INTELLIGENT CACHING MANAGEMENT SYSTEM

Amarjeet Singh
Research Scholar, Uttar Pradesh Rajarshi Tandon Open University, Allahabad, INDIA.
amarjeetsingh_9@rediffmail.com

ABSTRACT
In the upcoming world of Pervasive Computing, users are accessing information sources by a huge variety of mobile devices featuring heterogeneous capabilities. Those devices are attached to the Internet by various communication systems offering different quality of service. The key to meet the demands in this heterogeneous environment is the adaptation of contents to the capabilities of the devices and communication systems. However, content adaptation interferes with the effectiveness of web caching applied in the World Wide Web to improve the performance by avoiding redundant data transfers. Leveraging the advantages of web caching even in the world of Pervasive Computing is the subject of this paper. We outline and discuss an approach that joins the concepts of web caching and content adaptation in a uniform scheme. It is based on hierarchical proxies that do not only cache web objects but even cooperate to adapt the objects to the requirements of the particular clients when they travel down the hierarchy. By our approach the interference between content adaptation and web caching is avoided resulting in improved caching efficiency.

Keywords: Pervasive Computing, proxy caching, hierarchical caching, content adaptation, adaptation paths.

I. INTRODUCTION
Presently, browsing webs and files through the Internet are normal activity of the Internet users. Various services have been provided by the Internet Service Providers (ISPs), such as AT&T, Telstra Big Pond, Lexicon, CAT, etc., in order to offer and maintain their Quality of Services (QoS). One mechanism that has been implemented by every ISP is the installation of the proxy system to serve the Internet users. This mechanism helps reducing the retrieval time and also performs security assurance for the installed organizations. However, the performance of proxy implementation is depended on two factors: the cache size, and the management policy. Through the conversion from mainframe computing to personal computing, the administration of systems shifted from central to distributed management. With mainframes, professionals were responsible for creating and maintaining the single environment that all users accessed. With the advent of personal computing, users got to define their environment by installing any software that fit their fancy. Unfortunately, with this freedom also came the tedious, difficult task of system management purchasing the equipment and software, installing the software, troubleshooting errors, performing upgrades and re-installing operating systems, performing backups, and finally recovering from problems caused by mistakes, viruses and worms. Most users are not professionals and, as such, do not have the wherewithal to maintain systems. As a result, most personal computers are not backed up and not up to date with security patches, leaving users vulnerable to data loss and the Internet vulnerable to worms that can infect millions of computers in minutes. In the home, the challenges have outlined above lead to frustration; in the enterprise, the challenges cost money.

II. FUNCTIONS OF WEB CACHE SYSTEM
The main function of a Web cache System is to store the frequently accessed Web pages locally thereby making Web access faster. It accumulates the requests and sends a single individual request in their place to the destination server. On acquiring the requested data, it forwards it to the requester, making copies of that data within itself. Browsers retrieve portions of data from the cache rather than directly from the server. Thus, Web caches can help lighten the load on a Web server by reducing the number of incoming requests. However, most Web content providers neither have access nor control on which users or how many users arrive at their site. The cache server needs to sit nearer to the user end than to the Web server end. (Web load balancing schemes distribute the incoming load across multiple servers at the Web content provider end.)

In addition to reducing outgoing traffic by bundling duplicate requests from browsers, Web caches act like Web routers, assisting in sending the Web traffic efficiently over a network. While Internet Protocol routing directs low-level traffic irrespective of the data contents, Web routing directs application-specific HTTP traffic across the network. Because Web traffic constitutes most of the Internet traffic, improving Web routing can improve the overall performance of the Internet. Web caching system is similar to memory system caching – a Web cache system stores Web resources in anticipation of future requests. However,
significant differences between memory caching system and Web caching result from the non-uniformity of Web object sizes, retrieval costs, and cacheability. Once the cache server receives a Web request, it checks its database to see if it has the contents of the requested page stored somewhere. A successful retrieval from the local cache is called a cache hit, and an unsuccessful one is called a cache miss. In the case of a cache miss, the request is forwarded to the Web server or the next caching point. This server begins its own access to the requested URL. Such a first-time access to a page forces the cache server to contact the origin Web server that hosts the page. The cache server checks to see if the page can be cached, retrieves the data to cache locally, and, at the same time, passes the contents to the client. The user may never realize that the cache is between the client and server except in special circumstances it is important to distinguish between Web cache system and a proxy server as their functions are often misunderstood. Proxy servers serve as an intermediary to place a firewall between network users and the outside world. A proxy server makes the outgoing network connection more secure, but does little to reduce network traffic. Web caches can help lighten the load on a Web server by reducing the number of incoming requests, by storing frequently accessed pages.

The caching technique or architecture usually used on a proxy server is proxy caching. But this is not a must, meaning a proxy server can be using a transparent caching mechanism, as will be explained later. The most obvious beneficiary of Web caching is the user, who avoids some traffic snarls when browsing. The network administrator and the remote Web site also benefit. According to the National Laboratory for Applied Network Research (NLANR), large caches with lots of clients may field as many as 50% of the hits that would otherwise travel through a network individually to the origin site. A typical cache could easily field about 30% of the intended hits. Thus, statistically speaking, a Web cache could eliminate at least 30% of the Web traffic that would normally be going out over a wide area network (WAN).

The bandwidth between institutional networks and regional ISP (Internet Service Provider) and that between transoceanic and regional networks plays a vital part in the Web traffic. Web caching reduces this bandwidth consumption, thereby decreases network traffic and lessens network congestion. Overloaded Web servers and congested exchange points are the main reason for Web latency. Caching reduces access latency due to two reasons:

- **Frequently accessed documents are fetched from nearby cache servers instead of remote data servers, minimizing the transmission delay.**
- **Because of the reduction in network traffic, those documents not fetched can also be retrieved relatively faster than without caching due to less congestion along the path and less workload at the server.**

In Figure 1 a possible chain of caches in the WWW through which a request and responses might flow. Starting in a browser, Web request can travel through multiple caching systems on its way to the origin server. At any point in the sequence a response can be sent if the request matches a valid copy of the requested data in the cache.

### III. CACHE SIZING

Informally, the cache sizing problem is to determine the optimal size of an existing cache, i.e., a storage capacity that minimizes the total cost of serving requests submitted to the cache.

In general, both $M(s)$ and $A(s)$ are monotonic step functions. Note that minimal total cost need not occur at a single cache size, and those local minima the relative

![Figure 1: Caching Management System Architecture](image)

![Figure 2: Cache Cost Functions](image)
popularity of each document. A probabilistic representation ignores temporal locality and other workload details, greatly simplifying the problem; If workload is given as an explicit reference sequence, the difficult part of the cache sizing problem is computing aggregate miss cost as a function of cache size $s_{\text{min}}$. Designers are sometimes given performance constraints, and cache size is one of many parameters that must be chosen in such a way as to satisfy them. It is possible, for instance, that a certain minimal storage capacity $s_{\text{min}} > s^*$ is required to achieve hit rates high enough to satisfy a mean latency target.

The correct procedure is therefore to determine the minimal cache size $s_{\text{min}}$ required to satisfy all performance constraints, and finally choose the larger of $s_{\text{min}}$ and $s^*$. (Here we assume that additional cache hits resulting from choosing an optimal size $s^* > s_{\text{min}}$ will not cause performance constraints to be violated. This is a reasonable assumption; cache misses nearly always require more time and computational resources than hits.) In other words, if we are given exogenous performance constraints, our problem is one of optimal cache expansion rather than optimal cache sizing.

### 3.1 Cache Installation

The question of whether a cache should be installed at a given location must be answered before we consider optimal cache sizing. However, given an expected workload and a method for computing aggregate miss cost as a function of cache size $S_A(s)$, it is straightforward to decide whether a cache is economically justifiable: Installation entails some fixed cost $S_F$ in addition to the cost of storage $S_D(s)$. The cost of not installing a cache is $S_A(0)$, and the cost of installing a cache of optimal size $s^*$ is $S_F + S_A(s^*) + S_D(s^*)$.

We simply choose the alternative with lower cost. As in the cache sizing problem, the difficult part is computing $S_A(s)$ based on workload. So far we have ignored interactions between different caches’ workloads, e.g., the impact of browser caches on the workload that may exist in the total cost function. Finally, note that total cost increases monotonically for workload in one of two ways: as an explicit sequence of references (a trace), or in a probabilistic form that is tractable because careful studies of intermediate caching servers in distributed file systems have concluded that under some conditions such caches can degrade some performance metrics, e.g., latency [121,122]. As stated previously, throughout this dissertation we assume that cache hits are always preferable to cache misses; miss costs represent the additional penalty we incur from cache misses, and are nonnegative.

### 3.2 Removal Policies

Given that a cache has been installed and its capacity is fixed, the remaining problem is how best to serve its workload. If workload is represented probabilistically, this is a straightforward task due to the assumption of independent references: The cache must solve a classic knapsack problem, storing a subset of data payloads with maximal popularity weighted miss cost subject to a capacity constraint. Therefore when considering the cache service problem we shall restrict attention to the case where workload is represented as an explicit trace. A cache removal policy should strive to minimize the aggregate cost of processing all requests, i.e., the sum over all cache misses of miss cost.

Alternatively, we might speak of the value of cache hits rather than the cost of cache misses, and say that a cache should maximize value, perhaps by preferentially storing the most valuable documents. The two perspectives—cost minimization and value maximization—are substantively equivalent but differ in connotation, and in some cases we shall adopt the latter view. In particular, the “value” perspective is more natural in situations where miss costs are supplied to a cache by system users, e.g., servers and clients.

### 3.3 Redundant Transfers

Web transactions involve requests containing names (URLs) that elicit replies containing data payloads. Content providers define the relationship between URLs and reply payloads, and this relationship is neither simple nor stable: Identical URLs can yield different reply payloads and different URLs can yield identical payloads. We refer to these phenomena as resource modification and aliasing, respectively.

Traditional Web caches use URLs to organize and locate stored data, i.e., cached reply payloads are associated with, and accessed via, the URL that yielded them. Content naming practices at the server end can interact with client request patterns in such a way as to cause redundant payload transfers in conventional “URL-indexed” caches. Aliasing, for instance, can cause redundant transfers when a conventional cache already holds the payload needed to satisfy the current request, but not in association with the current request URL. These observations suggest that URL-indexed caching may be poorly suited to Web workloads.

### 3.4 Cache Consistency

To remain semantically transparent, caches must serve the same payload as the origin server would at the time they process requests. Consecutive accesses to the same URL sometimes yield different reply payloads, and URL-indexed caches therefore require
some mechanism to determine whether a payload cached in association with the current request URL is fresh, i.e., is the same as the origin server would return. The statelessness constraint discussed earlier precludes invalidation-based consistency mechanisms in which servers track cache contents and explicitly instruct caches to discard stale entries. Remaining alternatives include expiration, in which reply metadata specifies a time beyond which the reply data should not be considered fresh, and revalidation, in which caches verify freshness by contacting the origin server.

In practice, the freshness policies of today’s Web caches employ a combination of the two, serving requests from cache if a fresh cache entry is available for the current request URL and revalidating if an entry exists but is stale. Reply metadata may specify an absolute expiration time or an age limit for cache entries; if origin servers provide no such metadata, the cache freshness policy will typically compute an estimated time-to-live heuristically.

Revalidations may ask whether a resource has changed since it was retrieved from the Origin server or they may compare the entity tags of the cached resource with the origin server’s current view of the resource. Entity tags (“Etags”) are a kind of opaque, unordered version identifier that origin servers associate with payloads; matching Etags imply identical payloads.

Some URLs correspond to simple static files residing on disk at the server. Others, however, invoke scripts, programs, or database queries whose output is often termed “dynamic content.” Similarly, replies are sometimes customized for individual users using mechanisms such as “cookies”. Origin servers may explicitly mark customized and dynamic replies “uncachable,” or they may instruct caches to revalidate the cached payload each time it is used, thus saving bandwidth while preserving semantic transparency.

IV. PRACTICAL IMPLEMENTATIONS

We have developed a prototype based on this model which is called integrated system. I have extended our system and rewritten parts of it several times. The system that are describing, is the culmination of many months of experience. This chapter presents the design and justification for cache-based system management, as well as the detailed design and implementation of the Integrated.

4.1 Maintaining Data

• a Windows XP environment. Over the course of half a year, the Windows appliance has gone through two service packs and many security updates. The appliance initially contained Office 2000 and was upgraded to Office 2003. The appliance includes a large number of applications such as Adobe Photo-Shop, Frame Maker, and Macromedia Dream Weaver.

• a Linux environment, based on Red Hat’s Fedora Core, that uses NFS to access our home directories on our group file server. Over a period of eight months, the NFS Linux appliance required many security updates, which replaced major subsystems like the kernel and X server. Software was added to the NFS Linux appliance as it was found to be needed.

<table>
<thead>
<tr>
<th>Name of Disk</th>
<th>VAT Startup</th>
<th>Windows Reboot</th>
<th>Linux Reboot</th>
<th>Kernel Build</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexar 1 GB Flash Drive</td>
<td>53</td>
<td>129</td>
<td>42</td>
<td>455</td>
</tr>
<tr>
<td>IBM 4 GB Drive</td>
<td>65</td>
<td>158</td>
<td>53</td>
<td>523</td>
</tr>
<tr>
<td>Hitachi 40 GB Drive</td>
<td>61</td>
<td>84</td>
<td>43</td>
<td>457</td>
</tr>
<tr>
<td>Fujitsu 60 GB Drive</td>
<td>52</td>
<td>65</td>
<td>40</td>
<td>446</td>
</tr>
</tbody>
</table>

Table 1: Performance Characteristics of Four Different VAT Disks

• a Linux environment stores the user’s home directory in a user disk. This Linux appliance included all the programs that came with the distribution and was therefore much larger.

It also measures the size of all the COW disks for each appliance and the size of the latest version. The last column of the table, “Cache size”, shows an example of the cache size of an active user of each appliance. It observes from our usage that the cache size grows quickly and stabilizes within a short amount of time. It grows whenever major system updates are performed and when new applications are used for the first time. The sizes shown here represent all the blocks ever cached and may include disk blocks that may have since been made obsolete.

4.2 Effectiveness of Prefetching

In the following, first measure the access profile to establish that prefetching a small amount of data is useful. Second, measure the effect of prefetching on the performance of an interactive application.

4.2.1 Access Profile

In this experiment, measure the access profile of appliance blocks, to understand the effectiveness of prefetching based on the popularity of blocks. It took 15 days of usage traces from 9 users using the three appliances described above in their daily work. Note that during this period some of the appliances were updated, so the total size of data accessed was greater than the size of a single active version. For example, the Windows XP appliance had an active size of 4.5 GB and seven updates of 4.4 GB combined, for a total of 8.9 GB of accessible appliance data.

Table3.3 shows each appliance’s effective size, the size of all the accesses to the appliance in the trace, and the size of unique accesses. The results suggest that only a fraction of the appliance data is ever accessed by any user. In this trace, users access only 10 to 30% of the accessible data in the appliances.
Table 2: Statistics of Appliances in the Trace

<table>
<thead>
<tr>
<th>Appliance</th>
<th>Accessible Size</th>
<th>Accesses in Trace</th>
<th>Unique in Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows XP</td>
<td>8.9 GB</td>
<td>31.1 GB</td>
<td>2.4 GB</td>
</tr>
<tr>
<td>NFS Linux</td>
<td>3.4 GB</td>
<td>6.8 GB</td>
<td>1.0 GB</td>
</tr>
<tr>
<td>User-Disk Linux</td>
<td>6 GB</td>
<td>5.9 GB</td>
<td>0.5 GB</td>
</tr>
</tbody>
</table>

Figure 2 shows the percentage of accesses that are satisfied by the cache (Y-axis) if a given percentage of the most popular blocks are cached. The results show that a large fraction of data accesses are to a small fraction of the data. For example, more than 75% of data accesses in the Windows XP appliance are to less than 20% of the accessed data, which is about 5% of the total appliance size. These preliminary results suggest that popularity of accessed appliance data is a good heuristic for prefetching, and that prefetching a small fraction of the appliance’s data can significantly reduce the chances of a cache miss.

4.2.2 Interactive Performance

The responsiveness of an interactive application can be severely affected by cache miss delays. To simulate interactive workloads, created a VNC recorder to record user mouse and keyboard input events, and a VNC player to play them back to reproduce user’s actions. Using VNC provides us with a platform-independent mechanism for interacting with the desktop environment. Furthermore, it allows us to use VMware’s built-in VNC interface to the virtual machine console.

Other tools try to do this, but play back is not always correct when the system is running significantly slower than during recording. This is especially true for mouse click events. To reliably replay user actions, our VNC recorder takes screen snapshots along with mouse click events. Our replay works only on systems with little or no nondeterministic behaviour. Since used virtual machines, it can easily ensure that the initial state is the same for each experiment. Here used the Windows XP appliance to record a VNC session of a user creating a PowerPoint presentation for approximately 8 minutes in a LAN environment. This session is then replayed in the following experimental configurations:

- **Local**: the entire appliance VM is copied to the VAT disk and executed with un-modified VMware, without demand-fetching or caching.
- **Prefetched**: some of the virtual machine’s blocks are prefetched into the VAT’s cache, and the VM is then executed on top of that cache.

Prefetching measures the amount of data transferred over the network; due to compression, the amount of raw disk data transferred is approximately 1.6 times more. The amount of prefetching goes up to a maximum of 420 MB, which includes all of the blocks accessed in the appliance by our users. The total runtimes for the replayed sessions are within approximately 5% of each other – the additional latency imposed by demand-fetching disk blocks over DSL is absorbed by long periods of user think time when the system is otherwise idle. To make a meaningful comparison of the results, measure the response time latency for each mouse click event, and plot the distribution of response times over the entire workload. For low response times, the curves are virtually indistinguishable.

This region of the graph corresponds to events that do not result in any disk access, and hence are quick in all the scenarios. As response time increases, the curves diverge; this corresponds to events which involve accessing disk – the system takes noticeably longer to respond in this case, when disk blocks need to be demand-fetched over the network. The figure shows that PowerPoint running in the Integrated is as responsive as running in a local VM, except for times when new features have to be loaded from disk – similar to Windows taking a while to start any given application for the first time.

The most commonly accessed blocks are those used in the bootup process. This experiment only measures the time taken to complete the PowerPoint workload after the system has been booted up, and therefore the benefit of prefetching the startup blocks is not apparent in the results shown in the figure. However, prefetching the startup blocks improves startup time from 391 seconds in the no prefetching case to 127 seconds when 200 MB of data is prefetched.

The results show that prefetching improves interactive performance. In the case of full prefetching, the performance matches that of a local VM. Partial prefetching is also beneficial – it can see that prefetching 200 MB significantly improves the interactive performance of PowerPoint.

4.2.3 Feasibility of Online Backup
In our system, user data should always be backed up onto network storage. To determine whether online backup works for real workloads, collected usage traces for three weeks on personal computers of ten users running Windows XP. These users included office workers, home users, and graduate students. The traces contain information on disk block reads and writes, file opens and start and end of processes. Here also monitored idle times of keyboard and mouse; assume the user to be idle if the idle time exceeds five minutes.

![Figure 4: CDF Plot of Response Times for Different Levels of Prefetching](image)

In our system the user would log out and possibly shut down his VAT soon after he completes his work. So, the measures are interested in is whether there is any data that is not backed up when he becomes idle. If all the data is backed up, then the user can log in from any other VAT and get his most recent user data; if the user uses a portable VAT, he could lose it with no bad effects. To quantify this measure simulated the usage traces on our cache. To perform the simulation divided the disk writes from the usage data into writes to system data, user data, and ephemeral data. These correspond to the system disk, user disk, and ephemeral disk that were discussed earlier. System data consists of the writes that are done in the normal course by an operating system that need not be backed up.

Examples of this include paging; defragmentation, NTFS metadata updates to system disk, and virus scans. User data consists of the data that the user would want to be backed up. This includes email documents, office documents, etc., It categorize internet browser cache, and media objects such as mp3 files, that are downloaded from the web as ephemeral data and do not consider them for backup. In our traces there were a total of about 300GB worth of writes of which about 3.3% were to user data, 3.4% were to ephemeral data and the rest to program data.

Users were idle 1232 times in the trace, and in our simulation, backup stops during idle periods. Now estimate the size of dirty data in the cache when users become idle. The x-axis shows the size of data that is not backed up, and the y-axis shows the percentage of idle periods. From the figure, most of the time there is very little data to be backed up by the time the user becomes idle. This suggests that interactive users have large amounts of think time and generate little backup traffic.

This also shows that online backup, as implemented in the Collective, works well even on a DSL link. Even in the worst case, the size of dirty data is only about 35 MB, which takes less than 15 minutes to backup on DSL.

![Figure 5: Fraction of Times Where Dirty Data in Cache at the end of Session](image)

The results presented in this section illustrate that the system performs well over different network connections, and that it provides a good interactive user experience. Further, the results support our use of prefetching for reducing cache misses, and shows that continuous backup is feasible for most users.

V. EXPERIENCES

Caching Management System could be done at various locations and network points: near the content consumer (consumer-oriented), near the content provider (provider-oriented), and at strategic points in the network, based on user access patterns and network topology. Positioning caches near the client, as in proxy caching has the advantage of leveraging one or more caches to a user community. If those users tend to access the same kind of content, this placement strategy improves response time by being able to serve requests locally.

Caches positioned near or maintained by the content provider, on the other hand, improve access to a logical set of content. This type of cache deployment can be critical to delay-sensitive content such as audio or video. Positioning caches near or on behalf of the content provider allows that provider to improve the scalability and availability of content.

The use of both consumer-oriented and provider-oriented caching techniques is perhaps the most powerful and effective approach, since it combines the advantages of both while lowering the disadvantages of each. The last approach is the dynamic deployment of caches at network choke points. Although it would seem to provide the most flexible type of cache coverage, it is still a work in progress and, to the best of the authors’ knowledge, there have not been any performance studies demonstrating its benefits. The dynamic deployment technique also raises important questions about the administrative control of these caches.
VI. CONCLUSIONS

As it is with any evolving technology, caching management system is changing rapidly, especially being of interest to both the research universities and the industry. As mentioned earlier, we have made an attempt to capture the state-of-art schemes and methodologies involved. We present the integrated system, a prototype of a caching management system and its architecture for managing desktop computers. This chapter concentrates on the design issues of a complete system. Caching in the Integrated helps provide good interactive performance even over wide-area networks. Our experience and the experimental data gathered on the system suggest that the integrated system management architecture can provide a practical solution to the complex problem of system management. With the fair use of Caching Management System we can lessen the burden on web servers, improve its performance and at the same time reduce the network traffic.

REFERENCES