Web workloads influencing disconnected service access

by

Bharat Baddepudi Chandra, B.Tech.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF ARTS

The University of Texas at Austin

May 2001

Web workloads influencing disconnected service access

Approved by Supervising Committee:

Dedicated to my parents

Acknowledgments

I would like to thank Dr. Michael Dahlin, my thesis advisor, for his guidance and encouragement throughout the past two years. This thesis would not have been possible without his support and patience. His valuable suggestions and inputs in the discussion of the key ideas of this thesis and my other publications were extremely useful for my understanding of the subject thoroughly.

I also extend my thanks to my colleagues Amol Nayate and Lei Gao for bearing with me for the past two years and lending me a helping hand in making progress in my research work. I would also like to thank Ravi Kokku and Praveen Yalagandula for all the technical trivia.

Last, but not the least, I would like to thank my family and friends for their limitless love, affection and encouragement through all these years.

BHARAT BADDEPUDI CHANDRA

The University of Texas at Austin
May 2001

Web workloads influencing disconnected service access

Bharat Baddepudi Chandra, M.A.
The University of Texas at Austin, 2001

Supervisor: Michael D. Dahlin

Disconnected operation, in which a client accesses a service without relying on network connectivity, is crucial for improving availability, supporting mobility, and providing responsive performance. Because many web services are not cachable, disconnected access to web services may require mobile service code to execute in client caches. Furthermore as web workloads access large amounts of data, disconnected access must require prefetching data that will later be used on demand. Unfortunately, this can significantly increase the total amount of data fetched by a service. In this thesis we present an argument that aggressive prefetching is feasible. A study of the web workload characteristics at typical clients suggests a need for a flexible, automated resource management system to prevent denial of service attacks by these potentially unreliable mobile service codes that prefetch at the clients. We therefore present and evaluate a *Popularity based* resource management policy for such an environment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Traditional WAN caches provide limited hit rates and efforts to improve these rates have met with limited success. Many HTTP requests are essentially arbitrary RPCs(Remote Procedure Calls) that are not cachable using traditional means. Wolman et al. [1] find that uncachable web accesses reduce the upper bound on cache hit rates by about a factor of two. The increasing use of mobile devices further pressures traditional caching strategies. Mobile devices function in the disconnected mode, which allows clients to access web services without relying on the network connection between the client and the origin server. Supporting disconnected operation is a key problem for improving web services for three reasons.

1. Disconnected operation allows mobile clients to access services when their network connection is unavailable, expensive, or slow.
2. Disconnected operation can improve service availability. Studies consistently find that, in contrast with targets of "four nines" or "five nines" of availability (99.99% uptime or 99.999% uptime) for important services, the Internet network layer provides only about two nines of host-to-host

connection availability [2, 3, 4]. The resulting average of about 14 minutes per day of unavailability to a typical client hinders commercial sites as well as mission-critical sites.

3. Disconnected operation can significantly improve performance. Traditional web caching is a simple example of this strategy, and several studies have demonstrated even more dramatic speedups when service code is shipped to clients and proxies [5, 7].

1.1 Challenges

A combination of mobile service codes and prefetching mechanisms can support disconnected operations. In the context of disconnected operation, Chandra et al. [2] find that if an infrastructure supports mobile service code for disconnected operation, it can reduce average service unavailability by factors of 2.7 to 15.4, but if an infrastructure only supports caching and prefetching, average improvements are limited to 1.8 to 6.2. In order to provide better services, these services should have fair access to resources such as disk space, network bandwidth, CPU cycles and memory space at the client. In addition, multiple mobile services should coexist in a resource constrained environment without allowing denial of service attacks at the client. Current mobile service frameworks (e.g., Applets, Javascript) have limited flexibility in that they prevent access to disk by untrusted code (limiting support for disconnected operations) but provide no limits on CPU cycles or memory consumed (making the system vulnerable to denial of service attacks). Although several experimental systems have provided low level mechanisms for limiting resources consumed by untrusted Java code [8, 9], properties of web service workloads make it challenging to develop a scalable infrastructure for disconnected access. First, clients may access a large number of services, meaning that many untrusted services will compete for resources. Second, the resources available at client devices may vary widely. Third, prefetching and hoarding – key techniques for coping with disconnected operation – can dramatically increase the resource demands of applications: Ironically, providing mobile code with the ability to access disk to support disconnected operation worsens the resource management challenge because it gives applications an incentive to use more resources.

This work examines the impact of web workloads on constructing a scalable infrastructure to support disconnected access to web services. We focus on environments that allow web services to ship service code to caches and proxies and that allow this code to use prefetching, hoarding [12, 13, 14], write buffering, persistent message queues [15, 16], and application-specific adaptation [17] to mask disconnections by satisfying requests locally. Several researchers have proposed such systems [16, 5, 7]. In this thesis we propose that *aggressive* prefetching is feasible. We will quantify the costs and demands of this aggressive prefetching and analyze the framework needed to support it. The main challenge in building the framework is that it should scale to a diverse set of services used by a large user population. This paper focuses mainly on the prefetching issues and the client side support needed to maintain disconnected operations. The prefetching issues can be categorized broadly into server related, network related and client related issues. Each of these issues relates to the additional overhead that can potentially be experienced at these locations. These overheads can be reduced with improved scheduling algorithms at both the client and server and by using restrained versions of prefetching to reduce the network overhead [31, 33, 35]

1.2 Contributions

The key result of this work is quantifying the tradeoff between the costs of prefetching and the performance improvement perceived at the end user. A sensitivity analysis of this result provides a better understanding of the result by varying assumptions on the changes in cost, technology, and the end user devices. This work also makes two other contributions. First, we quantify the resource requirements of disconnected services by examining both the general requirements across services and the requirements of several case-study services. These data suggest that supporting disconnected operation for a large number of services is feasible. In particular, we argue that prefetching an order of magnitude more data than is used on demand may often be reasonable. However, we also find that careful resource management by the system and application-specific adaptation by the services is needed. Second, we develop a resource management framework that (i) provides *efficient allocation* across services to give important services more resources than less important ones, (ii) provides *performance isolation* so that aggressive services do not interfere with passive ones, and (iii) makes allocation automatic *self-tuning* decisions without relying on user input or directions from untrusted services.

Organization

The rest of this report is organized as follows. Chapter 2 describes and analyses a quantitative measure for the tradeoff between resources consumed and performance gained with prefetching. Chapter 3 studies the web workload characteristics at the client. Chapter 4 describes a simple resource management policy for clients. Chapter 5 looks into some related work in these areas and Chapter 6 summarizes our results and conclusions, and finally look into potential future work in this area.

Chapter 2

End-to-end costs and benefits of prefetching

This chapter evaluates a methodology for quantifying the cost and benefits of prefetching from the end user point of view. The web content is not only becoming more dynamic (e.g., CGI scripts, .asp) but also changing very rapidly. This is the reason why traditional caching mechanisms are not able to provide increased benefits to the client. Speculative prefetching can, to some extent, improve the performance delivered to the end user. By hoarding a subset of the pages the user is most likely to view, prefetching can improve both the hit rate and the latency experienced by the client. There are numerous push-based or pull-based algorithms that can be employed by the service or the client [18, 21].

The current internet infrastructure also suffers an average 1% downtime (or unavailability of about 14min/day) on a typical path from a client to a server [3]. To mask failures that cause this downtime and improve the availability to 3-9's or even 4-9's, active objects [16, 5, 7] can be deployed over the unreliable internet architecture. A mixture of active object technology and hoarding techniques such as prefetching can potentially improve the client perceived latency and hit-rates even in the case of failures [2]. Disconnected mode of operation can be viewed as a generalized case of failure, where the client and server cannot communicate over a reliable channel.

Some current systems provide the option of hoarding for disconnected operations. For example, systems such as AvantGo [44] for palm-size computers or Microsoft Internet Explorer [45] can be set to prefetch a (subset of a) particular site that the user might want to browse offline. Even in the traditional browser context, the CNN home page (http://www.cnn.com/index.html) has the feature of being reloaded every thirty minutes, even though the user might not have asked for the updated version. This feature ensures that the user has the latest version of the page in case she gets disconnected from the internet, thus increasing the hit rate.

Along with the potential benefits, prefetching also involves costs. These primarily include the cost of additional resource usage which can be in the form of network bandwidth, disk space, CPU cycles or memory consumed at the end user. Hence there is a need to trade off improved performance (measured in terms of the latency or availability or freshness) with the additional resources consumed in the process. This scenario can be viewed as in figure 2.1.

Note that in this report we do not provide a method to achieve 3-9's or 4-9's of availability nor do we propose the best prefetching algorithm. We just seek to understand the extent of the parameters involved in judging the amount of prefetching that could be justified.

As can be seen, the metrics involved in the trade-off are not directly

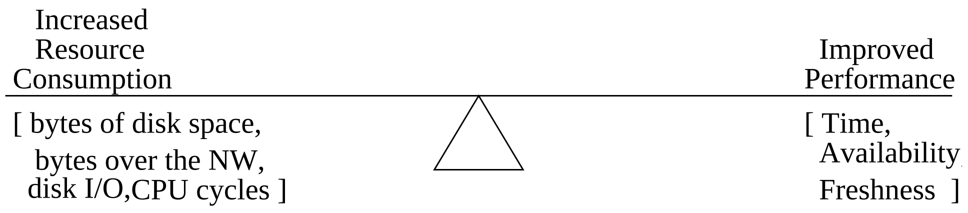


Figure 2.1: Trade-off between resources and response time.

Resource	Cost (dollars/10KB)
$NWCost_{LAN}$	0.0001
$NWCost_{Modem}$	0.0002
$StorageCost$	$8*10^{-6}$/month
$WaitCost$	0.02

Table 2.1: Cost Chart. Source Gray and Shenoy [32].

comparable. Gray and Shenoy [32] suggest a methodology for converting all the units of measurement into monetary values to estimate whether caching a data object is economically justified by comparing the cost of storage against the cost of network bandwidth and human waiting time. We extend that method to balance prefetch costs and benefits.

First, we summarize Gray and Shenoy's calculations for disk costs, network cost, and human waiting time benefit for caching an average web object. These results are shown in the Figure 2.1.

Based on the above measures we can estimate that downloading a 10KB object across an Internet/LAN network costs about $NWCost_{LAN} = \$.0001$, across a Modem costs about $NWCost_{Modem} = \$.0002$, across a wireless modem about $NWCost_{Wireless} = \$.01$, and Gray and Shenoy estimate that storing a 10KB object costs about $StorageCost = \$8 \cdot 10^{-6}/\text{month}$. Assuming that human time is worth about \$20/hour, the "waiting cost" due to cache miss latency is about $WaitCost_{LAN}$ =\$.02 when connected via an LAN/Internet, $WaitCost_{Modem}$ =\$.03 when connected via a modem, and $WaitCost_{Wireless} =$ \$.07 when connected via a wireless modem. Notice that disk space is "rental" of a byte per second while network is just cost per byte. (Once you use a bit of network bandwidth, you can never use that bit again; but when you stop using disk space, you can use it for something else.) We could also have computed cost of replicating an object through several versions; in that case the network transmission time would be amortized by version lifetime rather than (as in this case) the disk space being consumed for version lifetime. Based on these estimates, Gray and Shenoy conclude, for example, that once fetched, an object should be kept in cache even if the expected time to reaccess it is on the order of decades.

This methodology can be extended to estimate whether prefetching an object is justified. Suppose that P_{used} is the probability that a prefetched object is used before it is updated or evicted from the cache and that unreferenced prefetched objects are evicted after one month. Then the threshold probability P_T can be defined as the lowest P_{used} for an object that justifies prefetching it. Thus an object is prefetched only if its P_{used} is greater than the P_T.

$$P_T = \frac{NWCost_{prefetchNW} + StorageCost}{WaitCost_{demandNW} + NWCost_{demandNW}} \tag{2.1}$$

P_T can be viewed as the ratio of cost incurred due to prefetching to the cost incurred with traditional on-demand fetching. indicates how useful prefetching an object can be. In other words, if there is P_T probability of

	Network used for demand fetch		
Prefetch NW $\Downarrow$	LAN/Internet	Modem	Wireless
LAN/Internet	0.0054	0.0036	0.0014
Modem		0.0069	0.0026
Wireless			0.1251

Table 2.2: Estimate of the P_T threshold probability that a prefetched object is used that justifies prefetching it to reduce human waiting time to demand fetch it. Source [32]

prefetching an object, then it is economically justified to prefetch it. Note that amount of additional resources consumed is limited to a factor of at most $\frac{1}{P_T}$ compared to an on-demand system.

Table 2.2 summarizes our estimates of P_T. The rows correspond to different networks used for prefetching ($prefetchNW$), and the columns to different networks used for demand fetch (*demandNW*). The diagonal corresponds to prefetching on the same network as the later demand fetch. For a LAN/Internet environment, for example, it can make sense to prefetch an object if the user has a 0.5% chance of accessing it over the next month and before it expires.

This economic argument for aggressive prefetching is particularly attractive for supporting heterogeneous networks or disconnected operation. As the table indicates, prefetching when network bandwidth is cheap (e.g., on a LAN connected to the Internet) to avoid network traffic when it is expensive (e.g., on a wireless modem or when congested) can be advantageous even if there is a 1/200 chance that an object will be used. If, for example, network bandwidth per byte is 100x more expensive across a wireless PCS modem than across a DSL connection [1], then it makes sense to prefetch an object if there

[1]For example, suppose PCS provides about 1KB/s of useful bandwidth at \$0.10/minute for a cost of about \$1 per 600KB. And suppose that a \$50/month DSL connection provides

is a 1% chance that it will be used while mobile, even neglecting the benefits to human waiting time.

Assumptions and Limitations

In the case of disconnected operation, the $WaitCost$ term would be replaced by a (typically higher) *DenialOfService* term that represents the cost of not delivering the service to the user at all.

One factor ignored in these calculations is server costs. One concern about aggressive prefetching is that it could swamp servers. Our system controls this by putting prefetching under the control of programs supplied by servers. This allows servers to avoid prefetching when it would interfere with demand traffic. More generally, the above methodology can be extended to include all server processing costs.

Another argument against aggressive prefetching is that it may overwhelm the Internet infrastructure. Certainly, if everyone started aggressively prefetching tomorrow, capacity would be stretched. One way of viewing this calculation is that it suggests that the economic incentives to grow network capacity over time to accommodate prefetching may exist. Odlyzko [46], for example, suggests that in rapidly-growing communications systems, capacity and demand increases seldom occur at more than 100% per year, partially due to the time needed to diffuse new applications with new demands.

Another factor ignored in these calculations is the presence of heterogeneous devices. For example, a palmtop machine may have significantly less

100KB/s of useful bandwidth but is only used 1% of the time for a cost of about $1 per 50MB. Given the "use it or lose it" nature of DSL bandwidth pricing compared to the per-minute pricing of most wireless services, this estimate is conservative.

storage space than a desktop machine. This can be modeled by increasing the $StorageCost$ term [32].

Technology trends may favor more aggressive replication in the future. First, network and disk costs appear likely to fall much more quickly than human waiting time costs. Second, the deployment of commercial content-distribution networks (CDNs) may significantly reduce the network costs of prefetching by allowing clients to prefetch from a nearby CDN node rather than the origin server.

Overall, the raw capacity of disks and networks as well as the back of the envelope economic calculations all suggest that in many current environments, it may be reasonable for services to use significant amounts of spare capacity to support disconnected operation. We conclude that for web service workloads, capacity to support prefetching of ten times more data than is used on demand may often be available, though support to prefetch 100 times more data than is used may be excessive for today's configurations. Given this economic view we can interpret that users might increase the usage of prefetching in order to decrease their wait time. A caveat in this scenario is that it might not be feasible to support aggressive prefetching in the very near future due to the limitations of the network infrastructure.

Below we discuss the implications and sensitivity of the above result to the assumptions made above. The sensitivity to these results can be evaluated for assumptions on:

- Cost
- Technology
- Device heterogeneity
- Prioritized traffic

2.1 Cost assumptions

The cost values used for the network, server, client and human waiting time were assumed to be constants in the above analysis. These may however change with varying conditions and hence affect the value of the prefetch threshold. We analyze each of these factors in turn.

2.1.1 Network

In this section we present an economic view which justifies increased prefetching with the current trend of decreasing network costs. In the above prefetch threshold calculations we assumed that the cost of transmitting over a network is constant for that network. We can calculate an approximate value for this as shown below:

For a modem connection (costing \$20/month at 5 KB/s) with 10% utilization, the network transmission cost is

$$\frac{\$20/\text{month}}{30\text{days/month} * 86400\text{sec/day} * 5 * 10^3\text{bytes/sec} * 0.1} \approx 2 * 10^{-8}\$/\text{byte}.$$

For DSL connection (\$50/month; 100 KB/s) similar network cost is 10^{-9}\$/byte. Wireless access via cell phone (\$50/2000 minutes; 64kpbs) costs around $5 * 10^{-7}$\$/byte. Hence we will initially estimate the cost of network transmission is of the order of 10^{-8} \$/byte. We will examine the sensitivity of our result to these assumptions below.

Table 2.3 generated in 1998 indicates the cost of transmitting over the typical networks in that year. Table 2.4 shows the same monthly costs for providing some typical network connections in the current year. It can be

Network	Transmission Cost (dollars/MB)
Modem	0.25-0.50
Private line	0.5-1
T1 Frame Relay	0.3
Internet	0.04-0.15

Table 2.3: Costs of transmitting a megabyte of data over various networks(in 1998). Source Odlyzko [47].

Network	Speeds (Mbps)	Monthly Price ($)	Transmission Cost ($/MB)
ISDN	0.128	$395.00	0.285
T1	1.54	$900.00	0.054
Ethernet	10	$5000.00	0.046
T3	45	$51000.00	0.104

Table 2.4: The connection costs for various networks (in 2001). Source Netdoor.com [48].

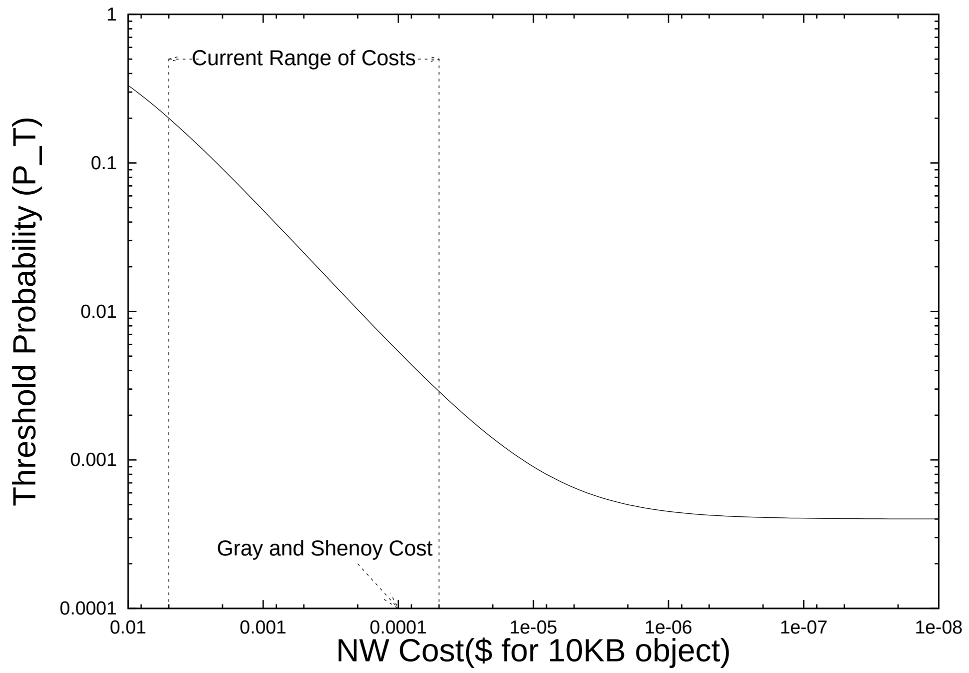


Figure 2.2: Decrease in prefetch threshold as network cost is decreased

observed that either

- Most of the network costs have become cheaper (e.g., T1 network) or
- The costs have remained relatively same (e.g., ISDN) or
- Some new relatively cheaper networks have emerged (e.g., T3)

Figure 2.2 shows the variation in the prefetch threshold P_T with decreasing network costs (assuming constant values for $StorageCost$ and $WaitTime$). It also gives a range of cost values for current networks. It can be seen that the prefetch threshold remains constant after a certain network cost, at which point the $StorageCost$ and $WaitCost$ dominate. Thus we can conclude that it is possible to prefetch larger amounts of data with decreasing network costs since P_T decreases.

Symbol	Description
s_o	size of an object
c_d	cost of disk space (per byte per second)
c_n	cost of network transmission (per byte)
l_o	object lifetime

Table 2.5: List of symbols used for comparing the network transmission cost versus the replication cost.

2.1.2 Client

We can compare the cost of transmission across the network to the cost of creating and maintaining a replica at the client by prefetching to predict the usefulness of the prefetched object.

Based on the definitions given in the table 2.5, we can write

$$\begin{aligned} \text{Disk space cost} &= s_o * c_d * l_o \\ \text{Network cost} &= s_o * c_n \end{aligned}$$

The values of c_d and c_n are the cost of disk space ($StorageCost$) and network transmission cost ($NWCost_{prefetchNW}$, calculated in the previous section), respectively.

Today disks are about \$.005/MB (e.g., 20GB disk costs \$100). We assume that a disk lasts 3 years. There are about 10^8 seconds in 3 years. So, to rent disk space the cost is $\approx$

$$\$.005/(10^6\text{bytes}*10^8\text{seconds}) \approx 10^{-16}\$/(\text{byte} * \text{second}) \approx 10^{-8}\$/(\text{byte} * \text{month})$$

$$\begin{aligned} StorageCost &\approx 10^{-8}\$/(\text{byte} * \text{month}) \\ NWCost_{prefetchNW} &\approx 10^{-8}\$/\text{byte} \end{aligned}$$

Hence

$$\begin{aligned} \frac{\text{Disk space cost}}{\text{Network cost}} &= \frac{c_d * l_o}{c_n} \\ &\approx \frac{10^{-16}\$/(\text{byte} * \text{second}) * l_o}{10^{-8}\$/\text{byte}} \\ &\approx 10^{-8}/\text{seconds} * l_o \end{aligned} \tag{2.2}$$

So, if an object's lifetime l_o is significantly more than 10^{-8} seconds (about 3 years) then the storage cost dominates. If its lifetime is considerably less than 3 years, then transmission cost dominates. The work in [26] examines the costs and benefits of prefetching popular long-lived objects. They evaluate a long-term prefetching algorithm which significantly improved steady state hit rates at modest bandwidth costs. These calculations indicate that the client can afford to prefetch an object if the local storage cost for the objects usage lifetime is lesser than the cost to actually prefetch. The $StorageCost$ in the prefetch threshold P_T equation 2.1 is thus dependent on the longetivity of the object and the network cost $NWCost_{prefetchNW}$. Thus equation 2.1 can be rewritten as

$$P_T = \frac{NWCost_{prefetchNW} * (1 + l_o)}{WaitCost_{demandNW} + NWCost_{demandNW}}$$

2.1.3 Server

In the above calculations the cost of maintaining an object at the server itself is being ignored. This term should be added to the P_T equation to obtain

$$P_T = \frac{NWCost_{prefetchNW} * (1 + l_o) + ServerCost}{WaitCost_{demandNW} + NWCost_{demandNW} + ServerCost}$$

Current web server machines cost upto \$2000 per month. A typical server cluster can be assumed to have around 500 such nodes each catering to around 30*10^6 requests per day [30]. For a typical server each request accesses around 10Kbytes. An example server provider [50] allows up to 50GB/month of burstable bandwidth for each server node. Hence the typical \$/byte cost for server storage is approximately

$$\begin{aligned} ServerCost &= \frac{2000 * 500\$/\text{month}}{50 * 10^3 * 500\text{MB/month}} \\ &= 4 * 10^{-8}\$/\text{byte} \qquad (2.3) \end{aligned}$$

Figure 2.3 shows the variation of P_T as the $ServerCost$ is varied. The range of values for $ServerCost$ are indicated. Once again the P_T becomes

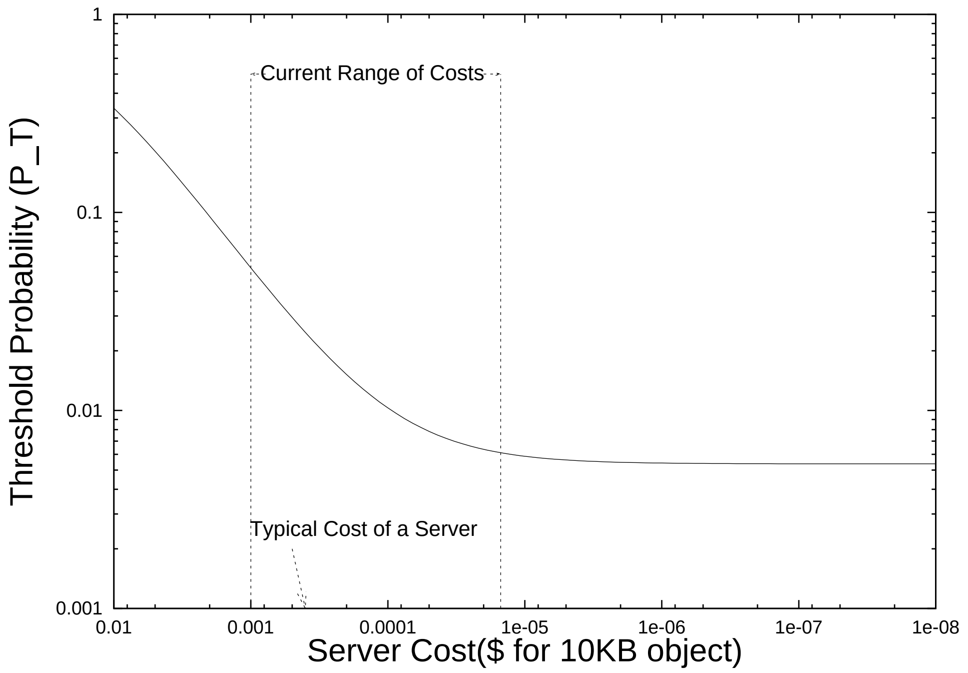


Figure 2.3: Decrease in prefetch threshold as server storage cost is decreased

constant as $ServerCost$ is dominated by $NWCost$s and $WaitCost$. It can be inferred that as the server technologies improve and costs fall, we can expect an increase the amount of prefetching feasible at the client.

2.1.4 Human wait time

The $WaitCost$ term might vary with the user in question and also with the time of the day. For example, one might assign more wait cost to a doctor who is waiting for a report in the middle of a surgery.

Varian [28] presents an argument wherein the total cost of time, C_T can be calculated from

$$C_T = [c + p(b)]\frac{1}{b}$$

where b is the bandwidth being used, c is the cost associated with the user and $p(b)$ is the dollar cost as a function of the bandwidth chosen. Hence, given a preferred bandwidth, b, one can estimate the range of c which will minimize the above total time cost. Also in Varian's [29] paper a pricing method based on usage and condition of the network is proposed whereby the user is charged per byte of data used based on the amount of congestion in the network. These results can be used to specify a range of values for $WaitCost$ depending on the user time and actual users perceived performance improvements.

2.2 Technology trends

Technology trends may favor more aggressive replication in the future. First, network and disk costs appear likely to fall much more quickly than human waiting time costs. Second, the deployment of commercial content-distribution networks (CDNs) may significantly reduce the network costs of prefetching by allowing clients to prefetch from a nearby CDN node rather than the origin server.

P_T decreases as technology improves and becomes cheaper. We can approximate Moore's law so that network and storage costs decrease at a rate of K every year.

$$P_T = \frac{\frac{NWCost_{prefetchNW}}{K^{t-t_0}} + \frac{StorageCost}{K^{t-t_0}}}{WaitCost_{demandNW} + \frac{NWCost_{demandNW}}{K^{t-t_0}}} \tag{2.4}$$

The plot of 2.4 is shown in figure 2.4 with K = 2, t_0 = year 2000. This

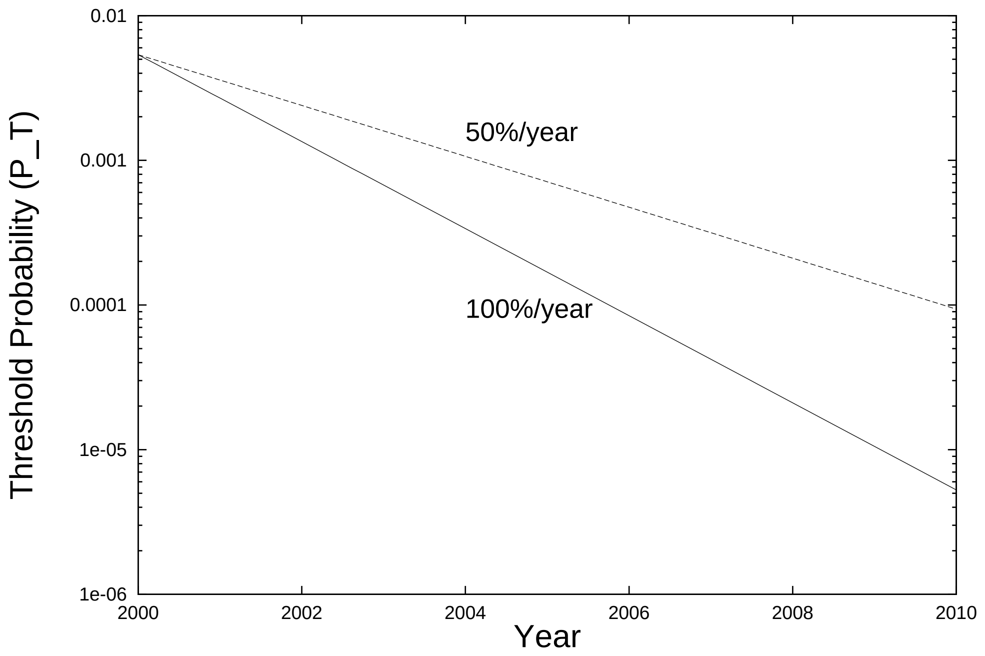


Figure 2.4: Effect of cheaper and better technology on prefetch threshold.

result is the combination of the results of the above cost assumptions.

Hence from this simple analysis, we can conclude that one can expect an increase in the amount of data that can be prefetched as the network infrastructure costs go down.

2.3 Heterogeneous devices

Another factor ignored in these calculations is the presence of heterogeneous devices. For example, a palmtop machine may have significantly less storage space than a desktop machine. The continuous usage of network may also be limited not only because of its high cost, variable performance and reliability but also because of the finite energy available to perform the operation. This can be modeled by increasing the $StorageCost$ term [32]. We have always assumed that the $prefetchNW$ and the $demandNW$ to be the same for the above results. By using different networks for demand fetching and prefetching one can decrease the P_T, e.g., by using a costlier network for demand fetch than prefetch (other things being the same).

From the above discussion of improving technologies, one can hypothesize that the decrease in the costs of bandwidth and disk space for hand held devices can lead to a further decrease in the storage cost term. The analysis of the statistics is out of the scope of this paper and is the goal of future research in this area.

2.4 Prioritized traffic

In this section we analyze the impact of prefetching on the network, the server and the client infrastructures. The above calculations were pessimistic in that they assumed prefetch traffic competed directly with demand traffic for resources at the network and the server. We will provide a brief survey of the various strategies that have been proposed to reduce the cost incurred by the network. At the server we measure the availability of spare capacity and propose mechanisms by which the an interference of prefetching requests with on-demand requests can be avoided. We study the usage patterns of the limited resources present at the client and come up with a novel resource management policy which can fairly, efficiently and automatically allocate and monitor the limited resources.

2.4.1 Network

One of the main concerns with aggressive prefetching is that the network might get congested because of the additional traffic that is generated by the increased traffic from prefetching. There have been many research projects which provide mechanisms for restraining and optimizing the prefetching algorithms.

- Crovella et. al., [31] present a *rate controlled* prefetching algorithm. This is a transport level method which can reduce the load on network router queues by maintaining the prefetch transfer rate sufficient to deliver it just in advance of users request.
- Another approach suggests the use of server based multicast instead of traditional unicast. In [33], Chuang and Sirbu present the *scaling law*,

$$\frac{P_m}{P_u} = N^k$$

  where P_m is the price of multicast streams to N nodes, P_u is the price of a unicast stream to a single receiver nodes, N is the number of multicast members in the network and k is the economies of scale factor ranging between 0 and 1. They find that this relation in independent of the network topology, network size and membership distribution and dependent only on the membership size, N. The idea is that multicasting prefetch can give better level of efficiency because of reduced overall bandwidth demand of content transmission since packet duplication only occurs when paths to multiple receivers diverge.

- Techniques such as digital fountain [51] and delta-encoding [52] can give lower priority to prefetching that demand traffic and so suite well to prefetching for content distribution and allow data transmission at a much lower cost than traditional network transmission.
- Bestravos [19] provide a protocol for Demand-based Document Dissemination (DDD) to reduce traffic. The model disseminates the most popular documents on server closer to clients in a hierarchical replication strategy which controls traffic over the network while maintaining load-balanced servers.
- The goal of the evolving IETF differentiated services (diffserv) [23] framework is to offer services without the need for per-flow state and signaling in every router. By aggregating many QoS-enabled flows with differentiated treatments within the network, diffserv eliminates the need to recognize and store information about individual flows in core routers. This technique can allow the prefetching mechanism to scale well.
- Application level support can be accomplished by prefetching based on the *time-of-day* heuristic. For example, the mobile service code can prefetch during periods of low activity such as nights or weekends.

The evaluation of the above schemes is the subject matter of future research in this area. We can estimate the useful spare network capacity that can be potentially used for improving the effectiveness of prefetching without burdening the infrastructure. The implementation issues for a prototype incorporating a combination of these strategies in beyond the scope of the current report.

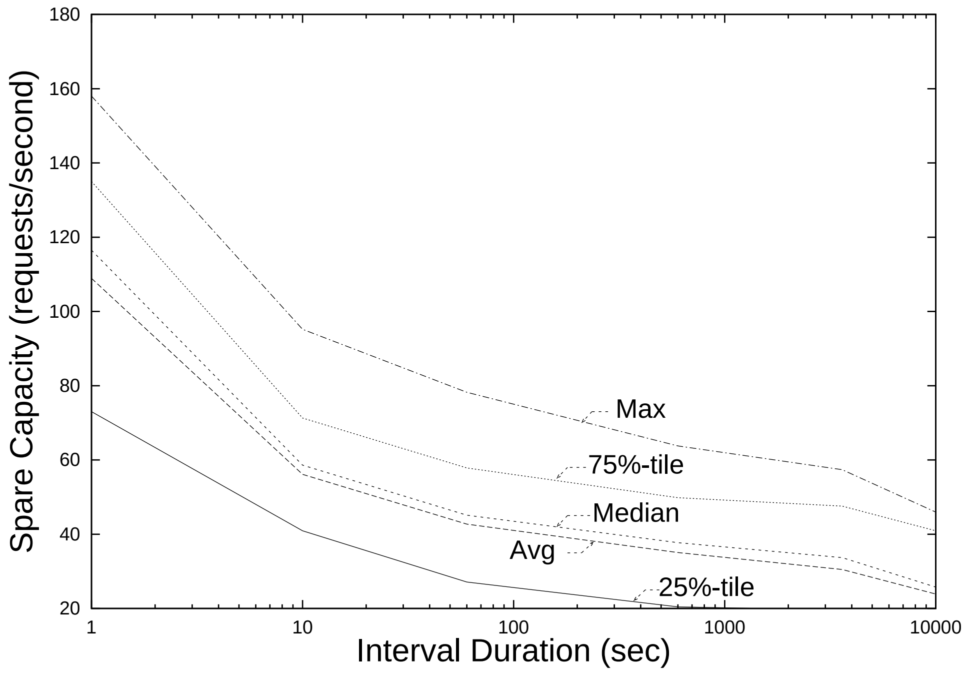


Figure 2.5: Distribution of spare capacity at server for different time intervals.

2.4.2 Server

Given the bursty nature of prefetching, one can expect the server to be swamped with prefetching requests. If the server processes these requests along with demand requests on a FCFS basis, then there is a potential problem of increased latency for the user waiting for his demand request. First we analyze the presence of spare resources (measured in terms of requests processed per second) at the server. If there is spare capacity then we propose methods by which it can utilized to serve prefetching and on demand requests fairly.

The amount of resources being consumed at the server is approximately proportional to the number of requests being processed per unit time. We analyzed a server trace obtained from the server web site for Olympics 1998. This workload was generated by a major Sporting and Event web site hosted by IBM, which in 1998 server 56.8 million requests on the peak day, 12% of which were to dynamically generated data [37]. The peak number of requests serviced by a server in unit time, can be approximated to the peak capacity of that server during that interval. The spare capacity available for prefetching in each time interval can be viewed as the difference of this maximum and the actual requests processed during that interval.

In Figure 2.6 we plot this spare capacity for the time granularity of one second, 10 seconds, one minute, ten minute, one hour, and four hours. It can be seen that a maximum of 167, 95.2, 78.23, 63.77, 57.38, and 41.86 additional requests can be processed per second, respectively, for each of the time granularities. Similar analysis is shown in figure 2.7 for the server accesses to Macys Web server for one day with a maximum of 77, 44.6, 35.85, 25.88, 22.74, 20.93 additional requests that can be processed for each of the time granularities above.

Figure 2.5 shows the distribution statistics (25%tile, mean, median, 75%tile, max) of capacities at different intervals of consideration for the Olympics trace.

Due to this available spare capacity at the server, it should be efficiently used to process prefetch requests (along with on demand requests). One assumption in [21] is that the on-demand requests and prefetch requests are of same priority. A potential hazard of this is that machine As on-demand requests (for which a user is waiting) might be delayed because of a set of prefetch requests from another machine B (which a user might not even use). We propose a dual queue model at the server which schedules the requests based on

- Non preemptive queues with higher priority to on demand queue elements.

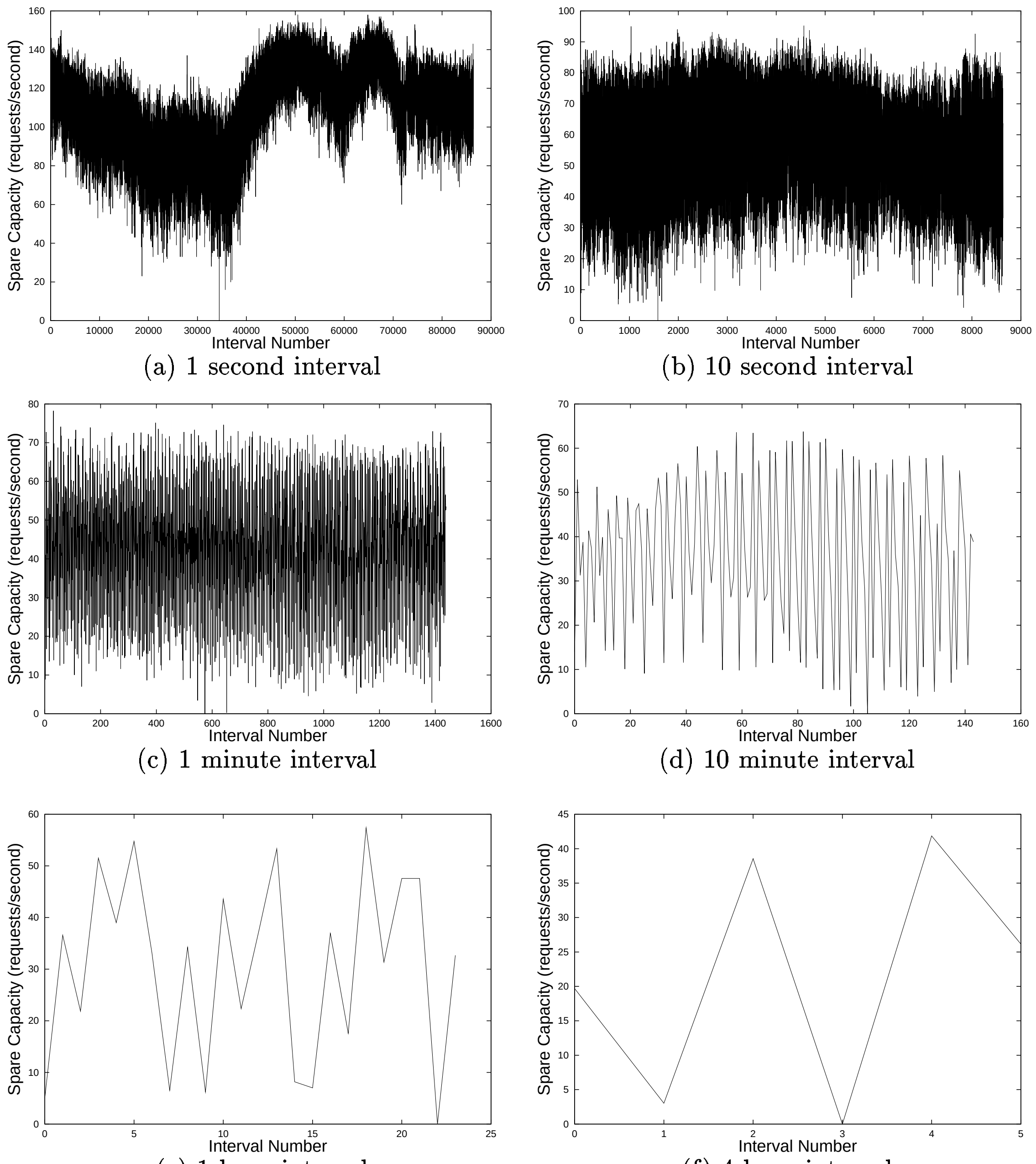
- Preempt and resume queues where prefetch requests are pushed back while on-demand get served.

Because of the increased load at the server, there will be an increase in the CPU cycles and disk bandwidth. CPU cycles cost can be negligible compared to disk bandwidth. In most cases, disk bandwidth cost will be less than network bandwidth cost if a disk IO is less than just a network transmission. Hence in all the above calculations a *free IO* system was assumed instead of the combined cost of disk seek, c_s (\$ per byte), and bandwidth.

One can use server-based speculative prefetch methods such as in [20] which reduced both server load and service time. In our current system we assume the mobile service code avoids prefetching when when it would interfere with on-demand traffic. In general, the above methodologies can be extended to include all server processing costs. An interesting question for future research can be the reduction in the locality and hit rate that might be a result of not processing prefetching requests (which tend to be to closely related objects).

2.4.3 Client

The effect of multiple untrusted mobile services running on the client can potentially cause a resource contention at the client. The main resources that we are interested in are disk space, network bandwidth, CPU cycles and memory. In Chapter 3 we present a trace-based analysis of the number of services that are accessed by a typical client. We also provide an estimate of the network bandwidth and disk space used by the services and the clients accessing them. These measurements suggest the availability of spare resources that can be used effectively to support disconnected operations. They also warrant a need for a flexible resource management framework so that the resource contention does not end up in a denial of service attack. These and other related issues are discussed in detail in Chapter 4.

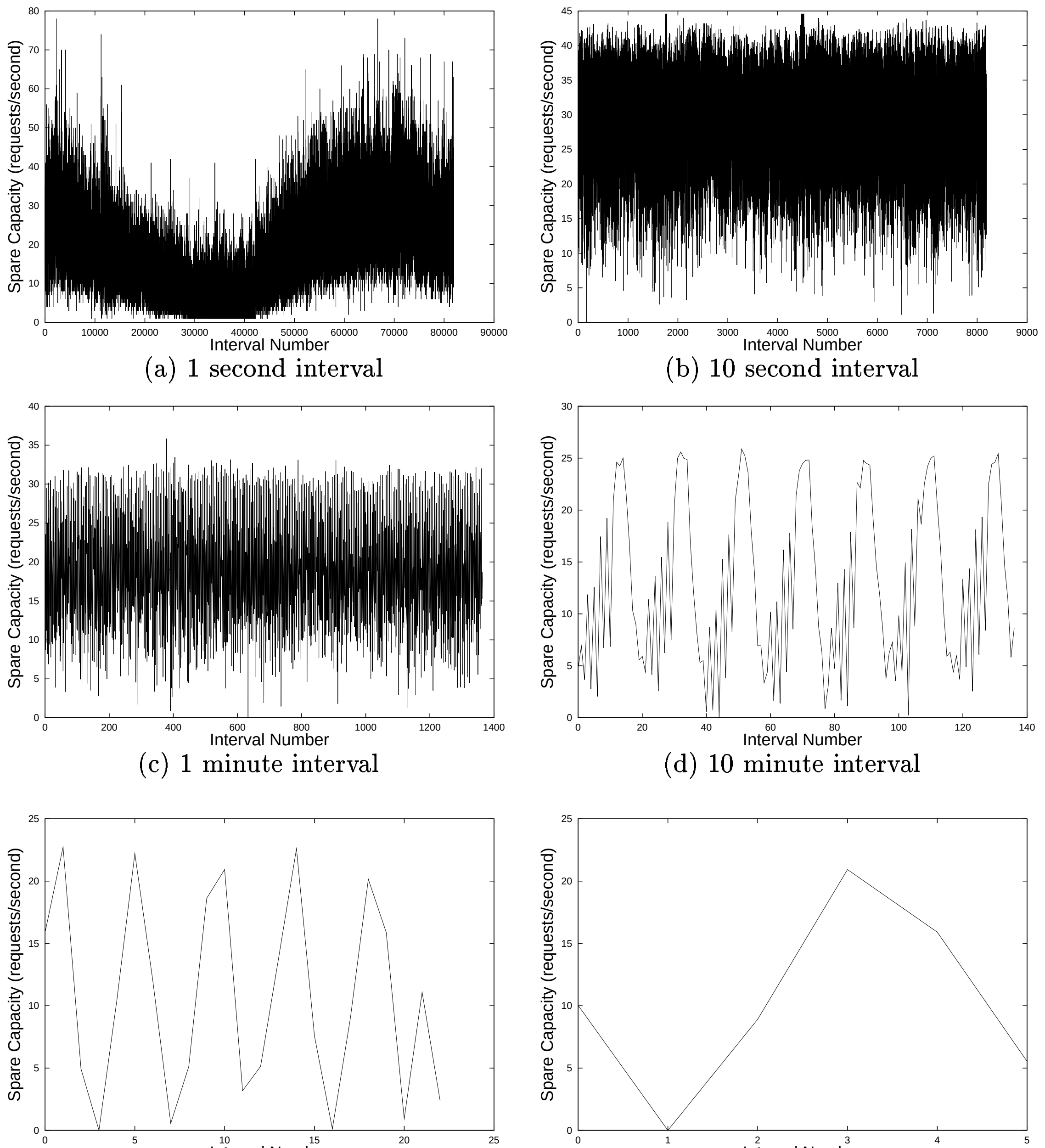


(a) 1 second interval (b) 10 second interval

(c) 1 minute interval (d) 10 minute interval

(e) 1 hour interval (f) 4 hour interval

Figure 2.6: Spare capacity available at the server at different time granularities for Olympics Server.



(a) 1 second interval

(b) 10 second interval

(c) 1 minute interval

(d) 10 minute interval

(e) 1 hour interval

(f) 4 hour interval

Figure 2.7: Spare capacity available at the server at different time granularities for Macys Server.

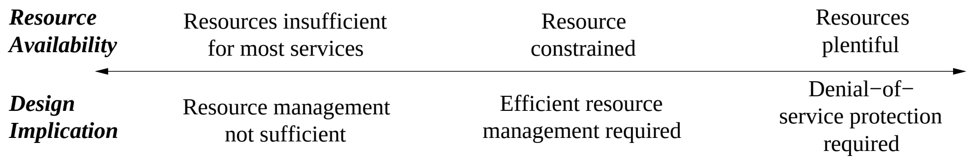
Chapter 3

Client resource requirements

This chapter examines the impact of web workloads on the requirements for client infrastructures that support disconnected service access.

3.1 Design space

Systems could operate in one of three regimes illustrated by Figure 3.1. First, services may demand large amounts of resources in order to support disconnected operation, and the aggregate demands of services may significantly exceed the total capacity of the infrastructure. In that case, providing a general infrastructure for disconnected operation may not be feasible. At the other extreme, resources may be plentiful relative to the likely demands of services. In that case, infrastructure should focus on providing mechanisms for shipping code to clients, running that code securely, and, perhaps, limiting "resource hogs" to prevent deliberate or unintentional denial of service; beyond that, resource management is not likely to determine system performance. The middle case is more challenging: if resources are constrained but



Resource Availability
Resources insufficient for most services
Resource constrained
Resources plentiful
Design Implication
Resource management not sufficient
Efficient resource management required
Denial−of−service protection required

Figure 3.1: Design space for resource management.

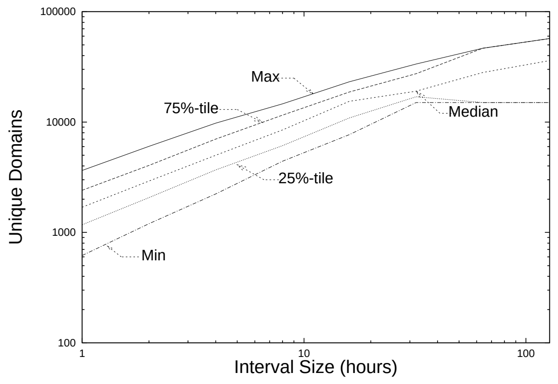
sufficient for applications to provide reasonable disconnected service, then a key task for the infrastructure is partitioning resources fairly among untrusted code modules to maximize global utility.

It is difficult to specify workload requirements definitively. First, applications vary widely. Some can easily operate in disconnected mode with few additional resources compared to their normal requirements. For example, a daily comic site could be prefetched in its entirety once per day for little more cost than fetching it on demand. Other services are not suitable for disconnected operation at all because they require live network communication (e.g., a stock ticker or phone call) or would require unreasonable amounts of state to be replicated at clients for disconnected operation (e.g., a search engine). Many services may operate between these extremes: by expending additional resources (i.e., prefetching data that may be needed in the future or buffering writes or outgoing requests) they can support disconnected operation. Examples of this class may include many shopping, news, entertainment, and corporate services.

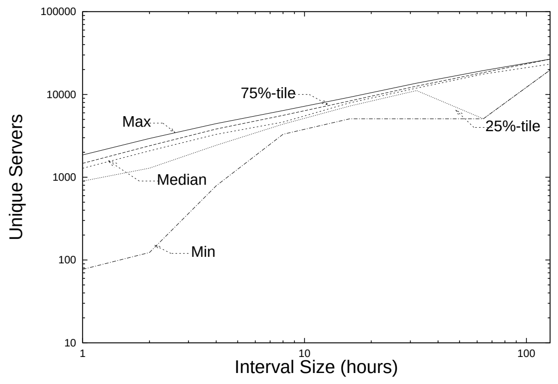
Note that the application-specific adaptation afforded by mobile code often may allow services to provide degraded, though still useful, service when disconnected. For example, although a stock trading service probably would not accept orders when disconnected, the company providing the service may desire to operate in "degraded" disconnected mode by turning off the order service but providing other services: a portfolio summary, news bulletins related to the user's holdings, a history of past orders, and so on. In this example, even though the "primary" function for a service is inoperable when disconnected, the service may gain significant benefit from mobile code that allows the user to access a subset of the services when disconnected.

A second challenge to precisely specifying workload requirements is that the potential demands of an individual service may span a wide range. In particular, prefetching is a common technique to cope with failures. Often, the more data that are prefetched the larger the fraction of client requests that can be handled during disconnection, and the better service that can be provided. For example, a news service might prefetch headlines and abstracts in a resource constrained environment, full text of articles from a few major sections (e.g., headlines, international, sport, finance) in a less constrained environment, and so on up to full text, pictures, and video from all articles in an unconstrained environment.

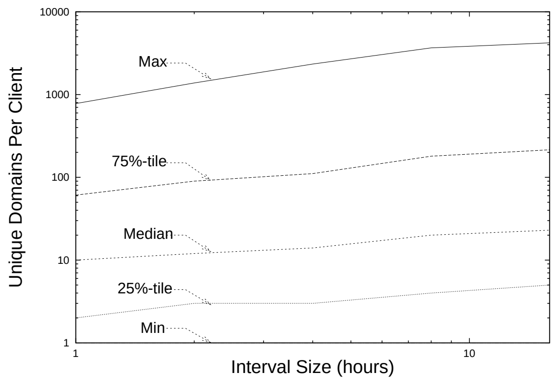
Given the methodological challenges posed by the wide range of web service behaviors, we take the following approach. We examine the average demands of current web workloads in order to assess approximately how many additional resources may be available for supporting disconnected operation. This provides a rough guide to the constraints of the system. We focus primarily on the bandwidth and disk space requirements of hoarding and related techniques such as prefetching. Although other techniques – write-buffering message queues, and application-specific adaptation – are also important for coping with disconnection, the resource demands of services using these techniques may not be significantly higher than the normal demands of the services. In contrast, aggressive prefetching may dramatically increase the network bandwidth and disk space demands of applications and therefore presents



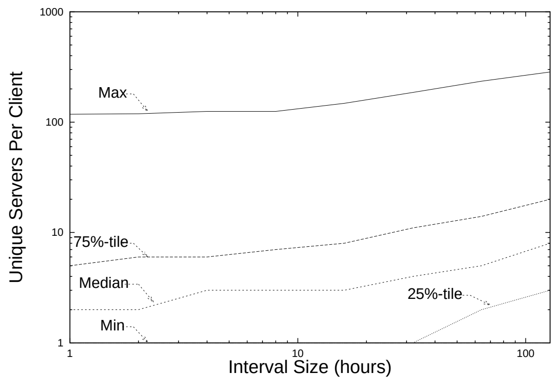
(a) Squid with shared proxy



(b) UCB with shared proxy



(c) Squid-1-day per-client



(d) UCB per-client

Figure 3.2: The number of domains (a,c) or individual IP addresses (b,d) accessed by each active shared proxy (a,b) or per-client cache (c,d) over different time scales.

the most direct challenge to scalability.

3.2 Workload characteristics

The operating regime of the system with respect to resources is largely determined by the workload. We study several client traces of web service workloads to determine how many services are in a client's working set and

Workload	Date	NClients	NServers	Sessions
Squid-P	3/28/00 – 4/03/00	1	131193	1557875
Squid-C	3/28/00	107	52526	403235
BU-P	1/17/95 – 5/17/95	1	4614	56789
BU-C	1/17/95 – 5/17/95	33	4614	68949

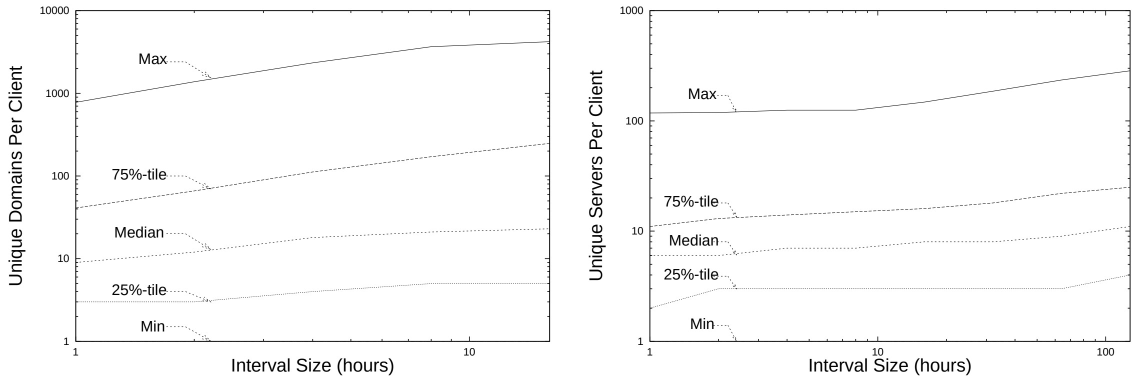
Table 3.1: Web access trace parameters.

how much data those services access. From this, we derive an estimate of the amount of spare capacity machines are likely to have to support disconnected operation and argue that it may be feasible for services to prefetch 10 or more times as much data as they access on demand.

We analyze two traces: Squid [36], which contains 7 days (3/28/00 – 4/03/00) of accesses to the Squid regional cache at NCAR in Boulder, Colorado that serves requests that miss in lower-level Squid caches, and the first seven days from UC Berkeley Home-IP HTTP traces [38]. Table 3.1 summarizes key parameters for our access pattern traces. The simulator uses information in the traces to identify cachable and non-cachable pages as well as stale pages that require reloads. In this analysis, we study the resource demands from each service, where a "service" is defined by the set of URLs from the same DNS domain (for the Squid trace) or from the same IP address (for the UCB trace, which identifies origin servers using a 1-way hash of the IP address).

3.3 Working set study

We study two cache configurations. In the first, we simulate a separate cache (Squid-C and BU-C in Table 3.1) at each client IP address in the trace. Since the UCB trace tracks individual client machines, this corresponds to the environment that might be seen by code that seeks to support mobile clients



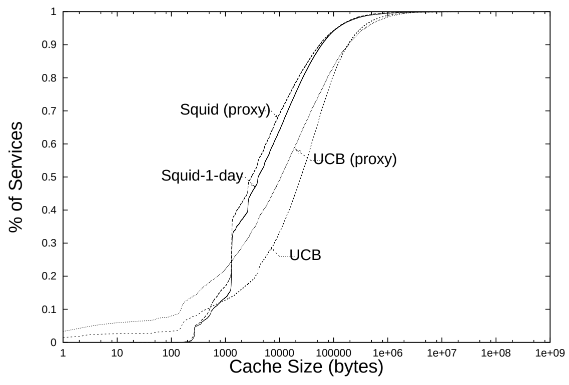
(a) Squid-1-day per-client (b) UCB per-client

Figure 3.3: The range of the maximum number of services accessed by different clients.

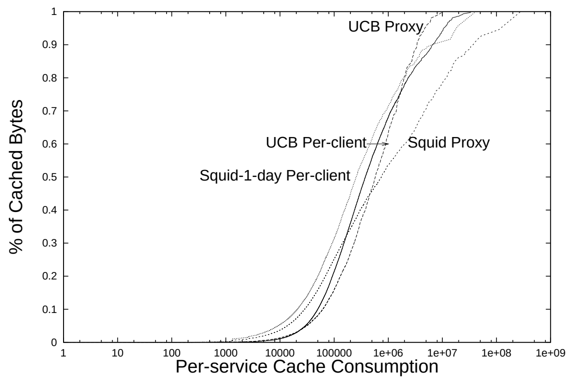
as well as to improve client performance and availability. In the second, we simulate a proxy cache shared by all clients (Squid-P and BU-P) in the trace. This configuration does not support client mobility, but it may improve service availability or performance. Note that the Squid traces remap client IDs each day, so we only examine the first day of the Squid workload in our per-client cache analysis. We refer to this workload as Squid-1-day for clarity.

Figure 3.2 summarizes the number of services accessed by each cache over different time scales to provide a rough guide to the "working set" size of the number of services a cache might have to host. Each graph summarizes data for a different trace/cache configuration. Each graph shows the minimum, 25th percentile, median, 75th percentile, and maximum number of services accessed by a cache over intervals of the specified length.

Figure 3.3 summarizes the distribution of the per-client maximum number of services accessed at different interval sizes for UCB. For example, if a client accesses 3 services during the first hour, 7 during the second, and 2 dur-



(a)



(b)

Figure 3.4: Per-service cache size demands. Cumulative histogram of the fraction of services that occupy the specified size.

ing the third, that client's maximum working set size for 1 hour is 7 services. The plot shows the range of maximums at different clients.

Three features of these distributions stand out. First, the working sets of a cache can be large over even modest time scales. For caches at individual clients, 25% of 16-hour-long intervals contain accesses to more than 10 services in the UCB trace and 200 services in the Squid-1-day trace(not shown). For proxy caches, 25% of 16-hour-long intervals contain accesses to more than 8,000 services in the UCB trace and 18,000 services in the Squid trace. Second, these working sets vary widely from client to client. For example, in the UCB trace 25% of clients never use more than 3 services in a 16-hour period and in the squid trace 25% use at least 148 services during at least one 16-hour period. Third, the scale of the number of services that an active proxy may have to host is large; scaling, say, a Java virtual machine to gracefully handle tens of thousands of mobile code extensions may require considerable research and engineering effort. The scale of the individual client working sets, conversely,

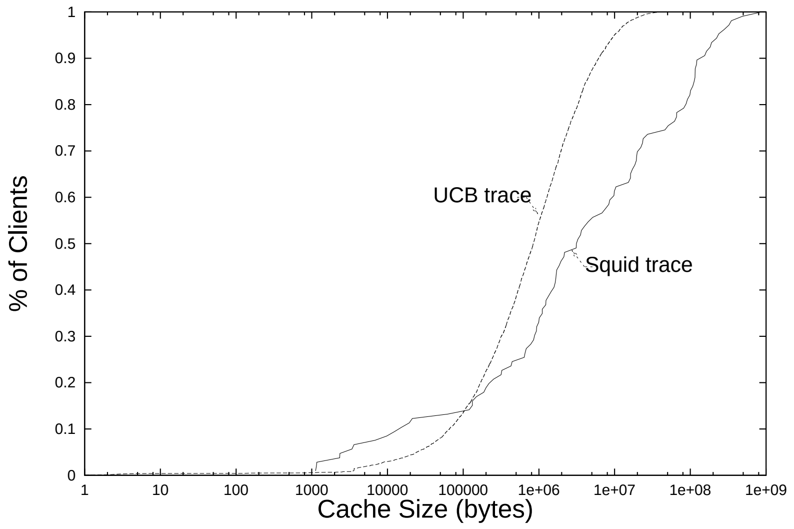


Figure 3.5: Per-client cache size demands.

appears relatively manageable.

These features have several implications on the system design. First, they suggest that resource management could be a significant challenge for some caches where many services compete for resources. They also suggest that the framework must be self-tuning over a wide range of situations both because the range of demands is large and because the number of services under consideration is often too large for convenient hand-tuning.

3.4 Disk space consumption

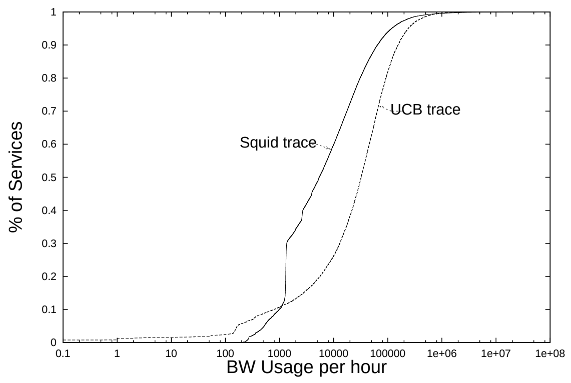
Figures 3.4 and 3.5 show the cumulative distribution functions (CDF) of disk space consumption of individual services and of the collection of services hosted by a cache, respectively. In Figure 3.4, the x-axis is the approximate service working set size (the amount of data the service accesses during the trace) and the y-axis is the fraction of services with working sets of the specified size or smaller. In Figure 3.5, we plot the total disk size consumed by all services at each client on the x-axis with the fraction of clients with disk consumption below the specified amount on the y-axis.

The graphs indicate that per-service demand fetched data typically have modest footprints in caches; for the Squid and UCB traces, 90% and 80% of services consume less than 100KB. A few large services consume large amounts of disk space. Overall, for the UCB per-client caches, the total data footprint of all services accessed by a cache is below 10MB for all but a few percent of clients. The Squid data footprints are significantly larger, but note that each "client" in the Squid trace may correspond to a lower-level Squid proxy serving many clients.

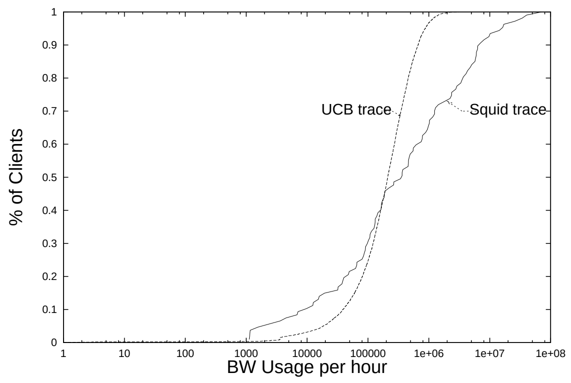
These data have several implications with respect to scalable resource management. First, the wide range of per-service and per-cache demands suggests the need for a flexible approach. For example, allocating 100KB to each service would waste large amounts of space for many services and be far to little space for others. Second, for the desktop clients that presumably make up most of the UCB trace, the amount of disk space consumed for caching demand-fetched objects is relatively small compared to the total disk capacity of such machines. This suggests that disk space considerations may allow significant prefetching. We discuss this issue in more detail below.

3.5 Network bandwidth consumption

Another key resource for supporting disconnected operation is network bandwidth. Figure 3.6 summarizes the network bandwidth consumption by the trace workloads. As indicated in Figure 3.6(a), most services have low average

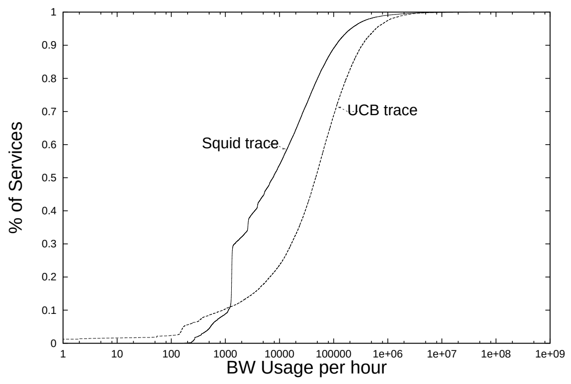


(a) Per-service distribution

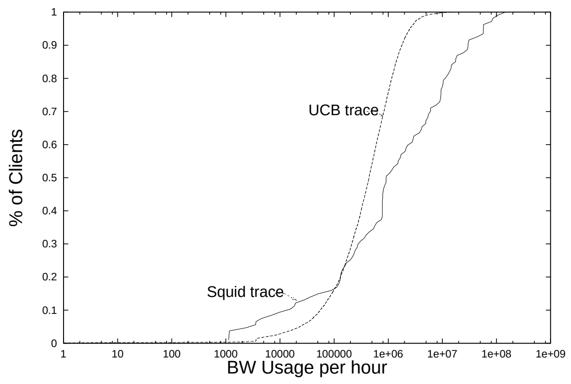


(b) Per-client distribution

Figure 3.6: Distribution of average bandwidth usage.



(a) per-service distribution



(b) per-client distribution

Figure 3.7: Distribution of maximum bandwidth usage.

bandwidth requirements of a few tens of KB per hour or less. This suggests that caches with modest bandwidth can support relatively large number of prefetching services. Figure 3.6(b) shows the hourly bandwidth usage at each client. 90% of clients demand less than 2 MB/hour – 8% of a 56Kbit/s modem and 0.4% of a 1Mbit/s DSL. This suggests that for services where prefetching is not time-critical, considerable spare bandwidth may be available.

Similar analysis for maximum bandwidth usage shows the distribution across clients and services of the maximum bandwidth demand during any hour for that client in Figure 3.7. 90% of clients demand less than 10MB per hour – 40% of a 56Kbit/s modem and 2% of 1Mbit/s DSL connection – during their worst hour. Most services need only a few hundreds of KB in their worst hour. This suggests that considerable spare bandwidth may be available even during periods of high demand.

The above data suggests that in terms of raw capacity, client disks and networks may be able to provide considerable resources to support disconnected operation. For example, in the UCB workload, 95% of client systems could allow each service to prefetch approximately 10 bytes for every byte served on demand and still consume less than 1% of a 10GB disk. Similarly, a client on a 1MBit/s DSL connection could prefetch 10 bytes for every byte served on demand and consume less than 4% of the raw connection bandwidth for most clients. Given this spare capacity, one might ask: is it reasonable for a service to fetch and store 10 or even 100 times as much data as is accessed on demand? In classic operating systems terms, fetching 10 or 100 bytes per byte accessed might seem excessive. However, the long latencies and significant failure rates of wide-area networks may make doing so a reasonable strategy. Furthermore, whereas the costs of prefetching (network bandwidth, server processing, and disk space) fall rapidly over time, the value of human time remains approximately constant. Thus, prefetching a large number of bytes to save a small amount of latency or risk of disconnection becomes a more attractive strategy over time. This result thus proves our earlier claim that aggressive prefetching is feasible.

Chapter 4

Resource management system for clients

In this chapter we first outline the requirements for resource management in this client environment running many untrusted services which access the local limited resources such as disk space and bandwidth. Then we sketch the details of such a system. A simple resource management system should meet the following requirements.

1. **Isolation.** The policy should prevent denial of service attacks by bounding the resources consumed by any service, and it should prevent aggressive services from interfering with other services. This goal is motivated by the fact that the target workload comprises large numbers of untrusted modules competing for resources.

2. **Efficiency.** The policy should divide resources among services so as to maximize overall utility. In contrast with the first goal, which might be achieved by placing a loose upper bound on worst-case resource demands,

this goal implies careful resource allocation may be necessary. This goal is motivated by our expectation that systems are likely to have sufficient resources to be useful for disconnected operation, but that they probably will not have sufficient resources to prefetch everything that applications might want.

3. **Self-tuning.** The policy should not require user intervention or hand tuning. This goal is motivated by the large number of services that a client may host as well as the wide range of service demands and system configurations likely to be encountered.

One observation worth noting is that due to the untrusted nature of the mobile services, the system has to base its allocation policies strictly on observable phenomena. We argue that one such phenomenon is the amount of useful work the service is doing for the user, i.e., the combination of the number of times it is used and the amount of useful data it transfers to the user.

4.1 System architecture

The resource management architecture consists of four components: popularity layer, scaling layer (time-scaled popularity), per-resource resource managers and override module. The overall architecture of the system is shown in figure 4.1

The general overview of how this system works is as follows. The popularity layer monitors the activities of services (mobile code) in the system and tracks the popularity of each. The scaling layer takes the output of the popularity layer and converts it to a proportional per-resource, per-service resource allocation. The override module, if active, will override the service's resource allocation by a manually specified value. Through the API provided by the scaling layer, the per-resource manager is able to access the fraction of each resource to which a service is entitled. Based on this information, the per-resource manager enforces the resource usage on each service in the system.

The per-resource resource managers and override modules can be based on existing mechanisms discussed here:

- The JRes project [9] proposes a Java API for resource management. The limitation of JRes prototype is that for all the processes in the system, an equal share of system resource is given. This idea is simple but not as efficient and applicable for a dynamic environment with untrusted mobile code.

- The Resource Container [10] paper presents a precise mechanism for tracking resource usage per process. The paper suggests that to guarantee proper resource accounting a *resource container* should be notified of any access a process (and all its threads) makes to a resource. We are using this concept to account for resources such as CPU cycles, network bandwidth, disk space, and memory space through resource containers.

- Start-time Fair Queuing (SFQ) scheduling [11] is a computationally efficient, work-conserving algorithm, which achieves fairness of network bandwidth allocation regardless of variation in a server capacity. The proportional-share schedulers for stateless resources, such as CPU and

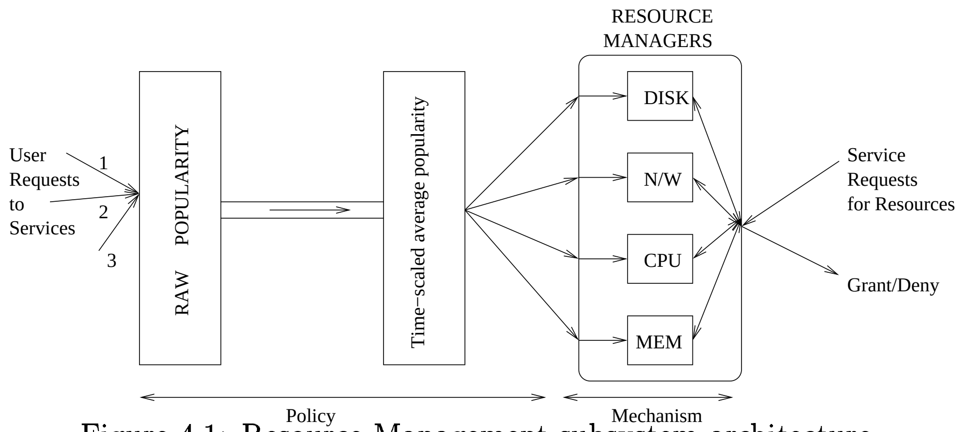

RESOURCE MANAGERS
User Requests to Services
1
2
3
RAW POPULARITY
Time-scaled average popularity
DISK
N/W
CPU
MEM
Service Requests for Resources
Grant/Deny
Policy
Mechanism

Figure 4.1: Resource Management subsystem architecture.

network, enforce resource limits by scheduling requests and threads using SFQ.

We extended these methodologies to fit in the dynamic environment by combining them with some internet-oriented approaches. For example, with combination of JRes and SFQ, we could not only limit the total amount of resources each program can use in the system, but also manage the resources of each process more dynamically as programs enter and exit the system.

The popularity layer and scaling layer are developed based on our policy which divides resources fairly among the services. This is the topic of interest for the rest of this report.

4.2 Policy requirements

The policy used for allocating resources should provide isolation, efficiency and automation as described above. A potential problem with standard resource management polices – such as LFU or LRU for cache replacement or FIFO or round-robin for CPU or network scheduling – is that these policies reward increasing resource demands with increasing allocations: as a program references more data, it is given more memory; as it spawns more threads or sends more network packets, it gains a larger fraction of those resources. Such approaches provide global allocation of resources that can meet the goal of efficiency (assuming that each application's requests have similar utility.) Such an approach also meets the goal of self-tuning. However, this approach is vulnerable to denial of service attacks.

A second simple approach is to give each service an equal share of resources. But such an approach faces a dilemma: making that fixed amount too large risks denial of service attacks while making it too small thwarts construction of useful services. For example, browser cookies represent requests from untrusted sources to store data on client machines, but limitations on allowed cookie size and number of cookies prevent construction of, say, a "disconnected Hotmail." On the other hand, if cookies large enough to contain a respectable inbox and outbox were made available to all services, a user's disk might fill quickly.

Hence we hypothesize that a popularity-based policy will have two properties that mix the benefits of these two simple approaches:

- Competitive with global policies for benign workloads
- Performance isolation for malicious workloads

4.3 Popularity-based resource policy

Given these constraints, a resource management system for mobile services should attempt to forge a compromise between static allocations that require no knowledge about users or services and dynamic approaches that require unrealistic amounts of knowledge about users or services. Our goal is to construct a dynamic allocation framework that can make reasonable, albeit not perfect, allocation decisions based on information about users or services that can readily be observed by the system and that are not easily manipulated by the services. We use service "popularity" as a crude indication of service priority, and allocate resources to services in proportion to their popularities. This approach is based on the intuition that services that users often access are generally more valuable to them than those the they seldom use. It also has the attribute of providing users with better service for those services that they often access.

Our approach is simple: for each resource and each service, the system maintains an exponentially decaying time-average of the number of requests by the user agent to the service. The resource manager for each resource allocates the resource to each service in proportion to the service's time average popularity as a fraction of the popularity of other services. Our resource schedulers are work conserving: if one service uses less than its full share of a resource, the excess is divided among the remaining services in proportion to their scaled popularities.

A key idea in the system is that separate scaled per-service popularities are maintained for each resource, and each resource uses a different timescale for computing its time average popularity. This is because the appropriate definition of "popularity" varies across resources because different resources must be scheduled at different granularities. In particular, "stateless" resources such as CPU can be scheduled on a moment-to-moment basis to satisfy current requests. Conversely, "stateful" resources such as disk not only take longer to move allocations from one service to another but also typically use their state to carry information across time, so disk space may be more valuable if allocations are reasonably stable over time. Thus, the CPU might be scheduled across services according to the momentary popularity of each service, while disk space should be allocated according to the popularity of the service over the last several hours, days, or longer. Other resources — such as network bandwidth, disk bandwidth, and memory space — fall between these extremes.

Although having different time scales for different resources might appear to introduce the need for extensive hand-tuning, we avoid this by choosing each resource's time scale to be proportional to the state associated with the resource or typical occupancy time in the resource for a demand request. For example, for disks, we count the number of bytes delivered by the system to the HTTP user agent and rescale the per-service running popularity averages by multiplying them by $\frac{1}{2}$ each time $diskSize$ bytes are delivered.

For network and CPU scheduling, we use the weighted sum of two popularities with each averaged over a different time-scale. The first represents the share of "demand" resources that should be allocated to allow a service to respond to user requests. This time scale should be on the order of the time needed to respond to a user request. We use 10 seconds. The second term represents the share of background CPU and network resources that should be allocated to allow a service to, for example, prefetch and write back data. Since these background actions primarily support disk usage, we use the disk's timescale here so that services are granted "background" network and CPU resources in proportion to the disk space they control. Since we wish to favor demand requests over background requests, we weight the first term much more heavily than the second in computing the total CPU and network resource fractions for each service. In particular, suppose that the scaling interval for the demand term is t_1, that the scaling interval for the background term is t_2, and that we scale the running average by $\frac{1}{2}$ at the end of each interval. If requests arrive at some rate r, then the total raw weight for the demand term is about $t_1 r + \frac{1}{2} t_1 r + \frac{1}{4} t_1 r \ldots \simeq 2 t_1 r$. Similarly, the total raw weight for the background term is about $2 t_2 r$. Therefore, to allow demand requests to dominate background requests during the first seconds after a demand request, we weight the demand term by a factor of $100 \frac{t_2}{t_1}$. During periods of idleness, the second term becomes dominant in roughly 100 seconds.

To verify our hypothesis, we have compared our popularity-based policy with other allocation policies such as fixed-allocation policies and with globally optimizing policies such as LRU. The results of this work are presented in the next section.

4.4 Evaluation

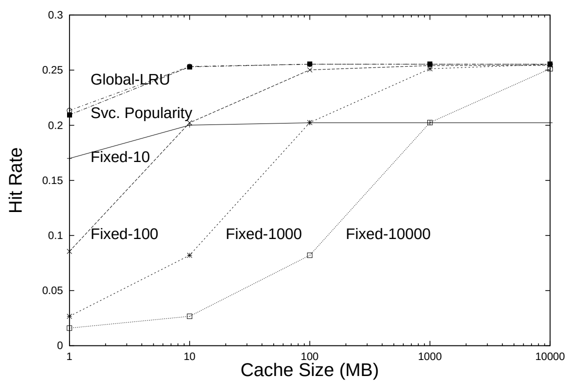
The simulation experiments in this section test whether a simple popularity-based policy can meet the three goals – isolation, efficiency, and self-tuning – outlined above. We use the same trace files we used to study the workload characteristics in Chapter 3.

We first study the resource management algorithms in the context of disk space management by examining three algorithms: (1) traditional *LRU* cache replacement that emphasizes global performance, (2) *Fixed-N*, which supports performance isolation by dividing the cache into N equal parts and allowing each of the the N most recently accessed services to use one part, and (3) *Service Popularity*, which allocates disk spaces in proportion to each service's time-scaled popularity as described above.

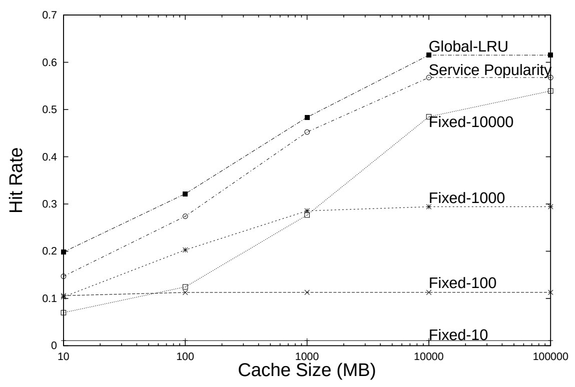
A key challenge in studying web services is that as indicated in Chapter 3, services' prefetching demands, prefetching strategy, and prefetching effectiveness vary widely. It is not practical to simulate application-specific prefetching and adaptation for each of the thousands of services that appear in our trace. The key observation that makes our analysis tractable is that for the purposes of evaluating resource management algorithms, it is not necessary to determine the impact of prefetching on the service that issues the prefetching; one may assume that a service benefits from its own prefetching. What is more relevant is the impact that one service's prefetching has on *other* services.

So, rather than simulating what benefit a particular service gains from prefetching, we focus instead on the impact that services' resource demands have on other services' performance. We simulate prefetching by a service by fetching sets of dummy data that occupy space but that provide no benefit.

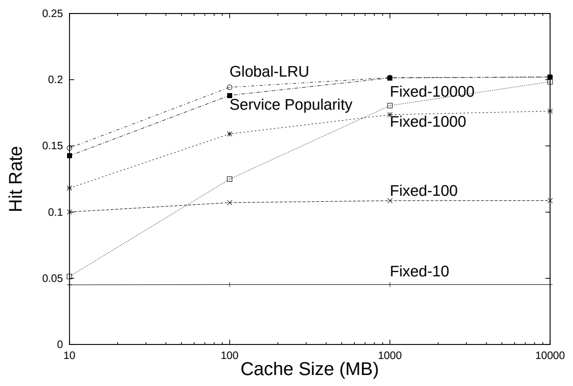
Figure 4.2 shows the hit rate of the LRU, Fixed-N, and Service Popularity algorithms as we vary per-client cache size (figure (a)) or total cache size (figures (b)) for UCB trace. In this experiment no services prefetch. This experiment thus tests whether the algorithms allocate resources fairly and efficiently when all services are equally aggressive relative to their demand consumption. In such environments, LRU works well because it optimizes global hit rate. Conversely, Fixed-N's performance suffers because it allocates the same amount of space to all services and because the parameter N must be chosen carefully to match the size of the cache. The Service Popularity



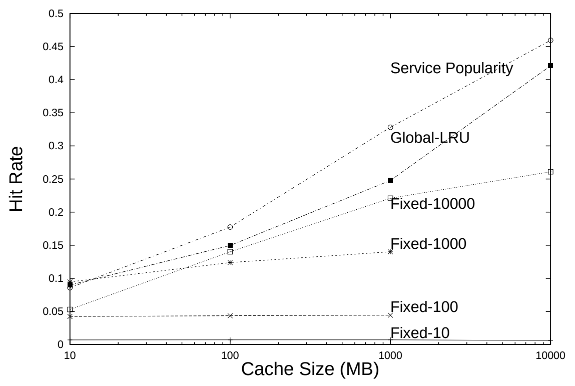
(a) UCB per client cache



(b) UCB shared proxy



(c) Squid per client cache

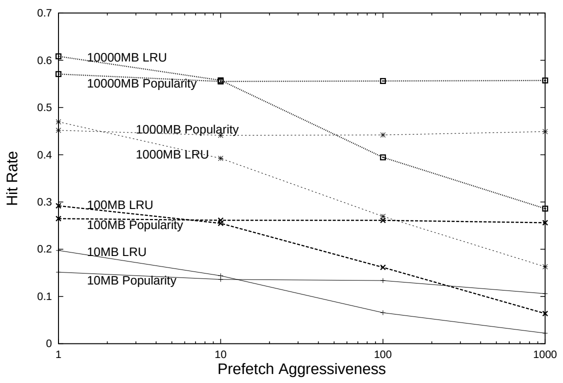


(d) Squid shared proxy

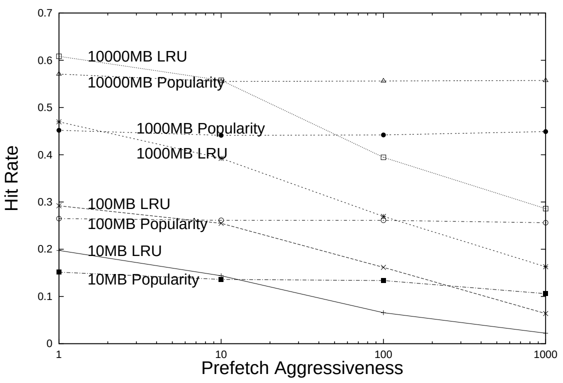
Figure 4.2: Cache replacement policy: Cache hit rate v. cache size.

algorithm is competitive with LRU across a wide range of cache sizes for both workloads and for both the per-client and proxy cache configurations. The results of the Squid traces are qualitatively similar. These results suggest two things. First, they indicate that the service popularity algorithm is reasonably efficient: it partitions resources nearly as well as the global LRU algorithm. Second, they provide evidence that the use of time averages proportional to the "natural frequency" of disk residence time supports our goal of developing a self-tuning algorithm.

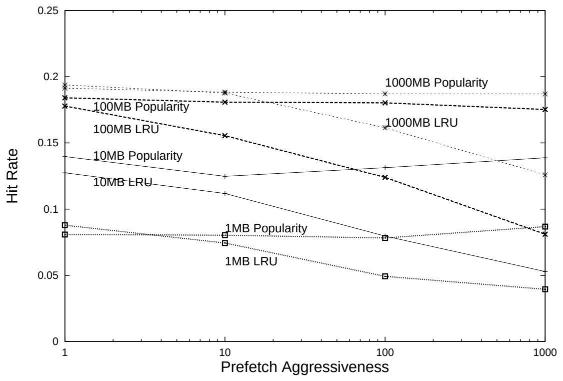
In Figure 4.3 we examine what happens when prefetching aggressiveness



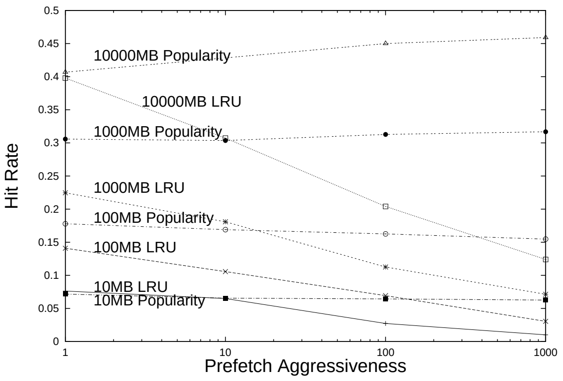
(a) UCB per client cache



(b) UCB shared proxy



(c) Squid per client cache



(d) Squid shared proxy

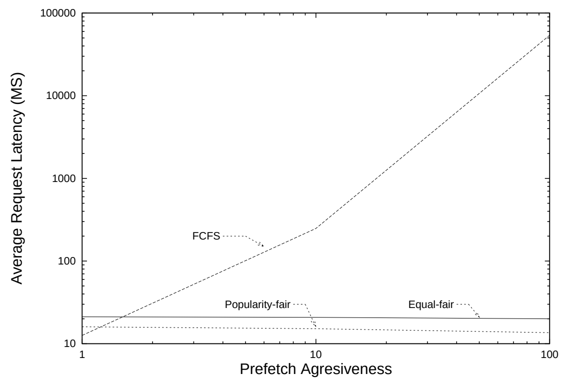
Figure 4.3: Cache performance with 20% of sites prefetching.

varies across services. We randomly select 20% of the services and introduce artificial prefetch requests from them. For each demand request, a prefetching service fetches ten objects whose total size is the x-axis value times the size of the demand object. The remainder of the services do not prefetch. The figure plots the performance of the services that do not prefetch. If a system that provides good isolation, the performance of less aggressive services should not by hurt by more aggressive services. In this experiment, when prefetching is restrained, the Popularity and LRU algorithms are competitive. However, as prefetching becomes more aggressive, the performance of non-prefetching sites suffers under LRU, whereas their performance under Popularity-based replacement remain largely unaffected.

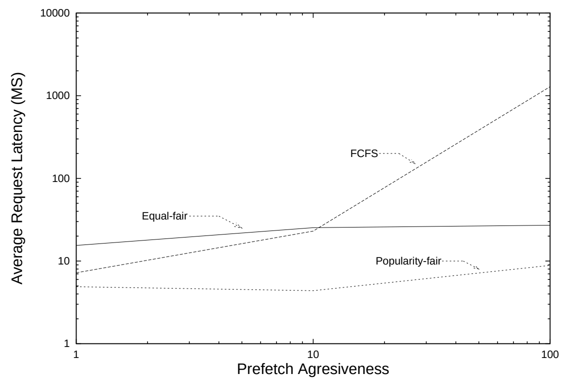
Figure 4.4 evaluates the resource management approach for network bandwidth. We consider three network schedulers: (1) FCFS which services requests in FIFO order, (2) Equal-Fair, which splits bandwidth equally across all services that request bandwidth using start-time fair queuing (SFQ) [11], and (3) Popularity-Fair, which also uses a SFQ scheduler, but which divides bandwidth according to the Popularity-based algorithm described above. In this simulation, we assume that the bottleneck in the network is the shared link. Note that our base SFQ scheduler is a work-conserving scheduler: if a service is not able to use its full allocation due to a different bottleneck, the algorithm divides the excess among the remaining services in proportion to their priorities.

To introduce prefetching load, we randomly select 20% of the services and introduce artificial prefetch requests from them at the rate specified on the x-axis. For each demand request, a prefetching service fetches ten objects whose total size is the x-axis value times the size of the demand object. The remainder of the services do not prefetch. The figure plots the performance of the services that do not prefetch. As for the disk space case, we do not assess the effectiveness of prefetching for the services that issue prefetching. Instead, we focus on how excess demand from one service affects other services.

Under FCFS scheduling, prefetching services slow down demand-only services by a factor of 10 and a factor of 2 in the Squid and UCB traces for prefetching rates of 10. In contrast, Equal-Fair is not sensitive to the aggressiveness of the prefetching services. Even though this algorithm does not favor recently accessed services over prefetching services, the fact that



(a) Squid 4MB/s network



(b) UCB 4MB/s network

Figure 4.4: Network response time v. prefetching aggressiveness.

only 20% of our services are prefetching and that they prefetch soon after their demand requests finish limits the amount of damage that prefetching inflicts on other services in this experiment. When there is no prefetching, Popularity-Fair is competitive with the FCFS scheduler. When prefetching by aggressive services increases, however, this increase has almost no affect on the less aggressive services.

4.5 Limitations

One focus of our evaluation is to determine whether the readily observable metric of popularity provides sufficient indication of user priority to serve as a basis for resource management. To make the analysis tractable, our analysis abstracts some important details.

In particular, our strategy of providing one credit per incoming HTTP request represents a simplistic measure of popularity. For example, one might also track the size of the data fetched or the amount of screen real estate the user is devoting to a page representing a service. Other means will also be required for streaming media.

In addition to these simplifications in these simulations, the algorithm itself has several significant limitations.

First, even if our popularity measures perfectly captured user priority, our resource management algorithm emphasizes simplicity over optimality. It could be enhanced in several ways. For example, one might implement a more complete economic model that gives services coins in proportion to their popularity and that allows “the market” to determine the prices of different resources over different time scales. Applications that have a surplus of one resource could then trade rights to that resource for a different one; or applications could follow application-specific strategies to optimize their resource usage (e.g., “my content is not time critical, so wait until 2AM when BW is cheap to prefetch it.”) Developing flexible economies and strategies for competing in them is an open research problem.

Second, our use of the requests from legacy HTTP user agents as a measure of raw popularity makes the system vulnerable to attacks in which legacy client-extension code running at clients (e.g., Java Applets or Javascript) issues requests to the mobile service proxy in the client’s name, thus inflating the apparent popularity of a service. This particular problem could be addressed by having browsers tag each outgoing request with the number of requests issued by a page or its embedded code since the last user interaction with the page; our system would then assign smaller coins to later requests. But this problem illustrates a more fundamental issue: any system that tries to infer priority from user activity provides the opportunity for applications to “game” the system by encouraging activities that will increase resource allocation. We must therefore compromise between simplicity on one hand and precision on the other.

Third, a user's value of a service may not correspond to frequency that the user accesses that service. For example, a user might consider her stock trading service to be her most important service even though she only accesses it once per day to read one page. Although popularity will clearly not capture precise priority, we believe that the heuristic that services a user often uses are likely to be more important than services she seldom uses is a reasonable compromise between simplicity on one hand and precision on the other. Our prototype system provides an "override module" to allow manual adjustment of service priorities if users desire to do so.

Chapter 5

Related work

5.1 Disconnected operations related

In the context of web services, previous studies have examined the performance benefits of web caching [1, 6], prefetching [18, 21, 22, 31], pushing updates [6], mobile code [5, 16, 7], and overlay routing [4]. Systems implementing variations of some of these techniques have been built. File caching [5], replication [43], hoarding [13, 14], and write buffering are standard techniques for coping with disconnection for static file services. Active channels [12] provide an interface for server-directed hoarding. In addition to being limited to static web pages, active channels require user intervention to enable each service, presumably to control servers' use of client resources. Similarly, Microsoft Internet Explorer [45] lets users identify groups of pages to hoard, but users must manually select sites and indicate prefetch frequency and hoard depth. AvantGo [44] for palm-size computers provides a similar interface where users are responsible for resource management; a user can specify a set of web pages (services) which can be downloaded onto the PDA. These methods however have the drawback that they are not very scalable to hundreds of services because they require manual intervention on a per-service basis.

Our work is closely related to the ideas of the Rover [16] and Odyssey [17] for building mobile applications, which uses mobile code, caching, and QRPC to allow applications to access services when disconnected. The differences between the approaches stem from our focus not only on mobility but also disconnections due to network and server failures and our goal of providing an infrastructure for large collections of services. As a result, we focus more of our attention on resource management, provide location independence so that extensions can run at clients, at proxies, and at servers. The strategies we discuss in this paper focus on providing disconnected operation for robustness and mobility and on the resource management problems that arise in such an environment with many mobile services.

5.2 Resource management related

Adaptive resource scheduling is an active research area. However, most proposed approaches are designed for benign environments where applications can be trusted to inform the system of their needs [39] or can be monitored for progress [40]. In our case applications as untrustworthy black boxes and hence we allocate resources based on inferred value from the user rather than stated demand from the applications. The former approach can be more precise and can get better performance in benign environments, but the latter provides safety in environments with aggressive extensions. Noble et. al [17] emphasize *agility*, the speed at which applications and allocations respond to changing resource conditions, as a metric of dynamic resource schedulers. We argue that for stateful resources such as memory and disk, agility must be restrained to match the rate at which the resource may usefully be transferred between applications.

A number of economics-based strategies have been proposed for distributing resources among competing applications in large distributed systems [42, 41]. The D'Agents project [41] is similar to our goal of developing a scalable resource management system with large set of untrusted codes. These systems target more general network topologies than ours and they use secure electronic currency to ration resources.

Chapter 6

Conclusion and Future Work

Disconnected operations can be effectively supported by a mobile service codes which hoard data from the server using techniques like prefetching. In this thesis work, we presented the case for supporting aggressive prefetching. We derived a quantitative trade-off between resources consumed and response time improvement gained by the client and performed a sensitivity analysis for the same. This lead us to believe that current infrastructure has sufficient spare resources to support prefetching of up to 10 times on-demand fetching. The improving technological trends and almost constant human wait-time cost can be interpreted as a justification to support even 100 times more prefetching in the future.

The disconnected system functionality can cause an increased overhead on the three elements of web interactions - client, server and the network. In this report we studied the effect on the clients in detail with an overview of issues at the server and network. An analysis of the workload characteristics at the client revealed a large number of services being accessed which can hog the limited resources at the client and cause resource contention. This motivated the deployment of a robust but flexible resource management system at the client. We presented a novel *popularity based* resource allocation policy which appears to provide good isolation and efficiency with no need for manual tuning.

Our evaluation of our simulations suggest that mobile service code and hoarding techniques hold promise to improve availability, mobility and performance. However, additional work is need to fully understand the implications of all parameters involved. As part of the ongoing research on this topic, we would like to come up with a similar infrastructure support at the server and network. The client resource management system should be extended to handle other resources, such as CPU and memory, to make it a more complete study. The resource management systems need to be more robust and improve the scalability of our virtual machine. More experiments and prototype version for aggressive prefetching should throw more light on the real world usage of the mechanisms proposed in this report.

Bibliography

[1] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. On the scale and performance of cooperative web proxy caching. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, December 1999.

[2] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN Service Availability. In *Proceedings of the Third USENIX Symposium on Internet Technologies and Systems*, March, 2001.

[3] V. Paxson. End-to-end Routing Behavior in the Internet. In *Proceedings of the ACM SIGCOMM '96 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August 1996.

[4] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-end Effects of Internet Path Selection. In *Proceedings of the ACM SIGCOMM '99 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 289–299, September 1999.

[5] P. Cao, J. Zhang, and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proceedings of Middleware 98*, 1998.

[6] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *Proc. of the Nineteenth International Conf. on Distributed Computing Systems*, May, 1999.

[7] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active Naming: Flexible Location and Transport of Wide-Area Resources. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems*, October 1999.

[8] G. Back, W. H. Hsieh, and J. Lepreau. Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, Oct 2000.

[9] G. Czajkowski and T. von Eicken. JRes: A Resource Accounting Interface for Java. In *Proceedings of 1998 ACM OOPSLA Conference*, October 1998.

[10] G. Banga, P. Druschel, and J. Mogul. Resource containers, A new facility for resource management in server systems. In *3rd USENIX Symposium on Operating Systems and Implementation (OSDI)*, New Orleans, LA, Feb. 1999. http://www.cs.rice.edu/˜druschel/osdi99rc.ps.gz.

[11] P. Goyal, H. Vin, and H. Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of the ACM SIGCOMM '96 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 157–168, August 1996.

[12] Active channel technology overview.

http://msdn.microsoft.com/workshop/delivery/channel/overview/overview.asp, 1999.

[13] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[14] G. Kuenning and G. Popek. Automated Hoarding for Mobile Computers. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 264–275, October 1997.

[15] IBM Corporation. Mqseries: An introduction to messaging and queueing. Technical Report GC33-0805-01, IBM Corp., July '95. ftp://ftp.software.ibm.com/software/mqseries/pdf/horaa101.pdf.

[16] A. Joseph, A. deLespinasse, J. Tauber, D. Gifford, and M. Kaashoek. Rover: A Toolkit for Mobile Information Access. In *Proceedings of the Fifteenth ACMSymposium on Operating Systems Principles*, December 1995.

[17] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, October 1997.

[18] D. Duchamp. Prefetching hyperlinks. In Proceedings of the USENIX Symposium on Internet Technologies and Systems, 1999. http://www.usenix.org/events/usits99.

[19] A. Bestavros. Demand-based Document Dissemination to Reduce Traffic and Balance Load in Distributed Information Systems. In *Proceedings of SPDP'95: The 7th IEEE Symposium on Parallel and Distributed Processing*, San Antonio, TA, October 1995.

[20] A. Bestavros, Using speculation to reduce server load and service time on the WWW. In *Proceedings of CIKM'95: The Fourth ACM International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1995.

[21] Z. Jiang and L. Keinrock. Prefetching Links on the WWW. In *ICC '97*, pages 483–489, Montreal, Canada, June 1997.

[22] Z. Wang and J. Crowcroft. Prefetching in World-Wide Web. In *Proc. Global Internet, IEEE*, pages 28-32, November 1996. http://www.cs.ucl.ac.uk/staff/zwang/papers/prefetch/Overview.html.

[23] Differentiated Services (diffserv). http://www.ietf.org/html.charters/diffserv-charter.html.

[24] M. Dahlin, B. Chandra, L. Gao, A. Nayate, A. Khoja, A. Razzaq, and A. Sewani. Resource Management for scalable disconnected access to web services. In *WWW10*, May 2001.

[25] M. Dahlin, B. Chandra, L. Gao, A. Khoja, A. Nayate, A. Razzaq, and A. Sewani. Using Mobile Extensions to Support Disconnected Services. Technical Report TR-2000-20, University of Texas at Austin Department of Computer Sciences, June 2000.

[26] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. The Potential Costs and Benefits of Long-term Prefetching for Content Distribution. In *Web Caching and Content Distribution Workshop*, in review, June 2001.

[27] T. Kelly and D. Reeves. Optimal Web cache sizing: scalable methods for exact solutions. In *Fifth International Web Caching and Content Delivery Workshop*, 22-24 May 2000, Lisbon, Protugal. http://ai.eecs.umich.edu/ ˜tpkelly/papers.

[28] H. R. Varian. Estimating the Demand for Bandwidth. In *Technical Report '99*, August 1999, University of California, Berkeley.

[29] J. K. MacKie-Mason and H. R. Varian. Pricing the Internet. In *Technical Report '93*, April 1993, University of Michigan, Ann Harbor.

[30] E. Brewer and B. Kuszmaul. How to Get Good Performance from the CM5 Data Network. In *Proceedings of the 1994 International Parallel Processing Symposium*, April, 1994.

[31] M. Crovella and P. Barford. The Network Effects of Prefetching. In *Proc. IEEE INFOCOM*, April 1998. http://www.cs.bu.edu/faculty/crovella/paper-archive/infocom98.ps.

[32] J. Gray and P. Shenoy. Rules of Thumb in Data Engineering. In *"Proc. 16th Internat. Conference on Data Engineering"*, pages 3–12, 2000.

[33] J. Chuang and M. Sirbu. Pricing multicast communications: A cost based approach. In *Proc. of the INET'98* 1998.

[34] G. Phillips, S. Shenker, and H. Tangmunarunkit. Scaling of Multicast Trees: Comments on the Chuang-Sirbu Scaling Law In *ACM SIGCOMM '99*, pages 41-51, 1999.

[35] R. Chalmers and K. Almeroth. Modeling the branching characteristics and efficiency gains of global multicast trees. In *INFOCOM '01*, 2001.

[36] Squid sanitized access logs. ftp://ftp.ircache.net/Traces/, April 2000.

[37] A. Iyengar, M. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. In *World Wide Web*, June, 1999.

[38] S. Gribble and E. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[39] M. Jones, D. Rosu, M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, Oct, 1997.

[40] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A Feedback-driven Proportional Allocator for Real-Rate Scheduling In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, Jan, 1999.

[41] J. Bredin, D. Kotz, and D. Rus. Market-based Resource Control for Mobile Agents. In *Autonomous Agents*, May 1998.

[42] Z. Qin, W. Wang, F. Wu, T. Lo, and P. Aoki. Mariposa: A Wide-Area Distributed Database System. In *VLDB Journal*, Vol 5, No 1, pp. 48–63, Jan 1996.

[43] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser". Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the Fifteenth ACMSymposium on Operating Systems Principles*, pp. 172–183, Dec, 1995.

[44] http://www.avantgo.com, February 2001.

[45] Microsoft internet explorer 5. http://www.microsoft.com/windows/ie/default.htm, 2000.

[46] A. M. Odlyzko. The history of communications and its implications for the internet. http://www.research.att.com/˜amo/, June 2000.

[47] A. M. Odlyzko. The economics of the Internet: Utility, utilization, pricing, and Quality of Service. AT&T Labs Research. 1998. http://www.research.att.com/˜amo/doc/complete.html.

[48] Internet Netdoor Inc. Services. http://www.netdoor.com/services/.

[49] V. N. Padmanabhan and R. H. Katz. TCP Fast Start: A Technique For Speeding Up Web Trasfers. In *Proc. Globecom '98 Internet Mini-Conference*, Sysdney, Australia, November 1998.

[50] Rackspace Managed Hosting. http://www.rackspace.com.

[51] J. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, and A. Roetter. Improved Congestion Control for IP Multicast Using Dynamic Layers. Digital Fountain Technical Report, June 2000. Submitted for publication.

[52] J.C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of the ACM SIGCOMM '97 Symposium*, Cannes, France, Sept. 1997.

Vita

Bharat Baddepudi Chandra was born in Hyderabad, India on December 12th, 1977, to Dr. Narasimham Baddepudi and Dr. (Mrs). Lakshmi Rajyam Baddepudi. After completing high school at Little Flower High School and Little Flower Junior College, Abids, Hyderabad, India in 1995, he entered the Indian Institute of Technology, Madras (Chennai), India. He received the Bachelor of Technology in Computer Science and Engineering in May 1999. In August 1999, he joined the graduate program in the Department of Computer Sciences at The University of Texas at Austin. His main research interests are in the fields of Distributed Systems and Operating Systems.

Permanent Address: H.No.11-13-21/2,
Alakapuri,
Hyderabad,
INDIA - 500035.
eHome: bhacha@hotmail.com

This thesis was typeset with $\LaTeX\,2_\varepsilon$[1] by the author.

[1] $\LaTeX\,2_\varepsilon$ is an extension of $\LaTeX$.