallest value among the E_k's. Heuristic algorithm II places the $k_{\min}$ least popular pages in B, provided that the total resource requirement is not larger than R. The details of this algorithm are illustrated in Figure 5.4. The computational requirement is in general $O(N \log N)$, which is an improvement over the $O(N^2)$ required by heuristic algorithm I.

We next evaluate the merit of heuristic algorithm II. The approach is the same as that used when heuristic algorithm I was investigated and the results are shown in Table 5.5. We observe that heuristic algorithm II also yields good results. Although it is slightly inferior to heuristic algorithm I in terms of the fraction of replications where it finds the best solution, the resulting costs tend to be closer to optimal

```
HEURISTIC II(a, u, P, R)
  for j ← 1 to N
  do
     B[j] ← j
  SORT-BY-INCREASING-B(B, b)
  Y ← 0
  c_best ← ∞
  k_best ← 0
  for k ← 1 to N
  do
     Y ← Y + b[B[k]]
     if R − Y < 0
       then break
     if C(B, k, a, u, P, R) < c_best
       then c_best ← C(B, k, a, u, P, R)
            k_best ← k
  A ← {1, 2, ..., N} − {B[1], B[2], ..., B[k_best]}
  return A
```

Figure 5.4: Heuristic algorithm II

than those provided by heuristic algorithm I.

So far, only results for $N \leq 25$ have been presented because of the considerable resources required to do the exhaustive search. We have run experiments comparing the performance of the two heuristic algorithms for larger N, $N = 50, 100, 500, 1000, 10000$. The results are summarised in Table 5.6. In this table p' is the percentage of replications where the two algorithms did not find the same cost, d' is the maximum percentage difference between the two costs, and t is the execution time of heuristic algorithm II expressed as a fraction of the execution

N	p	d
10	0.42%	0.2%
15	0.48%	0.1%
20	0.45%	0.2%
25	0.46%	0.08%

Table 5.5: Results for heuristic algorithm II

N	p'	d'	t
50	0.38%	0.039%	0.52
100	0.34%	0.014%	0.39
500	0.61%	0.0036%	0.50
1000	0.61%	0.0029%	0.51
10000	0.49%	0.000014%	0.54

Table 5.6: The two heuristic algorithms for large N

time of heuristic algorithm I. We can see that the results found by the two heuristic algorithms are very similar, suggesting that they lead to similar solutions. Table 5.6 also confirms that heuristic algorithm II has noticeably faster execution time than heuristic algorithm I.

Based on the results in this subsection, heuristic algorithm II should be the preferred algorithm because it yields similar solutions to those found by heuristic algorithm I but is noticeably faster in terms of execution time.

5.3.4 Examples

We now present some examples illustrating the details of the solutions found by our heuristic algorithms. For these examples, N was set to 10 and M was set to 1. Table 5.7 shows the values of P, R, b_i, and u_i for each example, along with the resulting set A. The results found by our two heuristic algorithms are identical and match the optimal solutions found by exhaustive search.

The results in Table 5.7 show that most pages will be placed in A and cached at the proxies. This suggests that although non-cached pages have zero cost, the server capacity required for their delivery is usually better spent reducing the cost of cached pages by means of more frequent transmission attempts. This is supported by comparing Example 5 with Example 6 in Table 5.7. They share the same page request and update rates but differ in availability of server resources. In Example 5, $R = 100$ and seven out of the ten pages are cached. However, if we reduce R to 10 as in Example 6, all ten pages are cached. Comparing Examples 5 and 6 with Examples 7 and 8 suggests that the above observation is not affected by P, the cost of transmitting a page update.

5.4 Concluding Remarks

In this chapter we have presented two heuristic algorithms that determine which pages should be cached at the proxies and find the rate of transmission attempts for cached pages. Both algorithms are for the case where $e^{-\frac{u_i}{n_i}}$ is negligible and, under this condition, they yield near-optimal staleness-related cost. Heuristic algorithm II should be the preferred algorithm when one takes into consideration computational complexity.

The results presented in this chapter allow us to fill in the following pieces of the system operation described in Section 3.2:

- The results presented in Section 5.2 allow the server to determine when to transmit page updates to the proxies.
- Our page selection heuristics, explained in Section 5.3, allow the server to determine which pages should be cached by the proxies.

These are significant results because they, in conjunction with the framework outlined in Chapter 3, provide a complete description of the server's operation. A server implemented following these guidelines will be able to effectively allocate its processing resources so as to deliver pages with near-minimal staleness-related cost.

Example	P	R	1	2	3	4	5	6	7	8	9	10	
1	1	100	86	99	77	75	1	2	48	66	12	60	b_i
			42	57	96	24	25	94	20	27	51	7	u_i
			{1, 2, 3, 4, 7, 8, 9, 10}										A
2	1	100	36	17	67	88	5	57	86	21	81	96	b_i
			85	100	11	80	45	27	88	97	11	49	u_i
			{1, 2, 3, 4, 6, 7, 8, 9, 10}										A
3	1	100	4	32	60	91	27	61	19	62	91	25	b_i
			12	59	8	16	18	41	20	54	32	58	u_i
			{2, 3, 4, 5, 6, 7, 8, 9, 10}										A
4	1	100	51	61	3	97	92	54	15	71	54	94	b_i
			10	61	89	75	55	29	24	73	71	92	u_i
			{1, 2, 4, 5, 6, 7, 8, 9, 10}										A
5	1	100	5	89	7	73	6	69	9	85	73	69	b_i
			94	55	70	56	13	55	32	66	47	70	u_i
			{2, 4, 6, 7, 8, 9, 10}										A
6	1	10	5	89	7	73	6	69	9	85	73	69	b_i
			94	55	70	56	13	55	32	66	47	70	u_i
			{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}										A
7	10	100	5	89	7	73	6	69	9	85	73	69	b_i
			94	55	70	56	13	55	32	66	47	70	u_i
			{2, 4, 6, 7, 8, 9, 10}										A
8	10	10	5	89	7	73	6	69	9	85	73	69	b_i
			94	55	70	56	13	55	32	66	47	70	u_i
			{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}										A

Table 5.7: Examples illustrating the solutions found by the page selection heuristic

Chapter 6

Page Fragments

In the previous two chapters, we have obtained the following results that can be used to develop strategies for delivering frequently-updated web pages such that the staleness-related cost is close to optimal:

- For a page that is cached, transmission attempts should be made at regular intervals.
- Any given page should either be cached at all proxies or at none of the proxies.
- Two heuristic algorithms have been developed that decide which pages to cache, and for each cached page, the rate at which transmission attempts should be made.

These results were developed for the case where updates are made on a per-page basis.

It has been suggested that efficiency can be improved by organising pages as a number of fragments [11–15]. In this chapter, we apply the above results to the case of page fragments with a view of understanding the conditions under which the use of fragments will lead to a reduction in cost.

This chapter is organised as follows. In Section 6.1 we describe how page fragments work and give an example of how they can be used. In Section 6.2 we extend the architecture described in Section 3.2 to include fragments. Section 6.3 shows how our performance model can be adjusted to accommodate this extended architecture. Section 6.4 describes our discrete event simulation used to evaluate the performance gain from using fragments. In Section 6.5 we present our simulation results, together with a discussion of these results.

6.1 How Page Fragments Work

Page fragments can best be explained by means of an example. Consider the delivery of personalised pages. In its most basic form, this involves customisation of the page content based on the requesting user and is increasingly common in applications like electronic commerce and sports event reporting [11]. An example

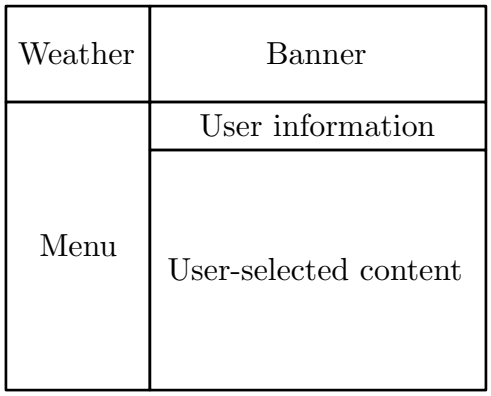

Figure 6.1: Sample layout for a personalised page

of a personalised page is illustrated in Figure 6.1. This page consists of weather conditions in the upper left corner; a banner to indicate the page's purpose; a section specific to the requesting user (containing, for example, a greeting); a menu listing the options from which the user may select; and a section that shows contents selected by the user, *e.g.*, news, sports, traffic conditions, or financial information. The above elements are referred to as *fragments*. In general, each fragment is used in the construction of one or more pages. A page therefore consists of its corresponding set of fragments.

Use of fragments may lead to improved efficiency. This is substantiated by the following observations, in the context of our example of a personalised page:

1. Personalised pages are constructed by the system in response to a request made by a user. If the concept of fragments is not used, each page may

only be of interest to a single user, reducing the effectiveness of the proxy caches. With page fragments, the same fragment may be used in multiple pages, representing a form of aliasing [111]. This suggests that caching of fragments could lead to improved scalability.

2. When a fragment common to many pages changes, only the fragment in question is affected. Consequently, if the fragment is cached at a proxy, the server only needs to transmit the update of this fragment. The situation is quite different if fragments are not used. Consider the example in Figure 6.1. The weather conditions are used in many different pages. Each page containing this information will change when the weather information is updated. If these pages are cached, considerable resources are required to transmit updates for each page when compared to those required to transmit fragment updates.

Various studies have explored caching of fragments, instead of pages, at the proxies as a means of providing scalable delivery of personalised content [11–15]. The proxies assemble the fragments to construct the pages that are requested by the users. We refer to pages assembled in such a fashion as "assembled pages".

6.2 System Architecture

In this section we extend the architecture described in Section 3.2 to include fragments. All fragments are stored at the server and are updated by an external process. Each user sends page requests to its proxy. Periodically each proxy notifies the server of the aggregate request rate of each fragment, defined to be the sum of the request rates for pages that use that fragment. Once again we focus on fragments that are updated frequently.

The server determines which fragments should be cached at which proxies. For those fragments that are cached, updates at the server are transmitted to the proxies. Upon receiving a request for a page, a proxy assembles the requested page using the required fragments and transmits it to the requesting user. During assembly, any required fragments that are not cached at the proxy are retrieved from the server.

6.3 Performance Model

To model the system that caches fragments at the proxies, we modify the model introduced in Section 3.3 as follows. There are F fragments in the system that are frequently updated. These fragments are used to construct N pages. We assume that the time between updates to fragment f follows an exponential distribution

with mean $1/\upsilon_f$. The amount of server resources devoted to delivering frequently-updated fragments is R units per second.

We define a function ϕ that describes the set of fragments that make up each of the N pages. For page i,

$$\phi(i) = \{f \mid \text{fragment } f \text{ is used to construct page } i\}$$

We also define the function π where

$$\pi(f) = \{i \mid f \in \phi(i)\} \tag{6.1}$$

i.e., $\pi(f)$ is the set of pages that contain fragment f. Note that one can determine π when ϕ is specified and vice versa. We may therefore use whichever is convenient to refer to a specific page-fragment mapping. For fragments that are cached, the server uses transmission attempts to send updates to the proxies.

For the case where fragments are updated frequently, it may not be possible to push fragment updates to the proxies immediately. This means that the fragments stored at the proxies may become stale. We use the same definition for the staleness of a fragment as we have already introduced for the staleness of a page. Specifically, a fragment is stale if it is cached and the server is not able to transmit updates

promptly to the proxies. The staleness of a fragment is the fraction of time that it is stale. A fragment that is not cached is never stale.

An assembled page is said to be stale if any of its constituent fragments is stale. The staleness of an assembled page is the fraction of time that it is stale.

Transmission attempts are used to keep the fragments that are cached at the proxies up to date. We note that the request rate of a given fragment is equal to the sum of request rates for those pages that contain the fragment. Let the request rate for fragment f be β_f. β_f is given by

$$\beta_f = \sum_{i \in \pi(f)} b_i \tag{6.2}$$

where b_i is the request rate for page i over all proxies as defined in Section 4.1. β_f will be used to compute the rate of transmission attempts for each fragment using the results outlined in Section 5.2.

Once again we focus on the resource requirements at the server. We assume that at the server, the processing of a request for a fragment that is not cached requires γ units of resources. Correspondingly, the processing required to transmit an update to a cached fragment is assumed to consume γP resource units.

6.3.1 Staleness-Related Cost

In this section, we define staleness-related cost in the context of pages that are assembled from fragments. For the case where fragments are not used, the staleness-related cost was defined in Section 4.2. Specifically, the cost for page i is given by Equation (4.2), *i.e.*,

$$q_i = (b_i - y_i)S_i \left(\frac{T}{L}\right)$$

where $(b_i - y_i)$ is the rate of requests for page i that arrive at those proxies that cache page i, S_i is the staleness of page i, and T/L is the average time between transmission attempts for page i.

Consider now the case of assembled pages. Similarly to the case of non-assembled pages, the staleness-related cost of page i is defined to be:

$$C_i' = (b_i - y_i')S_i'\frac{T}{L}$$

where y_i' is the sum of arrival rates of requests for page i at those proxies where none of the fragments belonging to page i are cached and S_i' is the staleness of assembled page i as defined earlier in this section. T/L is the average time between transmission attempts for page i. We note that, for an assembled page, transmission attempts are defined at the fragment level and not at the page level. An estimate

of T/L is therefore required.

In estimating T/L, our goal is to use an estimate that would lead to a fair performance comparison between when fragments are used and when fragments are not used. We note that when transmission attempts for page i are made at the page level, page i is known to be not stale immediately after each transmission attempt—either page i is not stale because no update has been made, or an update has been made and the update is transmitted. The latter case is more likely because page i is frequently updated. When fragments are used, a transmission attempt for a fragment belonging to page i may result in page i being not stale. If this is indeed the case, then we refer to such a transmission attempt as a *pseudo transmission attempt* for page i. We therefore re-write our definition for C'_i as follows:

$$C'_i = (b_i - y'_i)S'_i E_i \tag{6.3}$$

where E_i is the average time between pseudo transmission attempts for page i.

Finally, the staleness-related cost, over all pages, can be written as:

$$C' = \sum_{i=1}^{N} C'_i \tag{6.4}$$

An analytic characterisation of S'_i and E_i is extremely difficult. However, they

υ_f	The update rate for fragment f
b_i	The request rate for page i
$\phi(i)$	The fragments contained in page i
R	The capacity of the server used for frequently-updated fragments
γ	The relative cost of fragment delivery
$\mathcal{A}'$	The set of fragments that are cached

Table 6.1: Simulation inputs when fragments are cached

can be obtained by simulation. The details are described in the next section.

6.4 Simulator Description

We have used discrete event simulation to evaluate the performance gain from using fragments. Two simulators have been developed, corresponding to whether fragments or pages are cached at the proxies, respectively.

We first describe the case where fragments are cached at the proxies. The inputs to the simulation are listed in Table 6.1. Using Equations (6.1) and (6.2) along with the inputs b_i and ϕ_i, we can calculate β_f, the fragment request rate of fragment f, and $\pi(f)$, the set of pages that contain fragment f, $f = 1, 2, \ldots, F$. The availability of β_f allows us to determine the transmission attempt rate of f (denoted by ω_f) from Equation (5.6), for each fragment that is cached (*i.e.*, for each $f \in \mathcal{A}'$).

The υ_f's and ω_f's are the parameters needed to generate events that correspond

to updates and transmission attempts for each of the fragments. When these events are processed, data will be collected for S_i' and E_i (the staleness and the average time between pseudo transmission attempts for page i, respectively), $i = 1, 2, \ldots, N$. Our approach to data collection is as follows.

During the simulation, we can keep track of, for each fragment, whether the fragment is stale or not. Let the set of fragments within page i that are stale be D_i. Clearly $D_i \subseteq \phi(i)$. When a fragment within $\phi(i)$ becomes stale, this fragment will be added to D_i and $|D_i|$ is increased by one. On the other hand, when the server makes a transmission attempt for a fragment within D_i, this fragment is removed from D_i and $|D_i|$ is decreased by one. Recall that page i is not stale when none of its constituent fragments are stale. Therefore, a pseudo transmission attempt for page i occurs when a fragment transmission attempt results in $|D_i|$ being reduced from 1 to 0. In the simulation, the number of occurrences of pseudo transmission attempts for page i is recorded. The data collected allows us to compute E_i. We also collect data that allows the computation of S_i', the staleness of page i. This is given by the fraction of time that $|D_i| > 0$.

S_i' and E_i can then be used in Equations (6.3) and (6.4) to determine the staleness-related cost.

Consider now the case where pages are cached instead of fragments. To provide a fair comparison, we stay with a structure where each page has a number of

υ_f	The update rate for fragment f
b_i	The request rate for page i
$\phi(i)$	The fragments contained in page i
R	The capacity of the server used for frequently-updated pages
A	The set of pages that are cached

Table 6.2: Simulation inputs when pages are cached

fragments and updates are made at the fragment level. The difference, however, is that when a fragment is updated, all pages that contain this fragment will be affected, and these pages are considered as being updated at the same time.

The inputs to this simulation are listed in Table 6.2. Once again, $\pi(i)$ can be computed from ϕ_i using Equation (6.1). Because a page is updated each time one of its constituent fragments is updated, the update rate of page i, $i = 1, 2, \ldots, N$, can be computed as follows:

$$u_i = \sum_{f \in \phi(i)} \upsilon_f$$

The request arrival rates for the various pages are given by the b_i's and are used to determine the transmission attempt rate of each cached page i (denoted by n_i) using Equation (5.6). The υ_f's and n_f's are then used to generate events that correspond to fragment updates and page transmission attempts, respectively. When these events are processed, the total time that page i, $i = 1, 2, \ldots, N$, is stale is collected. This allows us to compute S_i, the fraction of time that page i is stale. S_i can then

be used to compute the staleness-related cost for page i.

6.5 Simulation Results and Discussion

We now present simulation results to illustrate the conditions under which the use of fragments will lead to a reduction in cost. This is accomplished by comparing the staleness-related cost when fragments are used to that when fragments are not used.

In our simulation, we consider the case where the system has a total of N pages and F frequently-changing fragments. Because these fragments are updated frequently, we assume that the update rate of each fragment is sufficiently high that $e^{-\frac{\upsilon_f}{\omega_f}} = 0$ for $f = 1, 2, \ldots, F$. We further assume that each fragment has an update rate of υ.

We generate ϕ mappings as follows. Let r_i be the number of constituent fragments in page i, $i = 1, 2, \ldots, N$. r_i is selected according to a uniform distribution between 1 and $2J - 1$ (the mean is J). The r_i fragments in the page are then selected randomly from the F fragments in the system. This process is repeated for all N pages. If in the resulting ϕ every fragment is not used in at least one page, a new ϕ is generated. This process is repeated until every fragment is used in at least one page.

Recall that $\pi(f)$ is the set of pages that contains fragment f. Also, for a given ϕ, one can determine π. From π we can determine the number of pages that use fragment f. Let m be the average of $|\pi(f)|$ for $f = 1, 2, \ldots F$. We have

$$m = \frac{1}{F} \sum_{f=1}^{F} |\pi(f)|$$

m represents the amount of fragment sharing among the pages. In general, a higher value of m means more sharing of fragments among pages.

We set γ, the relative processing cost of delivering a fragment, to $1/J$. This is based on the assumption that delivering a page's worth of data requires the same amount of processing no matter how many fragments are used to construct the page. P, the relative processing requirements for push, was set to 1.

The following metric will be used in our evaluation:

$$g = \frac{C}{C'} = \frac{\text{cost when fragments are not used}}{\text{cost when fragments are used}}$$

g can be interpreted as follows. If $g > 1$, the use of fragments will lead to a lower cost. Our focus is on conditions under which $g > 1$. In each of the experiments that follow, ten ϕ mappings were generated using the procedure described above and C' and C were collected for each, yielding ten g values. The mean of these ten runs

Factor	Label	Low Level	High Level
N	A	250	500
F	B	100	200
J	C	4	8
υ	D	80	120
Request rate distribution	E	Uniform	Zipf-like ($\alpha = 0.25$)

Table 6.3: Levels of the five factors for the $2^5 \cdot 10$ factorial design

is used as the result of the experiment. Each simulation was terminated after 30 seconds of simulated time. This run length was sufficient to give a 95% confidence interval width for g that was less than 0.1% of the mean when 10 replications were examined.

Our first experiments consist of a $2^5 \cdot 10$ factorial design to determine the effects on g of various factors [112]. The factors and levels under consideration are presented in Table 6.3. The rationale behind our choices is as follows.

The number of pages, N If fragments are not used, the server must split its available capacity between all of the cached pages. If there are many such pages, this can lead to infrequent transmission attempts for each page. However, if fragments are used, the total number of pages does not directly affect the server's resource use. This suggests that g will be influenced by N. We select 250 and 500 as sizes representing small to moderate page populations.

The number of fragments, F When there are many fragments, the server will

have only a small amount of capacity to devote to each, meaning that their transmission attempt rates may be low. This will affect C'. However, F has no direct effect when fragments are not used. We select 100 and 200 as levels representing small to moderate numbers of fragments, respectively.

The mean number of fragments per page, J For a given number of pages and a given number of fragments, the more fragments that appear in each page, the more pages will contain each fragment. In other words, for a fixed N and F, a higher value of J will result in a higher value of m. This means that there will be more sharing of fragments between pages. If many pages share a fragment, transmission of that fragment has the potential to make all those pages no longer stale. However, more fragments in a page may make it less likely that a fragment transmission reduces $|D_i|$ from 1 to 0. This would tend to increase the time between pseudo transmission attempts. We therefore expect J to have a significant impact on g. Levels of 4 and 8 are chosen to represent pages made of moderate to high numbers of fragments

The page request rate, b_i The page request rates will directly influence the fraction of server capacity allocated to each page or fragment (see Equation (5.6)). Because the transmission attempt rate of a page or fragment will have an effect on that page or fragment's staleness, the request rates of the different

pages, given by the b_i's, may affect g. A uniform distribution for the b_i's is a simple baseline reference case, while a Zipf-like distribution which assigns page i a request rate proportional to $1/i^{\alpha}$ represents a better match to real web traffic [71, 73, 74]. α was chosen to be 0.25, ensuring that, while there is variety in the page request rates, no page is requested extremely rarely. For the case of a uniform distribution, $b_i = U/N$ where U is the total request rate for all pages. When the Zipf-like distribution is used, the page request rates are scaled so that the total rate of requests is also equal to U. Hence,

$$b_i = G\frac{1}{i^{\alpha}}$$

where

$$G = \frac{U}{\sum_{k=1}^{N} \frac{1}{k^{\alpha}}}$$

The fragment update rate, υ The update rate of the fragments may have an effect on g. We have selected 80 and 120 as levels for υ because they represent a range of update rates for fragments that are updated frequently. Using these values of υ, over 90% of the values of $\frac{\upsilon_f}{\omega_f}$ in these experiments were greater than 2.5. This means that our approximation method in Chapter 5 introduces little error.

N	Number of pages	250
F	Number of fragments	100
J	Mean number of fragments per page	4
R	Server capacity	500 units per second
b_i	Request rate for page i	50 requests per second
υ_i	Update rate for fragment i	80 updates per second

Table 6.4: Parameter values used to test for the effect of fragment transmission attempt order

We first confirm that the order in which fragment transmission attempts are scheduled does not affect our estimate of g. To do this, the first transmission attempt for fragment f, $f \in \mathcal{A}'$, is scheduled at a random time between the start of simulation and $1/\omega_f$. We ran 100 simulations using the parameter values shown in Table 6.4 and the values of g were recorded. The width of the 95% confidence interval of these values was found to be less than 0.1% of the mean. This suggests that using a random offset for each initial transmission attempt will not have a significant impact on the value of g.

We now present our results for the $2^5 \cdot 10$ factorial design. The mean value of g over all replications of all experiments was found to be 3.799203. In Table 6.5 we show the fraction of variation in g explained by the various factors and their interactions. The corresponding coefficients are also shown [112]. We can see that the number of fragments F contributes most to the variation, followed by the number of pages N. The effect of the interaction between N and F is also

Factors	Coefficient	Contribution	Factors	Coefficient	Contribution
A	1.252775	0.341664	ABD	0.052217	0.000594
B	-1.554833	0.526286	ABE	0.011425	0.000028
C	0.371620	0.030064	ACD	-0.021273	0.000099
D	-0.226941	0.011212	ACE	0.001593	0.000001
E	-0.046193	0.000465	ADE	-0.001348	0
AB	-0.518832	0.058601	BCD	0.056208	0.000688
AC	0.121309	0.003204	BCE	0.006591	0.000009
AD	-0.073185	0.001166	BDE	-0.001926	0.000001
AE	-0.041538	0.000376	CDE	-0.001483	0
BC	-0.259662	0.014678	ABCD	0.010022	0.000022
BD	0.144116	0.004521	ABCE	-0.000143	0
BE	0.011165	0.000027	ABDE	0.006141	0.000008
CD	-0.093376	0.001898	ACDE	0.001384	0
CE	0.000153	0	BCDE	-0.003436	0.000003
DE	-0.000915	0	ABCDE	-0.007569	0.000012
ABC	-0.088818	0.001717			

Table 6.5: Results for the $2^5 \cdot 10$ factorial design

significant. This is followed in importance by J, the number of fragments per page. Neither the fragment update rate nor the request rate distribution make a strong contribution to the observed variation in g. The fraction of variation that is unexplained by the factors, and is therefore attributed to errors, was computed to be 0.002656. This does not represent a significant contribution to the variation in g.

Based on this analysis, we select N, F, and J for further exploration. Because the page request distribution does not make a significant contribution to the variability of g, we use the same page request rate for each page. Unless stated

R	Server capacity	1000 units per second
b	Request rate of each page	50 requests per second
υ	Update rate of each fragment	100 updates per second

Table 6.6: Parameter values used in our numerical examples

	$F = 100$		$F = 200$	
N	m	g	m	g
100	3.93	0.716	—	—
150	5.87	0.838	—	—
200	8.08	0.957	—	—
250	9.92	1.10	5.05	0.255
300	12.0	1.23	6.05	0.303
350	14.1	1.36	7.02	0.343
400	16.0	1.50	8.05	0.371
450	18.3	1.55	9.02	0.419
500	20.0	1.79	9.96	0.467

Table 6.7: g versus N

otherwise, the parameter values shown in Table 6.6 will be used for the simulations that follow. In the experiments that follow, 75% of the $\frac{\upsilon_f}{\omega_f}$ values are greater than 2.5.

Table 6.7 shows simulation results for various values of N, the total number of pages. The mean number of fragments per page, J, was set to 4, while results were obtained for F (the total number of fragments) equal to 100 and 200. We observe that a large N tends to favour the use of fragments. This can be explained as follows. In our selection of parameters, increasing N when F and J are fixed

will result in an increase in m, the number of pages that use each fragment. The benefit of using fragments is seen when $m \geq 9.92$ and $F = 100$. This indicates the level of fragment sharing required to realise a performance gain.

As to the interaction between N and F, we observe that for a given N, an increase in F from 100 to 200 results in less benefit from using fragments. This is due to the fact that a larger F leads to less sharing of fragments and fewer resources being available per fragment, thereby reducing the attractiveness of using fragments.

We next explore the effect of J, the mean number of fragments that are contained in each page. Table 6.8 shows g for various values of J and various combinations of N and F. We observe that an increase in J tends to decrease the attractiveness of fragments. This is because the presence of more fragments within a given page means that more fragments may become stale and the time between pseudo transmission attempts for the page may be increased. This is the case even if there is extensive sharing of fragments.

Finally, we examine the effect of the number of fragments F. Results for the cases of $N = 300$ and $N = 500$ are shown in Table 6.9. We observe that the attractiveness of fragments may be reduced if the total number of fragments F is large, an observation that can be confirmed by examination of Tables 6.7 and 6.8. This is because the more fragments the server must deliver, the fewer resources will

	$F = 100$ $N = 100$		$F = 100$ $N = 300$		$F = 300$ $N = 300$	
J	m	g	m	g	m	g
2	—	—	6.11	1.99	—	—
3	3.04	0.750	9.01	1.45	—	—
4	4.12	0.630	12.0	1.18	—	—
5	4.91	0.643	15.2	1.15	4.95	0.248
6	5.930	0.563	17.7	1.19	5.80	0.293
7	6.950	0.558	21.1	1.21	7.05	0.311
8	8.02	0.571	23.5	1.37	8.01	0.330

Table 6.8: g versus J

	$N = 300$		$N = 500$	
F	m	g	m	g
50	23.9	11.6	40.4	18.3
100	11.9	1.28	19.9	1.87
150	7.92	0.454	13.4	0.667
200	6.00	0.311	9.99	0.469
250	4.74	0.257	8.05	0.387
300	—	—	6.64	0.351

Table 6.9: g versus F

be available for each. This leads to increased fragment staleness and thus increased staleness-related cost for the assembled pages. The interaction between F and N is consistent with that observed in Table 6.7.

6.6 Concluding Remarks

In this chapter we have applied our architecture to the delivery of pages that are assembled from fragments. We have shown that the use of fragments can lead to savings in staleness-related cost if the following conditions are met:

Sharing of fragments between pages In general, the more pages that contain a given fragment, the greater the potential benefit from using fragments.

Small number of fragments per page The more fragments that are contained in any given page, the greater the chance that at least one of them will be stale. This means that the page is more likely to be stale and its staleness-related cost will rise.

Total number of fragments is small If there are too many fragments, the server will have insufficient resources to transmit each promptly. This will lead to increased staleness of the assembled pages.

Chapter 7

Summary and Future Work

7.1 Summary of Contributions

This thesis has focused on resource management issues inherent in designing information delivery systems for pages that are frequently requested and frequently updated. Our investigation has been based on an architecture where pages are cached at proxy servers and page updates are pushed to these proxies.

We began by presenting a measure of staleness and defining the cost of pages being stale. Our approach differs from others in that we take the position that the server is responsible for keeping the pages up to date. Any staleness is caused by the server not being able to transmit updates promptly. Our definition of staleness-related cost takes into account the rate at which requests are serviced with pages

that are out of date as well as the amount of time that the pages could be stale.

Using our definition of staleness-related cost we obtained analytic results describing how the available resources can best be used to transmit page updates to the proxies. Specifically, we defined a mechanism to transmit page updates to the proxies called "transmission attempts." We showed that the staleness-related cost for a page is minimised if transmission attempts for the page are made at regular intervals. Furthermore, we proved the important result that the cost for a page is minimised if it is cached at all of the proxies or at none of the proxies, depending on resource availability. This result is valid regardless of the values of the request arrival rate or page update rate.

We next focused on the case where the ratio of page update rate to transmission attempt rate for each page i (*i.e.*, $\frac{u_i}{n_i}$) is sufficiently large that $e^{-\frac{u_i}{n_i}}$ is negligible. We developed two heuristic algorithms that determine, for a given resource availability, which pages should be cached at the proxies, which pages should be retrieved directly from the web server, and the transmission attempt rate for each cached page such that the resulting staleness-related cost is close to optimal. By comparing with the optimal solution obtained by exhaustive search, we found that both algorithms yield accurate results. Heuristic algorithm II is more efficient in terms of computational requirements, so it should be the preferred algorithm.

Page fragments have been suggested as a means of improving system scalability.

We have extended our architecture to include fragments and used simulation to study the conditions under which the use of fragments is beneficial in terms of reducing staleness-related cost. These results provide valuable insights into the utility of fragments.

Taken together, the above contributions significantly advance our understanding of the resource management issues inherent in the delivery of dynamic information.

7.2 Future Work

Areas for future work include the following.

Page inter-update time distribution

In Section 4.2 our analytic results on the staleness of page i and the optimal delivery of page i are based on the assumption that the time between updates to this page is exponentially distributed. This assumption may not be valid for all classes of pages. For example, stock price information may be updated at regular intervals. It would be fruitful to explore the effects of pages with different inter-update time distributions and investigate how our analysis may be modified to provide optimal staleness-related cost for such pages.

Proxy capacity and response time

The details of the proxy servers have not been included in our investigation.

In general, each proxy server has finite processing resources which are used to service user requests and to capture page updates from the server. The portion of server capacity available for servicing user requests will have an impact on the system's response time. Furthermore, user requests that can be served from the cache have different resource requirements from those that are forwarded to the web server. A performance model should be developed to study the tradeoff between response time performance and the amount of resources required to process page updates.

Optimal page delivery strategies

In general, finding an optimal page delivery strategy involves solving the optimisation problem presented in Section 5.1. This is a very difficult task. We have described a two-step solution relying on approximation techniques that produces good results when $\frac{u_i}{n_i}$ is large enough for $e^{-\frac{u_i}{n_i}}$ to be negligible. Further work should be done to find efficient solutions that are applicable to situations where $e^{-\frac{u_i}{n_i}}$ is not negligible.

Average versus individual staleness-related cost

Our definition of staleness-related cost reflects the impact of staleness on the requests serviced by all of the proxies. Our optimisation problems are defined in this context. We have found that, from the perspective of a given proxy,

when the server's delivery strategy is tuned to serve the average requirements of all the proxies, the resulting cost is higher than that of a strategy designed to meet the requirements of that proxy alone [110]. These are preliminary results only. Further investigation is required to gain a full understanding of how one may construct delivery strategies that are tailored to individual proxies.

Different classes of users

Our optimisation of staleness-related cost is based on the assumption that there is only one class of requests (or users) served by all of the proxies. The issue of different classes of users has not been considered. In some cases, we may wish to give preferential treatment to one class of users and provide optimal page delivery to this class. This will have a negative impact on the other classes. Investigation into this topic should provide a good understanding of how the available resources can best be allocated to serve different classes of users.

Fragment layout guidelines

While we have shown the conditions under which page fragments can provide reduced staleness-related cost, we have made no suggestions about how one could automate page layout so as to best take advantage of this. For a

given set of pages that contain common data, methods for designing the page fragments should be explored.

Appendix A

Derivation of Equation 4.4

The optimisation problem is to minimise

$$q_i = \frac{b_i - y_i}{n_i T} \sum_{k=1}^{n_i T} \frac{e^{-u_i x_k} + u_i x_k - 1}{u_i} \tag{A.1}$$

subject to

$$\sum_{k=1}^{n_i T} x_k = T \tag{A.2}$$

Applying Equation (A.2) and after some manipulation, q_i can be rewritten as

$$q_i = \frac{b_i - y_i}{u_i n_i T} \left(\sum_{k=1}^{n_i T} e^{-u_i x_k} + u_i T - n_i T \right)$$

Ignoring the terms that are constant, minimising

$$f = \sum_{k=1}^{n_i T} e^{-u_i x_k} \tag{A.3}$$

will also minimise q_i.

Using the technique of Lagrangian multipliers we define the function h as follows:

$$h = \sum_{k=1}^{n_i T} e^{-u_i x_k} - \lambda \left(\sum_{k=1}^{n_i T} x_k - T \right)$$

We now minimise the unconstrained function h by setting its partial derivatives to zero, yielding

$$-u_i e^{-u_i x_k} - \lambda = 0 \qquad \forall k \tag{A.4}$$

and

$$T - \sum_{k=1}^{n_i T} x_k = 0$$

From these two equations, we obtain

$$\lambda = -u_i e^{-u_i / n_i} \tag{A.5}$$

Substituting Equation (A.5) into Equation (A.4) and simplifying, we get:

$$x_k = \frac{1}{n_i}$$

Finally, we show that the inflection point at $x_k = 1/n_i$ is a minimum for q_i [113]. Let $\vec{v}$ be a unit vector in $\mathbb{R}^{n_i T}$ and let

$$f = \vec{v} \cdot \nabla[\vec{v} \cdot \nabla q_i] \tag{A.6}$$

Substituting q_i from Equation (A.1) into Equation (A.6), after some manipulation we obtain

$$\begin{aligned} f &= \vec{v} \cdot \nabla \left[\frac{b_i - y_i}{u_i n_i T} \sum_{k=1}^{n_i T} v_k \left(-u_i e^{-u_i x_k} + u_i \right) \right] \\ &= \frac{b_i - y_i}{u_i n_i T} \sum_{k=1}^{n_i T} v_k^2 u_i^2 e^{-u_i x_k} \end{aligned}$$

Clearly $v_k^2 \geq 0$. Furthermore, at least one of $v_1, v_2, \ldots, v_{n_i T}$ must be nonzero because $|\vec{v}| = 1$. This means that there exists at least one value of k such that $v_k^2 > 0$. Note that $b_i - y_i > 0$ and because $u_i > 0$, $u_i^2 e^{-u_i x_k} > 0$. Therefore $f > 0$, q_i is convex, and $x_k = 1/n_i$ is a global minimum for q_i.

Appendix B

Summary of Notation

Symbol	Interpretation
A	The set of pages that are cached at the proxies
$\mathcal{A}'$	The set of fragments that are cached at the proxies
a_{ij}	The arrival rate of requests for page i at proxy j
B	The set of pages that are not cached at the proxies
b_i	$\sum_{j=1}^{M} a_{ij}$
C'	$\sum_{i=1}^{N} C'_i$
C_i	The staleness-related cost for page i assuming regular transmission attempts
C'_i	The staleness-related cost for assembled page i
C_A	$\sum_{i \in A} C_i$
F	The number of frequently-updated fragments stored at the server
G_i	The set of proxies that cache page i
J	The mean number of fragments per page
K	$\sum_{i \in A} n_i$
M	The number of proxies in the system
m	The mean number of pages that use each fragment
N	The number of frequently-updated pages stored at the server
n_i	The transmission attempt rate for page i
u_i	The update rate for page i
P	The resources required to transmit a page update to the proxies

Symbol	Interpretation
R	The server capacity in resource units per second
R_i	The capacity devoted to page i
y_i	$b_i - \sum_{j \in G_i} a_{ij}$
β_f	$\sum_{i \in \pi(f)} b_i$
γ	The relative resources required to transmit a fragment
$\pi(f)$	The pages that contain fragment f
$\phi(i)$	The fragments used to construct page i
υ_f	The update rate for fragment f
ω_f	The transmission attempt rate for fragment f

Bibliography

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – HTTP/1.1." RFC 2616, 1999.

[2] D. Wessels, "Intelligent caching for world wide web objects," in *Proceedings of INET'95*, 1995.

[3] J. S. Gwertzman and M. Seltzer, "The case for geographical push-caching," in *Proceedings of the 1995 Workshop on Hot Operating Systems*, 1995.

[4] H. Yu, L. Breslau, and S. Shenker, "A scalable web cache consistency architecture," in *Proceedings of ACM SIGCOMM*, pp. 163–174, September 1999.

[5] J. S. Gwertzman and M. Seltzer, "World wide web cache consistency," in *Proceedings of the 1996 COMPCON*, February 1996.

[6] R. Tewari, T. Niranjan, and S. Ramamurthy, "WCDP: A protocol for web

cache consistency," in *Proceedings of the 7th International Web Content Caching and Distribution Workshop*, 2002.

[7] M. H. Ammar, K. C. Almeroth, R. J. Clark, and Z. Fei, "Multicast delivery of web pages or how to make web servers pushy," in *Proceedings of the Workshop on Internet Server Performance, Madison, Wisconsin*, June 1998.

[8] P. Rodriguez, K. W. Ross, and E. W. Biersack, "Improving the WWW: Caching or multicast?," in *Proceedings of the 3rd International Web Content Caching and Distribution Workshop*, June 1998.

[9] R. Clark and M. H. Ammar, "Providing scalable web service using multicast delivery," *Computer Networks and ISDN Systems*, no. 29, pp. 841–858, 1997.

[10] J. Nonnenmacher and E. W. Biersack, "Asynchronous multicast push: AMP," in *Proceedings of the International Conference on Computer Communications*, pp. 419–430, November 1997.

[11] J. R. Challenger, P. Dantzig, A. Iyengar, mark S. Squillante, and L. Zhang, "Efficiently serving dynamic data at highly accessed web sites," *IEEE/ACM Transactions on Networking*, vol. 12, pp. 233–246, April 2004.

[12] G. v. Bochmann, J. W. Wong, D. Evans, T. C. Lau, D. Bourne, B. Kerhervé,

M.-V. M. Salem, and H. Ye, "Scalability of web-based electronic commerce systems," *IEEE Communications Magazine*, vol. 41, pp. 110–115, July 2003.

[13] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed, "A publishing system for efficiently creating dynamic web content," in *Proceedings of IEEE INFOCOM*, pp. 844–853, 2000.

[14] C. Yuan, Y. Chen, and Z. Zhang, "Evaluation of edge caching/offloading for dynamic content delivery," in *Proceedings of the 12th International World Wide Web Conference*, pp. 461–471, May 2003.

[15] W. Shi, R. Wright, E. Collins, and V. Karamcheti, "Workload characterization of a personalised web site—and its implications for dynamic content caching," in *Proceedings of the 7th International Web Content Caching and Distribution Workshop*, 2002.

[16] J. Cho and H. Garcia-Molina, "Synchronizing a database to improve freshness," in *Proceedings of ACM SIGMOD*, pp. 117–128, 2000.

[17] J. Cho and H. Garcia-Molina, "Estimating frequency of change," *ACM Transactions on Internet Technology*, vol. 3, pp. 256–290, August 2003.

[18] A. Dingle and T. Partl, "Web cache coherence," in *Proceedings of the 5th International World Wide Web Conference*, May 1996.

[19] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen, "Optimal crawling strategies for web search engines," in *Proceedings of the 11th International World Wide Web Conference*, pp. 136–147, 2002.

[20] E. G. Coffman, Jr., Z. Liu, and R. R. Weber, "Optimal robot scheduling for web search engines," *Journal of Scheduling*, vol. 1, pp. 15–29, June 1998.

[21] J. Edwards, K. McCurley, and J. Tomlin, "An adaptive model for optimizing performance of an incremental web crawler," in *Proceedings of the 10th Annual World Wide Web Conference*, pp. 106–113, May 2001.

[22] J. Gecsei, *The Architecture of Videotex Systems*. Prentice-Hall, 1983.

[23] M. H. Ammar, *Performance Analysis of Information Systems Using Broadcast Delivery*. PhD thesis, University of Waterloo, 1985.

[24] M. Ammar and J. W. Wong, "The design of teletext broadcast cycles," *Performance Evaluation*, vol. 5, pp. 235–242, December 1985.

[25] M. Ammar and J. W. Wong, "On the optimality of cyclic transmission in teletext systems," *IEEE Transactions on Communications*, vol. 35, pp. 68–73, January 1987.

[26] C.-J. Su and L. Tassiulas, "Broadcast scheduling for information distribution," in *Proceedings of IEEE INFOCOM*, 1997.

[27] L. Tassiulas and J. S. Chi, "Optimal memory management strategies for a mobile user in a broadcast data delivery system," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 1226–1238, September 1997.

[28] A. Bar-Noy, V. Dreizin, and B. Patt-Shamir, "Efficient periodic scheduling by trees," in *Proceedings of IEEE INFOCOM*, pp. 791–800, 2002.

[29] G. Herman, G. Gopal, K. C. Lee, and A. Weinrib, "The Datacycle architecture for very high throughput database systems," in *Proceedings of ACM SIGMOD*, pp. 97–103, May 1987.

[30] T. F. Bowen, G. Gopal, G. Herman, T. Hickey, K. C. Lee, W. H. Mansfield, J. Raitz, and A. Weinrib, "The Datacycle architecture," *Communications of the ACM*, pp. 71–81, 1992.

[31] J. W. Wong and M. H. Ammar, "Response time performance of videotex systems," *IEEE Journal on Selected Areas in Communications*, pp. 1174–1180, October 1986.

[32] J. W. Wong and H. D. Dykeman, "Architecture and performance of large scale information delivery networks," in *Proceedings of the International Teletraffic Congress*, pp. 4.4B.4.1–4.4B.4.7, 1988.

[33] J. W. Wong, "Broadcast delivery," *Proceedings of the IEEE*, vol. 76, pp. 1566–1577, December 1988.

[34] J. W. Wong and M. H. Ammar, "Analysis of broadcast delivery in a videotex system," *IEEE Transactions on Computers*, vol. C–34, pp. 863–866, September 1985.

[35] H. D. Dykeman, M. H. Ammar, and J. W. Wong, "Scheduling algorithms for videotex systems under broadcast delivery," in *Proceedings of the International Conference on Communications*, pp. 1847–1851, June 1986.

[36] D. K. Gifford, R. W. Baldwin, S. T. Berlin, and J. M. Lucassen, "An architecture for large scale information systems," in *Proceedings of the 10th ACM Symposium on Operating System Principles*, pp. 161–170, 1985.

[37] T. Imielinski and B. R. Badrinath, "Mobile wireless computing: Solutions and challenges in data management," *Communications of the ACM*, vol. 37, pp. 28–28, October 1994.

[38] B. R. Badrinath, A. Acharya, and T. Imielinski, "Impact of mobility on distributed computations," *ACM Operating Systems Review*, vol. 27, pp. 15–20, April 1993.

[39] T. Imielinski, S. Viswanathan, and B. R. Badrinath, "Energy efficient indexing on air," in *Proceedings of ACM SIGMOD*, pp. 25–36, 1994.

[40] M. H. Ammar, "Response time in a teletext system: An individual user's perspective," *IEEE Transactions on Communications*, vol. 35, no. 11, pp. 1159–1170, 1987.

[41] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data management for asymmetric communication environments," in *Proceedings of ACM SIGMOD*, pp. 199–210, 1995.

[42] S. Zdonik and M. Franklin, "Are "disks in the air" just pie in the sky?," in *Proceedings of the Workshop on Mobile Computing Systems and Applications*, 1994.

[43] M. Franklin and S. Zdonik, "Dissemination-based information systems," *IEEE Data Engineering Bulletin*, vol. 19, September 1996.

[44] S. Acharya, M. Franklin, and S. Zdonik, "Prefetching from a broadcast disk," in *Proceedings of the International Conference on Data Engineering*, February 1996.

[45] S. Acharya, M. Franklin, and S. Zdonik, "Dissemination-based data delivery

using broadcast disks," *IEEE Personal Communications*, vol. 2, pp. 50–60, December 1995.

[46] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast disks: Data management for asymmetric communication environments," Tech. Rep. CS-94-43, Brown University, 1994.

[47] A. Bar-Noy, B. Patt-Shamir, and I. Ziper, "Broadcast disks with polynomial cost functions," in *Proceedings of IEEE INFOCOM*, pp. 575–584, 2000.

[48] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann, "World-Wide Web: The information universe," *Electronic Networking: Research, Applications and Policy*, vol. 1, no. 2, pp. 52–58, 1992.

[49] J. Postel, "Internet protocol." RFC 791, 1981.

[50] S. E. Deering, "Host extensions for IP multicasting." RFC 1112, 1989.

[51] W. Fenner, "Internet group management protocol, version 2." RFC 2236, 1997.

[52] V. Liberatore, "Multicast scheduling for list requests," in *Proceedings of IEEE INFOCOM*, pp. 1129–1137, 2002.

[53] K. Stathatos, N. Roussopoulos, and J. S. Baras, "Adaptive data broadcast

in hybrid networks," in *Proceedings of the 23rd Annual Conference on Very Large Data Bases*, pp. 326–335, 1997.

[54] P. R. Rodriguez and E. W. Biersack, "Continuous multicast push of web documents over the internet," *IEEE Network*, vol. 12, pp. 18–31, March–April 1998.

[55] J. C.-I. Chuang and M. A. Sirbu, "Pricing multicast communication: A cost-based approach," in *Proceedings of the INET'98 Conference*, 1998.

[56] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *Proceedings of ACM SIGCOMM*, pp. 117–130, August 1996.

[57] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in *Proceedings of ACM SIGCOMM*, pp. 342–356, August 1995.

[58] D. DeLucia and K. Obraczka, "Multicast feedback suppression using representatives," in *Proceedings of IEEE INFOCOM*, 1997.

[59] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya, "Reliable multicast transport protocol (RMTP)," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 407–421, April 1997.

[60] K. L. Calvert, J. Griffioen, A. Sehgal, and S. Wen, "Concast: Design and implementation of a new network service," in *Proceedings of the International Conference on Network Protocols*, pp. 335–344, November 1999.

[61] J.-C. Bolot, T. Turletti, and I. Wakeman, "Scalable feedback control for multicast video distribution on the internet," in *Proceedings of ACM SIGCOMM*, pp. 58–67, September 1994.

[62] M. Grossglauser, "Optimal deterministic timeouts for reliable scalable multicast," in *Proceedings of IEEE INFOCOM*, 1996.

[63] M. Grossglauser, "Optimal deterministic timeouts for reliable scalable multicast," *IEEE Journal On Selected Areas In Communications*, vol. 15, pp. 422–433, April 1997.

[64] R. Yavatkar, J. Griffioen, and M. Sudan, "A reliable dissemination protocol for interactive collaborative applications," in *Proceedings of ACM Multimedia*, pp. 333–344, 1995.

[65] Y. Birk and D. Crupnicoff, "A multicast transmission schedule for scalable multi-rate distribution of bulk data using non-scalable erasure-correcting codes," in *Proceedings of IEEE INFOCOM*, pp. 1033–1043, 2003.

[66] K. C. Almeroth, M. H. Ammar, and Z. Fei, "Scalable delivery of web pages

using cyclic best-effort (UDP) multicast," in *Proceedings of IEEE INFOCOM*, March 1998.

[67] L. Rizzo and L. Vicisano, "A reliable multicast data distribution protocol based on software FEC techniques," in *Proceedings of the HPCS'97 Workshop*, June 1997.

[68] J. S. Gwertzman and M. Seltzer, "An analysis of geographical push-caching," in *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems*, 1997.

[69] A. Chankhunthod, P. B. Danzip, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," Tech. Rep. CU-CS-766-95, Department of Computer Science, University of Colorado – Boulder, 1995.

[70] R. Cohen, L. Katzir, and D. Raz, "Scheduling algorithms for a cache pre-filling content distribution network," in *Proceedings of IEEE INFOCOM*, pp. 940–949, 2002.

[71] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," Tech. Rep. 1371, Computer Sciences Department, University of Wisconsin-Madison, 1998.

[72] G. K. Zipf, *Human behavior and the principle of least effort; an introduction to human ecology.* Addison-Wesley, 1949.

[73] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing reference locality in the WWW," Tech. Rep. TR-96-11, Department of Computer Science, Boston University, 1996.

[74] I. Marshall and C. Roadknight, "Linking cache performance to user behaviour," in *Proceedings of the 3rd International Web Caching Workshop*, June 1998.

[75] F. Douglis, A. Feldmann, and B. Krishnamurthy, "Rate of change and other metrics: A live study of the world wide web," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 147–158, December 1997.

[76] J. Cho and H. Garcia-Molina, "The evolution of the web and implications for an incremental crawler," in *Proceedings of the 26th Annual Conference on Very Large Data Bases*, pp. 200–209, September 2000.

[77] S. Deng, "Empirical model of WWW document arrivals at access link," in *Proceedings of IEEE ICC*, 1996.

[78] M. E. Crovella and A. Bestavros, "Explaining world wide web traffic self-

similarity," Tech. Rep. TR-95-015, Computer Science Department, Boston University, 1995.

[79] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic evidence and possible causes," in *Proceedings of ACM SIGMETRICS*, pp. 160–169, May 1996.

[80] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in web client access patterns: Characteristics and caching implications," Tech. Rep. BUCS-TR-1998-023, Computer Science Department, Boston University, 1998.

[81] S. U. Khaunte and J. O. Limb, "Statistical characterization of a world wide web browsing session," Tech. Rep. GIT-CC-97-17, College of Computing, Georgia Institute of Technology, 1997.

[82] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," Tech. Rep. BU-CS-97-006, Computer Science Department, Boston University, 1998.

[83] J. Almeida and P. Cao, "Measuring proxy performance with the wisconsin proxy benchmark," in *Proceedings of the 3rd International Web Caching Workshop*, June 1998.

[84] B. D. Davison, "Simultaneous proxy evaluation," in *Proceedings of the 4th International Web Caching Workshop*, March 1999.

[85] Transaction Processing Performance Council, "TPC Benchmark^{TM} W," 2001.

[86] J. Jung, A. W. Berger, and H. Balakrishnan, "Modelling TTL-based internet caches," in *Proceedings of IEEE INFOCOM*, pp. 417–426, 2003.

[87] E. Cohen and H. Kaplan, "Refreshment policies for web content caches," in *Proceedings of IEEE INFOCOM*, pp. 1398–1406, 2001.

[88] E. Cohen and H. Kaplan, "Ageing through cascaded caches; performance issues in the distribution of web content," in *Proceedings of ACM SIGCOMM*, pp. 41–53, 2001.

[89] Z. Fei, "A novel approach to managing consistency in content distribution networks," in *Proceedings of the 6th International Web Content Caching and Distribution Workshop*, 2001.

[90] M. Reddy and G. P. Pletcher, "Intelligent web caching using document life histories: A comparison with existing cache management techniques," in *Proceedings of the Third International Web Caching Workshop*, June 1998.

[91] B. Krishnamurthy and C. E. Wills, "Study of piggyback cache validation for proxy caches in the world wide web," in *Proceedings of the USENIX Symposium on Internet Technology and Systems*, December 1997.

[92] B. Krishnamurthy and C. E. Wills, "Proxy cache coherency and replacement—towards a more complete picture," in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, June 1999.

[93] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar, "Engineering server-driven consistency for large scale dynamic web services," in *Proceedings of the 10th Annual World Wide Web Conference*, pp. 45–57, May 2001.

[94] D. Li and D. R. Cheriton, "Scalable web caching of frequently updated objects using reliable multicast," in *Proceedings of the USENIX Symposium on Internet Technology and Systems*, October 1999.

[95] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive push-pull: Disseminating dynamic web data," in *Proceedings of the 10th International World Wide Web Conference*, pp. 265–274, May 2001.

[96] M. Mikhailov and C. E. Wills, "Evaluating a new approach to strong web

cache consistency with snapshots of collected content," in *Proceedings of the 12th International World Wide Web Conference*, pp. 599–608, May 2003.

[97] V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive leases: A strong consistency mechanism for the world wide web," in *Proceedings of IEEE INFOCOM*, pp. 834–843, 2000.

[98] A. Ninan, P. Kulkami, P. Shenoy, K. Ramamritham, and R. Tewari, "Cooperative leases: Scalable consistency maintenance in content distribution networks," in *Proceedings of the 11th International World Wide Web Conference*, pp. 1–12, May 2002.

[99] J. W. Wong, D. Evans, and A. K. Kock, "Caching and multicast delivery," in *Electronic Commerce Technology Trends: Challenges and Opportunities*, pp. 29–40, IBM Press, 2000.

[100] N. Yu and A. Vahdat, "Design and evaluation of a continuous consistency model for replicated services," in *Proceedings of Operating Systems Design and Implementation*, October 2000.

[101] A. Labrinidis and N. Roussopoulos, "Webview materialization," in *Proceedings of ACM SIGMOD*, pp. 367–378, May 2000.

[102] A. Labrinidis and N. Roussopoulos, "Update propagation strategies for im-

proving the quality of data on the web," in *Proceedings of the 27th Annual Conference on Very Large Data Bases*, pp. 391–400, September 2001.

[103] B. E. Brewington and G. Cybenko, "How dynamic is the web?," in *Proceedings of the 9th International World Wide Web Conference*, May 2000.

[104] B. Liu, G. Abdulla, T. Johnson, and E. A. Fox, "Web response time and proxy caching," in *Proceedings of WebNet98*, November 1998.

[105] L. D. Catledge and J. E. Pitkow, "Characterizing browsing strategies in the world-wide web," *Computer Networks and ISDN Systems*, vol. 27, pp. 1065–1073, April 1995.

[106] B. M. Duska, D. Marwood, and M. J. Feeley, "The measured access characteristics of world wide web client proxy caches," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[107] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 193–206, December 1997.

[108] J. Shim, P. Scheuermann, and R. Vingralek, "Proxy cache design: Algorithms, implementation and performance," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, pp. 549–562, July–August 1999.

[109] O. Bahat and A. M. Makowski, "Optimal replacement policies for non-uniform cache objects with optional eviction," in *Proceedings of IEEE INFOCOM*, pp. 427–437, 2003.

[110] J. W. Wong, D. Evans, and M. Kwok, "On staleness and the delivery of web pages," *Information Systems Frontiers: A Journal of Research and Innovation*, vol. 5, pp. 129–136, April 2003.

[111] T. kelly and J. Mogul, "Aliasing on the world wide web: Prevalence and performance implications," in *Proceedings of the 11th International World Wide Web Conference*, pp. 281–292, May 2002.

[112] R. Jain, *The Art of Computer Systems Performance Analysis*. New York: Jon Wiley & Sons, 1991.

[113] W. Kaplan, *Advanced Calculus*. Reading: Addison-Wesley, 1952.