AT&T Labs Research

subject: **Efficient Algorithms for Predicting Requests to Web Servers**

date: **December 21, 1998**

from: **Edith Cohen**
Dept. HA6171000
FP A019
973-360-8709
edith@research.att.com
HA6171000-981221-01TM

Balachander Krishnamurthy
Dept. HA6163000
FP D229
973-360-8648
bala@research.att.com
HA6163000-981221-01TM

Jennifer Rexford
Dept. HA6173000
FP A169
973-360-8728
jrex@research.att.com
HA6173000-981221-01TM

TECHNICAL MEMORANDUM

Internet traffic has grown significantly with the popularity of the Web; consequently user perceived latency in retrieving web pages has increased. Caching and prefetching at the client side aided by hints from the server are attempts at solving this problem. We suggest techniques to group resources that are likely to be accessed together into *volumes*, which are used to generate hints tailored to individual applications, such as prefetching, cache replacement, and cache validation. We discuss theoretical aspects of optimal volume construction, and develop efficient heuristics. Tunable parameters allow our algorithms to predict as many accesses as possible while reducing false predictions and limiting the size of hints. We analyze a collection of large server logs, extracting access patterns to construct and evaluate volumes. We examine sampling techniques to process only portions of the server logs while constructing equally good volumes. We show that it is possible to predict requests at low cost with a high degree of precision.

1 Introduction

The popularity of the World Wide Web is largely responsible for the dramatic increase in Internet traffic during the past few years. A growing number of users are accessing "Web pages," each consisting of one or more resources, including text, images, and applets. These resources must be retrieved from the Web server and transferred across the network, unless the contents are available in the browser cache or at a shared proxy cache. Retrieving a resource typically requires the browser to establish a TCP connection and generate a request with the appropriate URL. The server transmits a response message with the contents of the resource. The process of opening the TCP connection, submitting the HTTP request, and transmitting the HTTP response introduces perceivable latency and consumes bandwidth along the path from the server to the client. By storing popular resources closer to the client sites, proxy caches can reduce response latency, server load, and network traffic. Proxies can also perform other performance-enhancing functions, such as prefetching, cache validation, and TCP connection management, on behalf of its clients.

The proxy's policies for managing cached resources and TCP connections rely on assumptions about client access patterns. For example, cache replacement strategies try to identify which resources are least likely to be accessed in the near future. Similarly, prefetching decisions are based on accurate estimates of future requests, to avoid burdening the server and the network with extra transmissions. The server, by receiving requests from a large number of proxies and clients, is in a unique position to gauge the popularity of its resources, and the likelihood that certain resources are accessed together. For example, the server can speculatively disseminate resources to the proxy [1] , guide proxy prefetching decisions [2–5] , improve cache coherency by sending information about modified resources [6,7] . The overhead can be minimized by focusing on the resources that are most likely to be requested in the future. Proxy's cache replacement policy can be enhanced with estimates of future accesses [8,9] .

In earlier work, we introduced an end-to-end framework for server response messages to include hints (in groups called *volumes*) about future requests, customized to the requesting proxy [10] . Here, we present a detailed design and evaluation of algorithms for grouping resources that are likely to be accessed together. In contrast to previous research on server hints, we discuss the theoretical aspects of optimal volume construction, and develop efficient heuristics that can be tuned to specific proxy applications. The algorithms estimate which resources will be requested within a time interval in the future. Tunable parameters allow the algorithms to strike a balance between the desire to predict as many accesses as possible with the need to control the number of false predictions, while limiting the number of hints that are sent to the proxy. We also consider simple techniques for reducing the computational overheads for constructing and evaluating the server volumes.

The key contributions of the paper are:

- **Problem formulation:** We present a precise theoretical formulation of the volume-construction problem, based on the time interval for predicting future access and three important performance metrics. Based on this formulation, we describe several optimization problems that require balancing the trade-offs between the number of accesses that are predicted in advance, the number of false predictions, and the total number of hints sent to the proxy.

- **Greedy heuristic:** After proving that the optimization problems are NP-complete, we present a greedy algorithm for constructing volumes. The greedy volumes are constructed using a notion of marginal utility of adding a resource to a volume given the current set of volumes, until an overall threshold utility value is reached. Although the greedy volumes perform very well, volume construction incurs a significant computational overhead. Hence, we consider faster alternatives such as *pairwise volumes*.

- **Pairwise volumes:** Expanding on our earlier paper [10] , we present a detailed treatment of heuristics for constructing volumes, based on the likelihood that pairs of resources are accessed together. Similar to previous work [1,2,5] , we define a probability threshold for deciding which resources to include in a volume. We describe an efficient algorithm for computing access probabilities, and elaborate on several techniques for identifying ineffective hints that do not usually generate new predictions.
- **Multifactor experiments:** To evaluate the volume-construction techniques, we characterize the interaction between the various parameters and performance metrics on a collection of server logs. One set of logs are from AT&T's EW3 Web hosting service and the rest were from past experiments [10] . The experiments provide guidelines for setting the algorithm parameters for different proxy applications. We also show that our approach of computing access probabilities and removing ineffective predictions is competitive with the more complex greedy heuristic. The pairwise volumes were also used as a seed for the greedy volume construction to reduce the overhead of the greedy algorithm.
- **Reduction of processing requirements:** Processing large server logs requires significant computational resources. Through simulation experiments, we show that it is possible to substantially reduce the amount of data processing involved in generating volumes. The overheads of constructing and evaluating volumes can be reduced by operating on samples of the server logs. Generating volumes from a shorter duration of the log, or a fraction of the clients/proxies, is competitive with computing across the entire collection of accesses.
- **Evaluation methodology:** We describe our software used to repair and remove inconsistent entries in the server logs, generate an anonymized intermediate format for efficient post-processing, and for building volumes and computing performance metrics in a semi-automated fashion.

The rest of the paper is as follows: Section 2 describes the volume construction problem and its key parameters. Section 3 presents an efficient algorithm for computing the likelihood that pairs of resources are accessed together, including techniques for improving the effectiveness of the predictions and reducing computational complexity. The optimization problems being NP-complete, we present a greedy heuristic for computing volumes from server logs. Section 4 describes the logs used to evaluate our algorithms. The simulation experiments on server logs in Section 5 systematically characterize the interaction between the various parameters, and investigate ways to reduce the amount of data that must be processed to construct and evaluate server volumes. Section 6 concludes the paper with a discussion of future research directions. The NP-completeness proof of optimal volume construction, the theoretically-motivated greedy heuristic's performance guarantees and the performance of the pairwise heuristic are in Appendices A, B, and C.

2 Volume Construction Problem

In this section we describe our prediction model of future requests and discuss the various optimization criteria of interest. The greedy algorithm developed to obtain an approximate solution is then presented.

2.1 Prediction Model

Studies of Web access patterns have shown that recent requests are useful for predicting which resources will be accessed in the near future. The proxy-server exchange of request and response can be enhanced by the server predicting resources (and their attributes, such as Last-Modified-Time, size, etc.,) the client will access in the future. Many of these accesses stem from navigating through the hypertext links of recently-retrieved Web pages. We consider techniques that generate hints based only on the current request rather than maintaining detailed information about past accesses, since the large number of requesting proxies and contacted servers would introduce substantial overhead.

Instead, an offline computation can determine the relationship between various server resources, based on information available in the request logs. The server has the history of past accesses including the triple (s, a, t), where client s requests resource a at time t. This information can be extracted from the server logs and post-processed to compute associations between resources. To reduce the post-processing overhead, it is possible to perform these computations on a subset of clients and/or a smaller time interval in the logs. Volumes can be recomputed periodically to ensure that the server generates hints based on recent information from the logs. Each resource a has a *volume* v_a, which is a list of resources that are likely to be requested in the near future after an access to a, along with some resource attributes (such as Last-Modification-Time of the resource). Effective and efficient algorithms for computing the set of volumes $\{v_a\}$ based on the sequence of triples (s, a, t) are needed. This basic formulation has guided a number of recent studies of Web prefetching [1, 2, 5] .

As proxies request resources from the server across time, they receive a sequence of hints for future accesses. A prediction of an access to a resource b is most useful if it occurs some minimum time τ and maximum time T before the request. If the prediction is generated too close to the predicted access (e.g., less than half a second in advance), the proxy does not have sufficient time to act on the server hint. For example, a proxy is likely to begin downloading embedded images immediately upon receiving the enclosing page, lowering the value of predictions for these resources. On the other hand, predictions that arrive too far in advance (e.g., one hour before the predicted access) may also be of limited value. In practice, it may be difficult to predict accesses very far into the future, since client requests often depend on recent history. Summarizing, the two key parameters are:

- *Minimum time interval (τ):* The minimum time interval that an access to a can predict an access to b.
- *Maximum time interval (T):* The maximum time interval that an access to a can predict an access to b.

Hence, we consider that an access to resource b at time t was *predicted* if the proxy receives at least one hint for b during the time interval $[t-T, t-\tau)$. Similarly, a hint generated at time t for resource b is a *true prediction* if and only if the proxy receives at least one request for b during $[t+\tau, t+T)$.

2.2 Optimization Criteria

The hints returned to the proxy should maximize the proportion of requests that are predicted in advance, while minimizing the proportion of hints that are incorrect. A resource that appears in multiple hints in the same time interval is treated as a single prediction. To reduce the overhead of transmitting hints and the receiving proxy processing them, the server should minimize the total number of hints. We consider three key performance metrics, the first two of which are borrowed from the Information Retrieval field.

- *Recall:* The likelihood that a resource requested by the client has appeared in at least one hint in the last $[\tau, T)$ seconds.

- *Precision:* The likelihood that a resource that appears in a hint will be accessed by the client in the next $[\tau, T)$ seconds.
- *Average Hintsize:* The average number of hints in a message from the server to the client.

These criteria suggest a number of possible optimization problems, including:

1. Maximize recall, subject to an upper bound on the average hintsize
2. Minimize hintsize, subject to a lower bound on recall
3. Maximize recall, subject to a lower bound on precision
4. Maximize precision, subject to a lower bound on recall

Optimal volumes according to the first two criteria generate tradeoffs where recall increases with hintsize. Optimal volumes constructed according to the last two criteria are such that precision increases as recall drops. These desirable properties, consistent with user's expectations, are guaranteed for greedy volumes and as we shall see in Section 5, they generally hold for other heuristics.

The choice of an optimization criterion depends on how the proxy uses the server hints [10] . For example, a prefetching application may require high precision to avoid the high cost of incorrect predictions. In contrast, a cache coherency policy may demand high recall, to receive the Last-Modification-Time information for a large portion of the cached resources, even if the hints include excess information.

These optimization problems are NP-complete; proof of reduction to 3-SAT is in Appendix A. For efficient construction of server volumes, we offer heuristics based on computing the pairwise probability that an access to a resource a is followed by an access to resource b. The basic pairwise heuristic includes b in v_a whenever the implication probability exceeds a threshold (see Section 3). The more refined and theoretically-motivated greedy algorithm iteratively recomputes these probabilities, based on past decisions (Section 2.3). The greedy algorithm always performs within factor of $O(\log n)$ from optimal where n is the number of requests (the performance ratio is explained in Appendix B). The pairwise heuristic has performance ratio of $\Theta(n)$ on worst case sequences. In practice, the greedy algorithm outperforms the pairwise heuristic, but both techniques can achieve high precision and high recall with a reasonable average hintsize, as shown in Section 5. Greedy, however, incurs significant computational overhead; hence the need for the pairwise heuristic.

2.3 Greedy Algorithm

The greedy heuristic constructs volumes by iteratively augmenting the sets $\{v_a\}$, based on the utility of the remaining resources. Each iteration performs a single pass on the server log to evaluate the *marginal utility* $u(b, a)$ of adding resource b to volume v_a, given the current set of volumes. The definition of $u(b, a)$ depends on the optimization criteria. For example, for the recall vs. hintsize trade-offs:

$$u(b, a) = \frac{\text{number of additional predicted requests for } b}{\text{total number of requests for } a}$$

captures the ratio of "benefit" (true predictions) and "cost" (total predictions), and has provable performance properties. Similarly, for the recall vs. precision trade-offs:

$$u(b, a) = \frac{\text{number of additional predicted requests for } b}{\text{additional prediction time of } b}$$

Input:
A threshold U
For each resource b, a set $\texttt{candidates}(b)$

Iterate:
Compute $u(b, a)$ for all b and $a \in \texttt{candidates}(b)$
For every resource b:
Let $a \in \texttt{candidates}(b)$ be the resource that maximizes $u(b, a)$.
If $u(b, a) \geq U$, $v_a \leftarrow v_a \cup \{b\}$
Remove from $\texttt{candidates}(b)$ all a with $u(b, a) < U$.
Until: $\forall b$, $\texttt{candidates}(b) = \emptyset$

Figure 1: Greedy Heuristic

where the *prediction time* of b is the proportion of time with at least one *active* prediction for b. A prediction for a resource is considered to be active for a time interval if it was in a volume of a resource that was requested at least τ and at most T time units ago.

Figure 1 provides a description of the greedy algorithm. Volumes are constructed subject to a threshold U on the marginal utilities. At the end of each iteration, for each resource b, we find the largest value of $u(b, a)$, and if $u(b, a) \geq U$, we add b to v_a. If $u(b, a) < U$ for all pairs (b, a), the algorithm terminates. The algorithm starts with a set of empty volumes. In each iteration, each resource b is added to at most one volume v_a. Each item in v_a is labeled with the respective marginal utility. For each resource b, we maintain a set $\texttt{candidates}(b)$ of *candidate predictors*, consisting of all resources a such that we consider placing b in v_a. The running time of the greedy heuristic hinges on the number of iterations, which is bounded by the size of the largest candidates set (which is at most the number of resources).

We now discuss two techniques used to obtain an efficient implementation of greedy. The efficiency of the algorithm can be improved by limiting the size of the initial candidate set. A straightforward initialization places *all* resources in each of the sets $\texttt{candidates}(b)$, resulting in a quadratic number of candidates. We carefully initialize by including only pairs with initial $u(b, a) \geq U$; such pairs are found efficiently using our techniques for constructing pairwise volumes (see Section 3). It is also possible to remove resources from the candidate set during iterations of the greedy algorithm. The marginal utility $u(b, a)$ is non-increasing with respect to the first optimization criteria. Hence, resource a can be removed from $\texttt{candidates}(b)$ as soon as $u(b, a) < U$.

3 Constructing Pairwise Probabilities

The greedy heuristic showed a way to construct high quality volumes although at potentially high computation cost. We now discuss the cheaper pairwise approach. The server can generate predictions by estimating the likelihood that resources are accessed together. After describing how these access probabilities are computed, we introduce heuristics to improve accuracy and reduce volume size by determining which predictions are most effective.

3.1 Pairwise Counters

The complexity of the greedy algorithm stems from evaluating the utility of putting resource b in volume v_a, based on the other volumes that already include b. To avoid this overhead, we construct volumes based on the probability that resources a and b are accessed together, independent of the predictions generated by other resources. We define $p_{b|a}$ as the proportion of requests for resource a that are followed by a request for resource b by the same client within at least T seconds and after τ seconds. Resource b is included in volume v_a if $p_{b|a}$ is greater than or equal to a threshold probability p_t. The value of the threshold probability depends on the desired trade-off between precision, recall, and hintsize. By processing a log of accesses the server accumulates counters of the number of accesses to each resource, and the number of times a pair of resources occurs together. The key parameter and counter variables are:

- *Threshold probability (p_t):* The minimum value of $p_{b|a}$ for including b as a hint after a request for a
- *Pairwise counters ($c_{b|a}$)*: The number of accesses to b that occur within at least T seconds and after τ seconds of an access to a
- *Individual counters (c_a)*: The number of accesses to resource a

where $p_{b|a} = c_{b|a}/c_a$.

Nuances arise in handling repeated accesses to resources a and b within a small interval of time. For example, when resource b is requested multiple times after an access to resource a (e.g., $abbb$), we count this as a single occurrence of the pair of resources. That is, although the counter c_b increases with each access to b, the counter $c_{b|a}$ is incremented just once. Otherwise, the ratio $c_{b|a}/c_a$ could be larger than 1. Counting predictions more than once is somewhat misleading, as it can artificially inflate the probability $p_{b|a}$. For example, if one request for a is followed by three requests for b, and two other requests for a are not followed by requests for b, we assign $p_{b|a}$ to 1/3, not 3/3. This is crucial for the precision metric, since it accounts for false predictions generated on requests for a not followed by b. Repeat accesses to resource a (e.g., $aaab$) are handled differently. We credit each occurrence of a with generating a prediction, as long as b occurs within the subsequent prediction interval (τ, T). For example, if three accesses to a are followed by a single access to b, we assign $p_{b|a}$ to 3/3, not 1/3.

Related work has been similar: [1] computes $p_{b|a}$ such that multiple occurrences of b after a is credited just once. [2] gives multiple credit for such a pattern but conditional probabilities are not exploited. In [5] , the counters are incremented only if there is a hyperlink from resource a to b and only when b is accessed immediately after a.

The counters $c_{b|a}$ and c_a can be computed by sequencing through the requests in the server log. The probabilities are computed by sequencing through the triples (s, a, t), where client s requests resource a at time t. The algorithm processes the requests one client at a time. For a given client, we maintain a FIFO list of requests from the preceding T seconds. In processing a request for b, we first remove requests that occurred more than T seconds ago from the head of the FIFO. Then, we sequence from the tail of the list and increment $c_{b|a}$ for each resource a in the list, after skipping any requests that occurred less than τ seconds ago. Sequencing continues till the end of the list, unless we encounter a previous request for resource b. Else, the algorithm would credit some resource a with multiple predictions of resource b in a single time interval. Suppose a client has a sequence of requests for $axbyb$ in a small interval of time. The first request for b increments counters $c_{b|a}$ and $c_{b|x}$; the second access to b increments $c_{b|y}$, but not $c_{b|a}$ and $c_{b|x}$.

3.2 Reducing the Number of Counters

Computing the pairwise probabilities $p_{b|a}$ could potentially require a quadratic number of counters $c_{b|a}$, though many pairs of resources rarely occur together. Complexity can be reduced using heuristics that create counters only for pairs of resources that are likely to be accessed together. If the counter $c_{b|a}$ does not exist, the algorithm would not accumulate statistics for recent accesses to a when processing an access to b. As a simple heuristic, the counter $c_{b|a}$ could be created only if b appears as a *hypertext link* in a [5] . Studies have shown that an average HTML page contains 15 hypertext links [11] . Resources of several content types do not link directly to any other resources, thereby reducing the number of counters at the expense of missing some of the predictions and generating some predictions later than necessary.

For example, suppose the request sequence "y, z, b" is very common, but only z includes a hypertext link to b. Then, the counter $c_{b|y}$ would not exist, and an access to resource y would not generate a prediction for b. This would miss predictions particularly when τ is large enough to mask a sequence of accesses leading to b.

As a generalization of the heuristic, it is possible to create counters for pairs of resources that are reachable through a sequence of up to $h \geq 1$ hypertext links. However, traversing one or more hypertext links introduces additional complexity in the volume-construction process, since it requires parsing of the Web pages to extract and traverse the hypertext links. Contents of Web pages also change across time, including additions and deletions of hypertext links [12] . An alternative heuristic would create counters for resource pairs with URLs with the same *directory* prefix or set of directory prefixes, with the assumption that these resources may have similar content and/or belong to the same set of Web pages. For example, the resources *www.site.com/f/g/h.html* and *www.site.com/f/i/j.html* match in their first directory, but not the subdirectory. Like the hypertext heuristic, the directory-prefix heuristic may inadvertently omit counters for some pairs of resources that occur together frequently. In addition, the directory-prefix approach generates symmetric counters $c_{b|a}$ and $c_{a|b}$, even when a never follows b.

A more flexible heuristic would generate counters based on a rough estimate of the likelihood that b follows an access to a. For example, the counter could be created upon encountering an instance of b following up to T seconds and more than τ seconds after an access to a. In effect, the algorithm for accumulating the counters would create a new counter whenever it needed to increment one that does not already exist. However, this heuristic may still create a large number of counters, particularly when some resources occur together a small number of times. (e.g., a proxy accessing unrelated sets of resources in the same time period, due to requests from several clients). The number of unnecessary counters can be limited through random sampling, with a bias toward pairs with high pairwise probabilities. This heuristic involves two passes through the log to first generate a set of counters, and then compute their actual values. Suppose the first pass encounters an access to b that is preceded by an access to a in the $[\tau, T)$ time interval. If the counter $c_{b|a}$ does not already exist, we create a counter with a probability inversely proportional to the frequency of accesses to resource a. If a is often followed by b, then the counter is more likely to be created; the popularity of resource a alone is not a sufficient criterion. Also, the likelihood of creating a counter should be inversely proportional to the probability threshold p_t. This results in a liberal creation of counters for small values of p_t, and a more conservative creation of counters for larger values of p_t. After generating the counters, the log must be processed a second time to accumulate the access statistics from the beginning.

In summary, the number of counters can be reduced by these techniques:

- *Hypertext links*: Create counter $c_{b|a}$ if resource b is reachable from a through a sequence of up to $h \geq 1$ hypertext links.
- *Directory prefix*: Create counter $c_{b|a}$ if the URLs for resources a and b match in the first $k \geq 0$ directories.
- *Randomization*: While processing the log, create counter $c_{b|a}$ with probability $\frac{C}{p_t c_a}$ upon encountering the pair, where C is some constant that governs tradeoffs between the total number of counters created and the likelihood of missing important pairs.

These three approaches reduce the storage complexity of the pairwise volume-construction heuristic, at the risk of missing certain pairs of resources that occur together frequently.

3.3 Volume Thinning

The implication probability $p_{b|a}$ estimates the value of including resource b in volume v_a, independent of all of the other resources that may predict b. But local decisions can lower precision. Suppose that nine out of every ten accesses to a are followed by an access to b, resulting in $p_{b|a} = 9/10$. But, suppose these nine accesses to a occur with other resources x and y that also have high implication probabilities for b (e.g., sequences "x, y, a, b" where $p_{b|x}$, $p_{b|y} \geq p_t$). If the volumes v_x, v_y, and v_a all include resource b, then a never generates a new prediction for b. In fact, including b in volume v_a actually *decreases* precision, since a *mis*predicts b one-tenth of the time. In contrast, the greedy algorithm avoids placing b in volume v_a by reevaluating the utility of a as predictor after adding b to volume v_x or v_y.

To improve the precision and reduce the hintsize of pairwise volumes, we consider heuristics for removing ineffective predictions, without incurring the high computational complexity of the greedy algorithm. After computing the implication probabilities $p_{b|a}$ and applying p_t to construct an initial set of volumes, we perform a second pass through the log to measure the effectiveness of each predictor. The various heuristics differ in how they distribute "credit" for each prediction across the multiple resources that precede b. One possible heuristic favors the *first* resource that generates a prediction for b, within the $[\tau, T)$ interval [10] (in $xyab$, credit x for prediction); an alternate approach credits the *last* resource in the $[\tau, T)$ interval (i.e., a), under the assumption that the current access to b is more correlated with the most recent request from the same client than with the earlier accesses (x, and y).

These heuristics count the number of times $e_{b|a}$ that resource a (with $p_{b|a} \geq p_t$) is credited with generating a prediction for b. Then, resource b is included in volume v_a if the proportion of predictions exceeds a threshold e_t (i.e., if $e_{b|a}/c_{b|a} \geq e_t$). Hence, a resource b is included in volume a only if $p_{b|a}$ and $e_{b|a}$ are high enough, relative to p_t and e_t, respectively. Although these heuristics attempt to locate the most effective predictors, they thin the volumes without reevaluating the dependencies between resources. For example, suppose resources x, y, and z occur in two types of sequences xyz and yxz. Then, both heuristics would credit both x and y with half of the predictions of z. If the threshold e_t exceeds 0.5, neither v_x nor v_y would include resource z, even though including z in either volume would have successfully predicted all accesses to z. Or, if $e_t \leq 0.5$, the heuristics would place z in both volumes, even though one of the two predictions is always redundant. In comparison, the greedy algorithm would select one of the two resources (x or y) to include z in its volume, and then reevaluate the utility of the other resource.

4 Server Logs and Evaluation Tools

Our evaluation was based on several server logs gathered from outside and inside AT&T. One set of server logs were studied in earlier work dealing with volumes [10] – the logs from Amnesty International USA (AIUSA), Apache, Marimba, and Sun Microsystems. A new set of logs were obtained from AT&T's Easy World Wide Web (EW3) [13] , a web hosting part of the AT&T WorldNet business unit. Several thousands of companies and organizations have their web pages (with product information and other company related material) hosted on EW3. Roughly three months of server logs were obtained from EW3 and we examined server logs with significant traffic. We chose four out of the top ten logs based on the number of requests (smallest log had almost 4 million requests, largest had 46 million requests). The Sun log had fewer requests than the largest log but had many more unique resources and a higher request rate.

Server Log (days)	Number of Requests	Number of Clients	Unique Resources
Sun (9)	13,037,895	218,518	29,436
EW3_4 (94)	6,273,792	79,623	2638
EW3_6 (94)	6,281,303	102,454	784
EW3_18 (94)	3,902,302	58,839	279
EW3_60 (94)	45,903,646	481,628	2,031

Table 1: Characteristics of server logs reported in this paper

The entries needed for our study were the IP address of the client (proxy), the accessed resource, and the time of the access. We converted the first two into hashed integers, the ASCII time representation into an integer form (number of seconds since Dec 31 1969). To generate volumes and compute counters, we used stable sorting on the hashed forms of source IP addresses and processed the logs efficiently.

We now look at the software we used to process and analyze the server logs [14] . As is typical of server logs, several records were malformed due to very long lines (often due to inclusion of certain cgi resources that had an abnormal number of <name, value> pairs). We used the *ast* library and *sfio* (safe/fast i/o) [15,16] routines to avoid problems emanating from fixed width arrays and overflows. Sfio provides primitives designed to handled arbitrary size buffers and is portable across various architectures and file sizes. Its ability, for example, via `sfgetr()` to automatically convert a record being read in to a C string that can be parsed (using a *libast* routine `tokscan()`), enabled safe and quick parsing of logs into elements of interest to us. In order to eliminate multiple embedded `'/'` characters in URLs and to delete malformed URLs (with embedded `http` strings in the middle of the URL for example), we used the fast regular expression routines in *libast*.

The choice of C as language was a natural one in the presence of such powerful libraries, and we avoided the overhead of *perl* like interpreted programs. A small collection of C programs and *ksh* scripts sufficed to generate cleaned triples, volumes, and the various metrics. The data from the various logs were cleaned to generate the triples and the pairwise probabilistic volumes were generated. Next, the volumes were thinned using the effectiveness threshold criteria. Finally, the precision, recall, and hintsize metrics were extracted from the volumes. These tasks were accomplished via a set of small (around 100 lines) C programs. Very small *ksh* scripts extracted the relevant metrics output by the programs and fed them to a *gnuplot* drawing program. The entire process has now been fairly automated and can be run via a master script or a series of small scripts with only the parameters being varied.

5 Performance Evaluation

We first discuss trade-offs between the performance metrics for "regular" pairwise volumes (without effectiveness thresholding). We begin with the basic properties and interactions of the metrics of interest and then study the effect of varying different parameters such as time interval and effectiveness threshold. We compare the performance of all our volume-construction methods: greedy and pairwise (with and without effectiveness thresholding). Lastly, we investigate two trade-offs important for improving the computational complexity of maintaining and constructing volumes: transience of access patterns and volumes (how frequently volumes need to be regenerated) and impact on performance when volumes are constructed using a small sample of the log.

5.1 Trade-Offs for Pairwise Volumes

Figures 2–3 show the behavior of regular pairwise volumes for log 6, across several values of the time interval (T). Though we show only one log here, other logs show similar patterns of results. As can be seen, a similar general behaviour is depicted across all time intervals: recall and hintsize decrease with increasing probability threshold, whereas precision increases. The decrease of recall and hintsize with probability threshold, and the increase of recall with hintsize, are due to the fact that pairwise volumes with higher probability threshold are contained in the respective volumes with lower probability threshold. The increase in precision with decrease in recall and hintsize, is a property of optimal volumes and a desirable property from any heuristic. Pairwise volumes, mostly have this property (but not always, for an exception on the Sun log, see [10])

Figure 2 (a) shows that recall initially increases sharply as the probability threshold drops to about $p_t = 0.8$, For threshold values below 0.8, the recall steadily and slowly increases with decreased p_t. This indicates that very few requests are "covered" by high threshold values, but beyond that, "coverage" steadily increases, indicating that lower p_t values provide measurable additional information. Figure 3 (b) shows that the dependence of recall on average hintsize is concave (has a non-increasing first derivative). This corresponds to the fact that hints with higher pairwise probabilities are typically more valuable than hints with lower pairwise probabilities. Again, this is a property of optimal volumes which is consistent with our expectations, and hence, desirable in a heuristic.

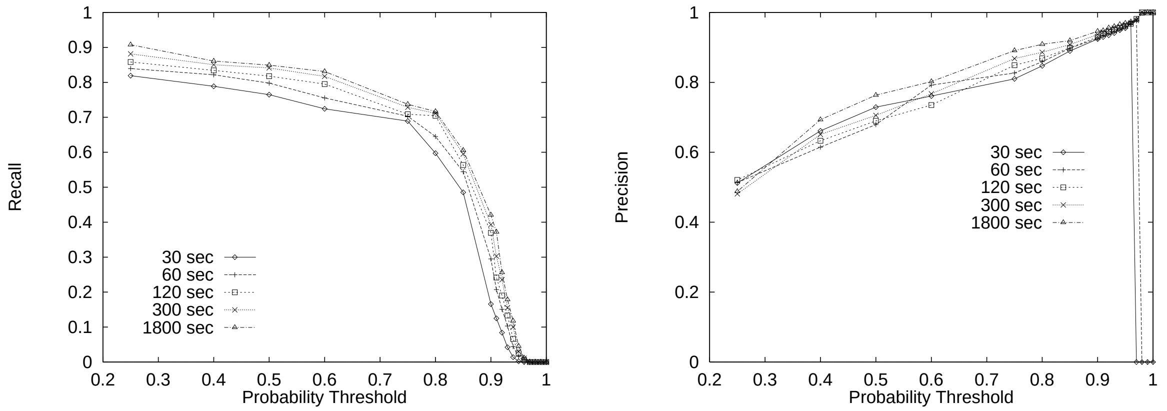
5.2 Maximum Time Interval (T)

Different applications may warrant the use of different lookaheads into future requests. Cache replacement benefits from longer time windows into the future (larger values of T), while cache validation, desiring minimal staleness, benefits from the more recent updates (small values of T).

Figures 2–3 vary the maximum time interval (T). Most of the "achievable" recall is captured in the smallest (30 seconds) time interval, reflecting the fact locality of reference is more pronouncingly present in smaller time scales. Beyond that, however, there is a measurable increase in recall with larger time intervals (T), while maintaining the same or better performance in terms of trade-offs with hintsize and precision, as the smaller time intervals. This indicates that there is good added value in considering longer time intervals, when appropriate for the application. The added value, however, tapers off as larger and larger time windows are needed to achieve similar increases in recall. For example, the increase in recall from 300 seconds (5 minutes) to 30 minutes is comparable to the increase from 120 to 300 seconds.

5.3 Effectiveness Threshold (e_t)

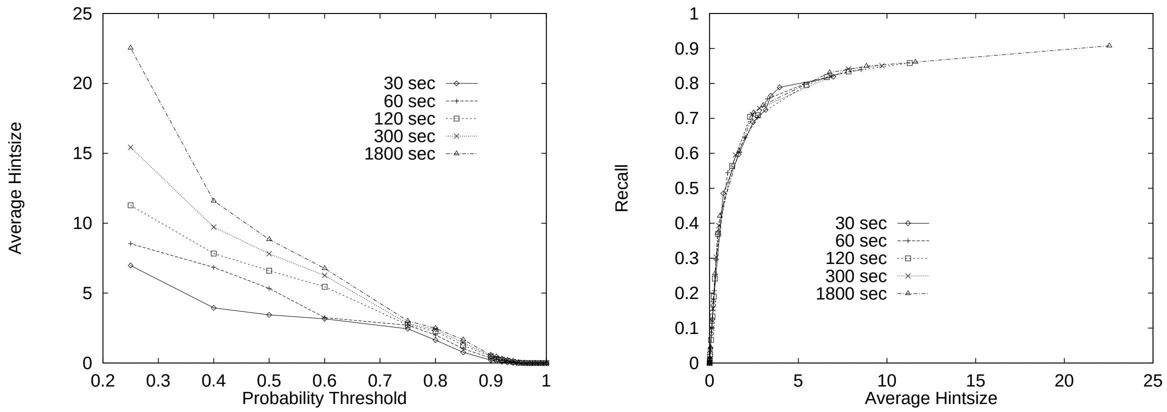
As discussed earlier, pairwise volumes can be thinned using effective thresholding which is controlled by a parameter e_t ($e_t = 0$ corresponds to regular pairwise volumes). We used the heuristic of distributing higher credit to the last resource in the (τ, T) interval (in contrast to the first-resource heuristic used in [10]). Figure 4 varies the effectiveness threshold for various logs. Higher e_t values produce volumes that are subsets of respective volumes with lower e_t values. Hence, as seen in the figures, recall and average hint size decrease with e_t. For the technique to be valuable, we expect precision to increase rapidly and for the hintsize to decrease more rapidly as recall drops. Consistently across logs, small e_t values exhibit significant drop in hintsize and a significant jump in precision, with only a moderate drop in recall. The improvements in hintsize and precision tapers off with larger e_t values, while recall continues to decrease steadily. The increase in precision levels off and then even drops, while hintsize levels off. Precision decreases because the effectiveness



(a) Recall vs. probability threshold

(b) Precision vs. probability threshold

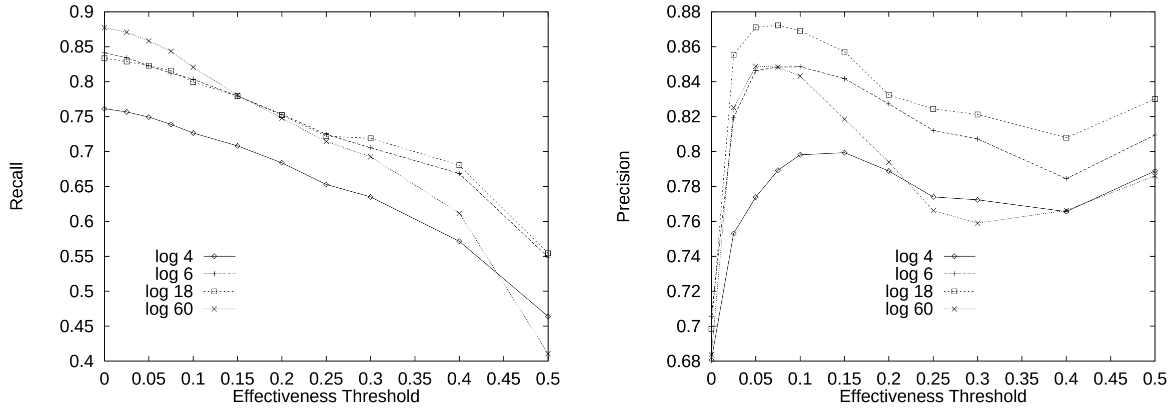
Figure 2: Recall and precision against probability threshold for different maximum time intervals T (log 6, $\tau = 0$ seconds)



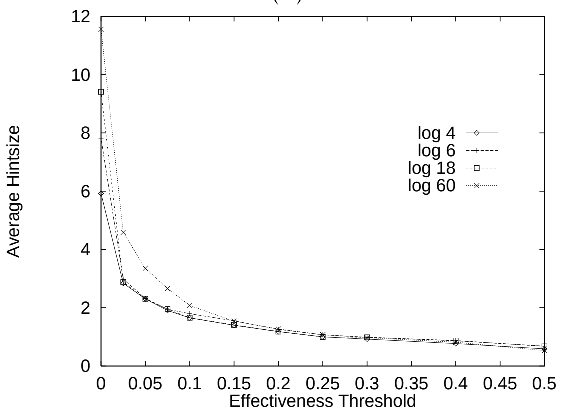
(a) Average Hintsize vs. Probability threshold

(b) Recall vs. Average Hintsize

Figure 3: Average hintsize vs. threshold and recall for different maximum time intervals T (log 6, $\tau = 0$ seconds)



(a) Recall vs. Effectiveness Threshold (b) Precision vs. Effectiveness Threshold



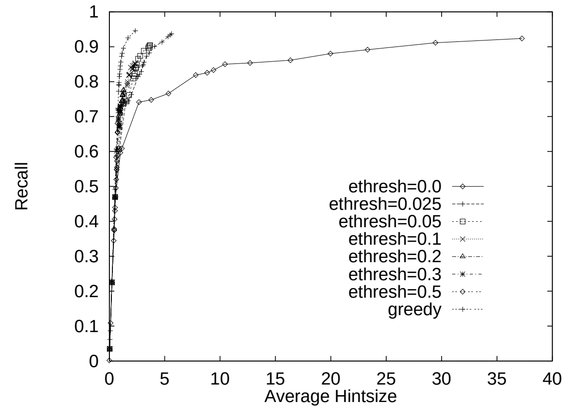
(c) Average Hintsize vs. Effectiveness Threshold

Figure 4: Recall, precision, and average hintsize against effectiveness threshold for different logs ($T = 300$ seconds, $\tau = 0$ seconds, $p_t = 0.25$)

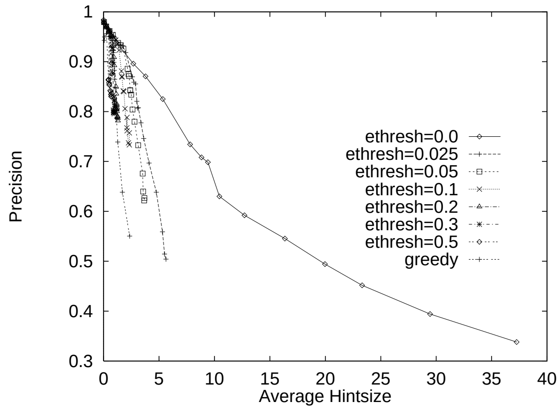
thresholding does not evaluate the impact of removing a volume element on the effectiveness of other predictors. Eventually, the thresholding becomes an overly aggressive mechanism that removes good predictors. Precision starts to increase again on e_t values exceeding 0.4, but this increase is accompanied by a sharp decrease in recall, and furthermore, the precision never recovers to the levels achieved by much higher-recall volumes. This shows that smaller values of e_t (around 0.1) provide the most improvement and use of higher e_t values is strongly discouraged.

5.4 Performance Comparison of Greedy, Pairwise, and Effectiveness Thresholding

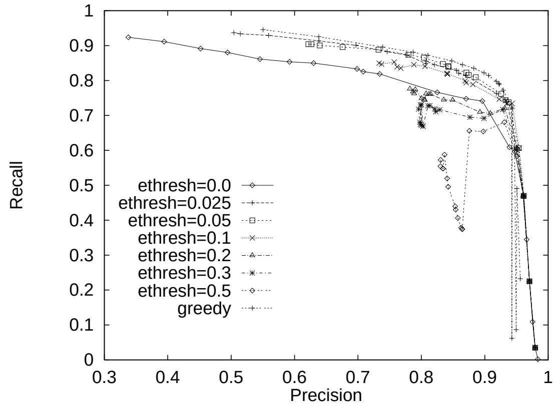
Figure 5 compares the performance of our volume construction heuristics on log EW3_18 (similar behaviour seen in other logs examined). We used greedy volumes, constructed for the optimization criteria of hintsize vs. recall, for all our experiments. As proposed in Section 2.3, the greedy algorithm was seeded with pairwise volumes, in order to achieve reasonable computation time. The greedy algorithm, as expected, outperforms all other techniques (at the cost of higher computation). The plots show how effectiveness thresholding helps in dramatically reducing the average hintsize and improving precision (by lowering false predictions), across the range of recall values. The plots



(a) Recall vs. Average Hintsize



(b) Precision vs. Average Hintsize



(c) Recall vs. Precision

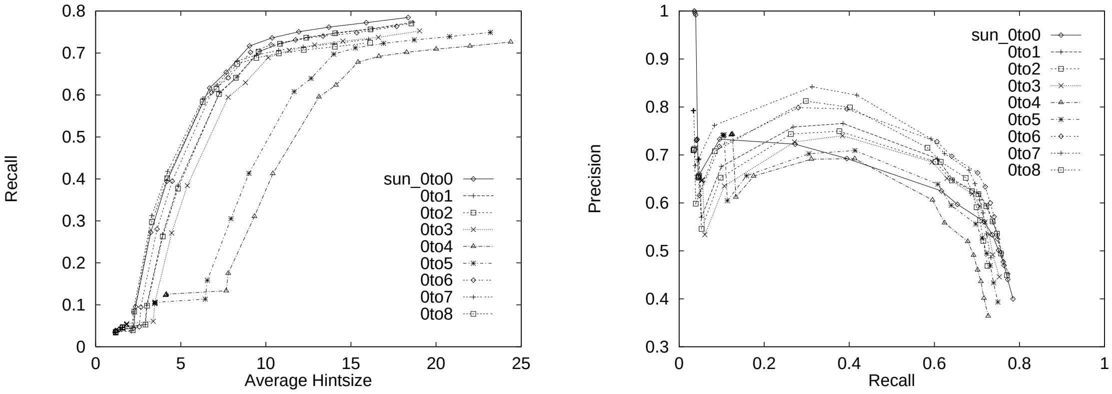
Figure 5: Recall and precision against average hintsize, precision vs. recall for various techniques (pairwise, effectiveness thresholding, greedy) for log EW3_18, with $T = 300$ seconds, $\tau = 0$ seconds

verify the behavior argued in Subsection 5.3: minimal effectiveness thresholding ($e_t \leq 0.1$) is the most valuable, while higher threshold values ($e_t \geq 0.2$) are both worse performers and exhibit undesirable dependencies (where precision decreases for lower recall).

Ultimately, the choice of volume construction method boils down to trading performance and complexity of constructing the volumes. Pairwise requires a single pass, and the effective heuristic an additional pass on the log. Greedy, in contrast, requires multiple passes on the log (typically, scores of iterations in our experiments), even when initialized with a pairwise volume. By stopping earlier after fewer iterations, greedy might still be able to generate volumes that are better than pairwise volumes, and run in time comparable (being explored in ongoing work).

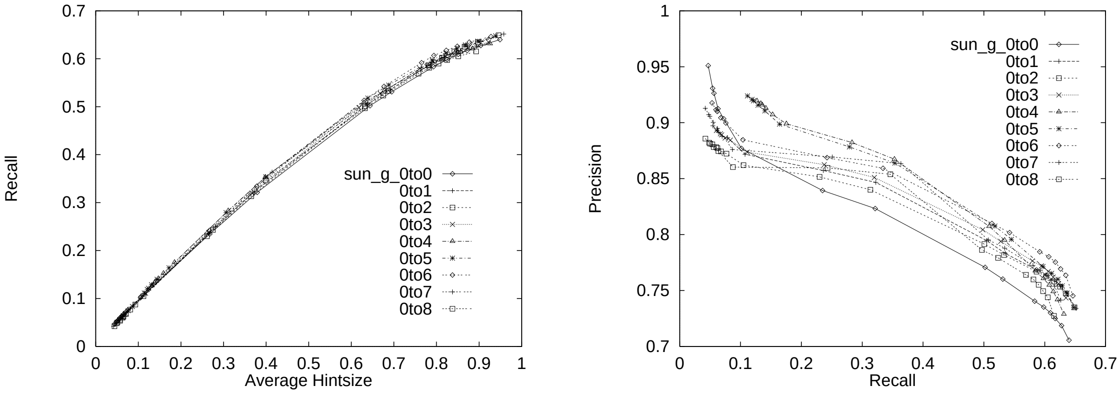
5.5 Aging of Volumes

We now look at how often volumes should be recomputed by the server. This often depends on how transient user access patterns are and the underlying URL structure. Daily to weekly updates, or updates only upon major changes to the site, sufficed for the examined logs. We partitioning server logs into separate days, built volumes for particular days, and applied these volumes to logs of different days.



(a) Recall vs. Average Hintsize (b) Precision vs. Recall

Figure 6: Pairwise volume of day 0, applied to Days 0–8, $T = 300$, $\tau = 0$

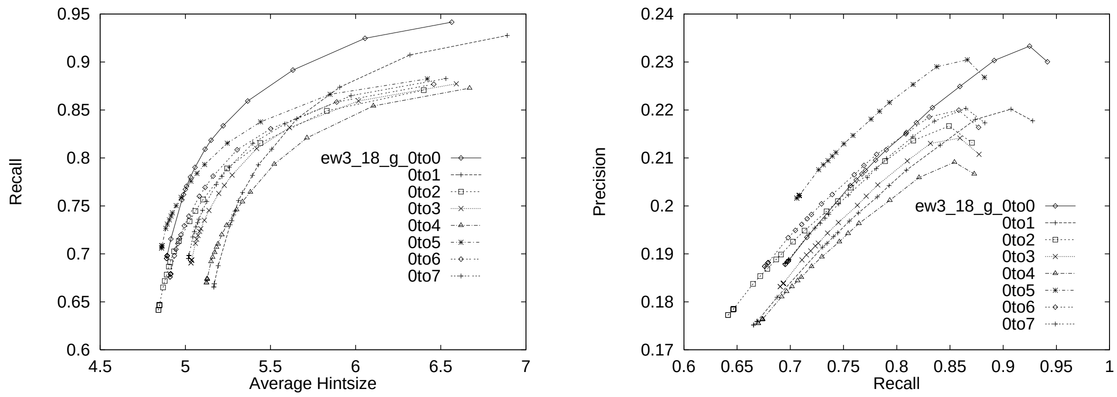


(a) Recall vs. Average Hintsize (b) Precision vs. Recall

Figure 7: Greedy volume for day 0, applied to Days 0–8, $T = 300$, $\tau = 0$

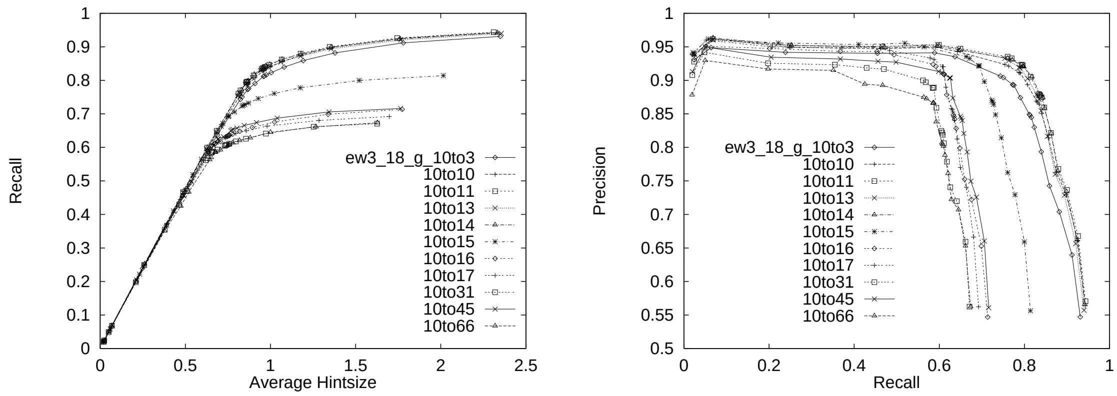
The Sun log was partitioned into 9 days (labeled 0–8, days 4 and 5 were weekend days). Figures 6–7 plot the tradeoffs of recall vs. average hintsize and precision vs. recall, when greedy and pairwise volumes for day 0 (with $T = 300$ and $\tau = 0$) are applied to all other days. The results show that access patterns did not change significantly during the time frame of a week. Weekend traffic exhibited different access patterns (apart from a different rate of requests) than weekday traffic. Figure 7 shows Greedy eliminating the initial "flat areas" of the recall vs. average hintsize curves. The flat areas are due to resources b with correlated predictors, that is, resources a with high $p_{b|a}$, that tend to predict the same set of requests for b. Once the highest (in terms of $p_{b|a}$ predictor is "in" (b is placed in v_a), placement of b in volumes of additional correlated predictors adds to the average hintsize without predicting many additional requests for b. Greedy avoids this by iteratively reevaluating the utility of each predictor. Figure 6(b) shows pairwise volumes exhibiting nonmonotonic dependence of precision and recall (the desirable behavior is for precision to increase as recall drops). As Figure 7(b) shows, this is stratified by greedy volumes.

We partitioned the EW3_18 log into separate days (0–93) and generated greedy volumes (with $T = 300$ and $\tau = 0$) for days 0 and 10. Figures 8 plots tradeoffs when volume of day 0 is applied to all days in the following week. Figure 9 plots the performance of the day 10 volumes on days in the same week and same-weekday in the previous and next several weeks. Figure 9 shows three clusters of days for which the day 10 volumes had similar performance. The clusters are for days 3–14, day 15, and days 16-66. Within each cluster there is a slight deterioration with time. This indicates



(a) Recall vs. Average Hintsize (b) Precision vs. Recall

Figure 8: Greedy volumes of day 0 in EW3_18 log, applied to days 0–7, $T = 300$, $\tau = 0$

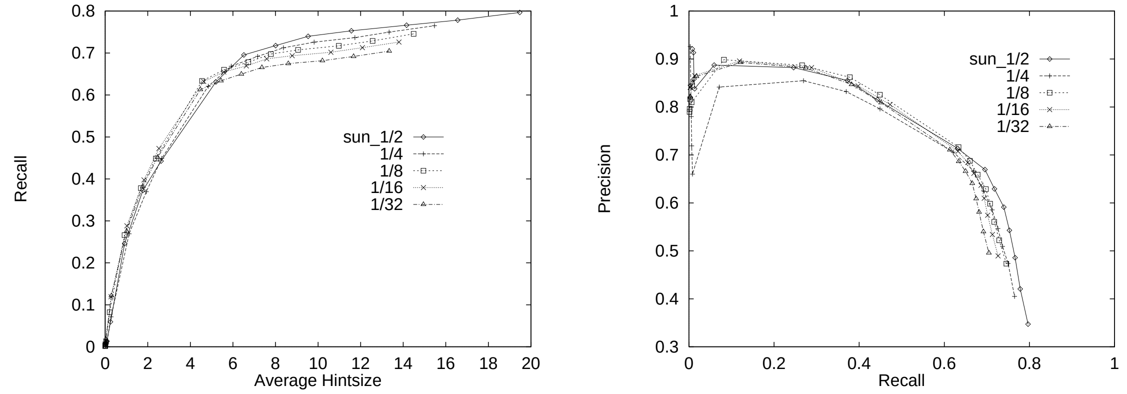


(a) Recall vs. Average Hintsize (b) Precision vs. Recall

Figure 9: Greedy volumes of day 10 in EW3_18 log, applied to various days in days 3–66, $T = 300$, $\tau = 0$

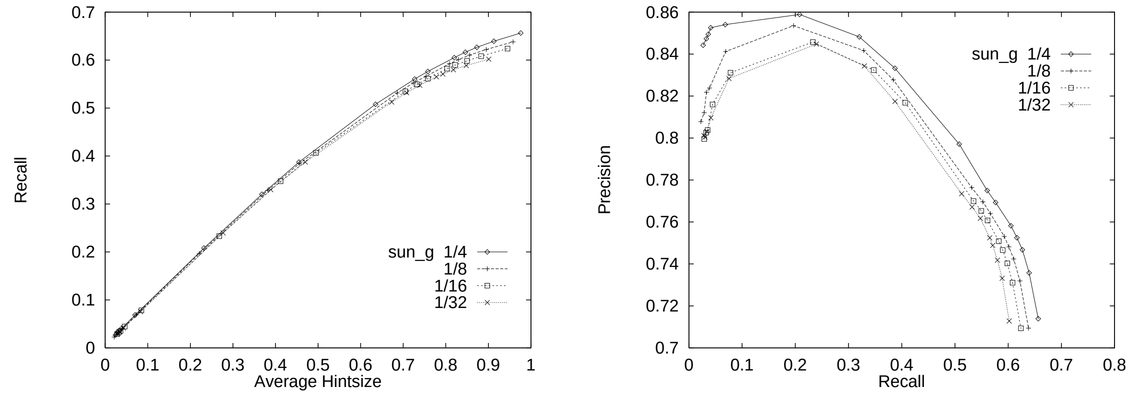
that a change in access patterns occurred during day 15, but otherwise, access patterns remained steady for prolonged periods of time. The change in access patterns was due to reconstruction on the site: the number of unique resources accessed increased by about 60% on day 15, and was relatively steady during previous and subsequent days.

The day 0 volumes show that greedy volumes geared towards one optimization criteria may perform poorly on another. The greedy volumes are constructed to "optimize" hintsize vs. recall. The day 0 volumes, however, (see Figure 8) achieve poor precision (note that precision increases with recall!). These volumes contained very high-utility (in the hundreds) items, where utility is the ratio between new predicted requests and number of hints, and is usually below 1, as discussed in Section 2.3. High-utility low-precision items originate by requests for s followed by blocks of requests to r, and other requests for s that are not followed by any request for r. Placing r in v_s has high utility on the hintsize vs. recall trade-offs; however, it yields many false predictions, and thus low precision. This phenomenon explains our choice of computing $p_{r|s}$ for pairwise volumes, where we attempted to obtain good performance on all three metrics (see Section 3). For pairwise volumes, we credit s exactly once for predicting r in the sequence "$srrrrr$" (greedy credits multiple times). This suggests using a similar counting in computing utility for the construction of greedy volumes; this would yield worse hintsize vs. recall tradeoffs, but obtain better precision.



(a) Recall vs. Hintsize (b) Precision vs. Recall

Figure 10: Performance of pairwise volumes generated for SUN sublogs containing requests from a fraction of the clients, and applied to the full log



(a) Recall vs. Average Hintsize (b) Precision vs. Recall

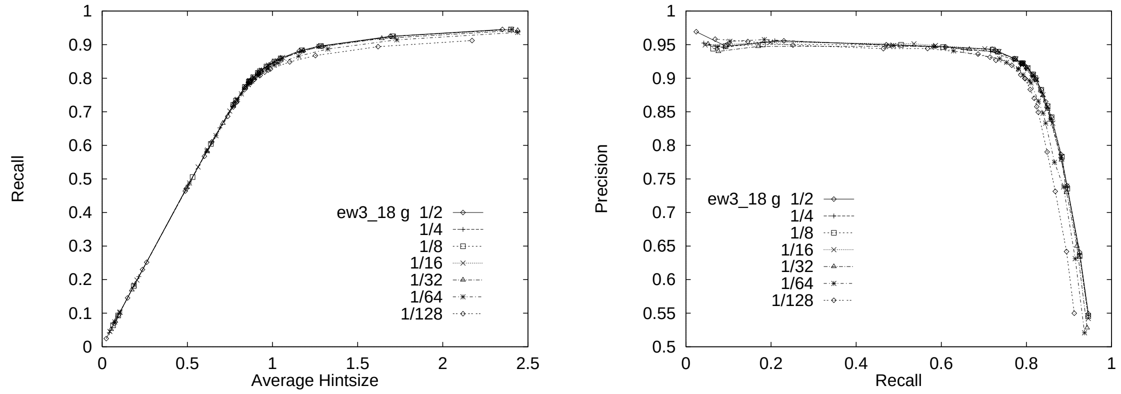
Figure 11: Performance of greedy volumes generated for SUN sublogs containing requests from a fraction of the clients, and applied to the full log

5.6 Constructing Volumes with Sampled Logs

The size of the logs often dictates the efficiency of our volume construction methods. We examine trade-offs of computation time and performance obtained by processing only a *sample* of the log ("sublog") to produce volumes. These sublogs were generated by including requests made by a fraction (random sample) of the clients. Sampling requests alone is likely to distort access patterns statistics whereas client sampling provides an unbiased sample.

On the Sun log, we generated pairwise volumes using 1/2, 1/4, 1/8,1/16, and 1/32 of the clients, and greedy volumes using 1/4–1/32 of the clients (with $T = 300$ and $\tau = 0$). These volumes were applied to the full log. The largest client, responsible for 5% of all accesses, exhibited significantly different access patterns and its inclusion during volume generation significantly affected the resulting volumes and their performance[1] The evaluation below excludes the largest client from all sublogs. The results are shown in Figures 10-11. Figure 12 plots results for a similar experiment for greedy volumes on EW3_18 log, using 1/2-1/128th of the clients. The figures show a very small decrease in performance for significant reduction in the size of the log that needs to be processed.

[1] This client is a spider that contacted the site daily.



(a) Recall vs. Average Hintsize (b) Precision vs. Recall

Figure 12: Performance of volumes generated for EW3_18 sublogs containing requests from a fraction of the clients, and applied to the full log

5.7 Summary of evaluation

Greedy volumes yielded the best performance in term of trade-offs of hintsize, recall, and precision. Though we were able to speed up greedy by seeding it with pairwise volumes, it requires scores of iterations, with each iteration requiring a pass through the log. An alternative two-pass construction is obtained by using pairwise volume with effectiveness thresholding. Low e_t values (around 0.1) consistently achieve large improvements in performance over pairwise volumes. All volume construction methods can benefit from sampling the log. Sampling clients results in very slight degradation of performance for large reductions in the size of the log processed. Access patterns, typically, are fairly stable; it thus suffices to update volumes daily or weekly, or upon reconstruction of the site.

6 Conclusions

We have examined a variety of algorithms and techniques to group resources at a Web server site for predicting future requests. The predictions can be used in applications such as prefetching of resources to reduce user-perceived latency, reduction of network traffic and bandwidth by obviating cache validation requests, and better cache replacement at a proxy site. Strong theoretical basis has been provided for the optimal volume construction problem and a variety of heuristics have been examined on a collection of large real-world server logs. We looked at three primary metrics: precision, recall, and hintsize. We analyzed server logs and constructed thinned volumes to be used as hints. We also show that it often suffices to look at subsets of the logs to get similar metrics. Since the problem naturally decomposes to an independent task of predicting requests for a single resource, we plan to look at incremental techniques in future.

A NP Completeness of Constructing Optimal Volumes

In Section 2 we formulated several optimization criteria for volumes construction, based on obtaining optimal tradeoffs of hintsize and recall or of recall and precision.

- Given a server log (sequence of triples) and an upper bound on average hintsize, find a set of volumes which maximizes recall.

- Given a server log (sequence of triples) and a desired recall, find a set of volumes that achieves this recall with minimum average hintsize.

The NP-completeness of both problems can be established by considering the decision problem (see [17]): Given a server log (sequence of triples) and two parameters, h and r, find a set of volumes (if one exists) with average hintsize at most h and recall at least r.

Two other optimization problems are:

- Given a server log and a lower bound on desired precision, find a set of volumes which maximizes recall.
- Given a server log and a desired recall, find a set of volumes that achieves this recall with maximum precision.

Similarly, the NP-completeness of both variants can be established by considering the decision problem: Given a server log (sequence of triples) and two parameters, p and r, find a set of volumes (if one exists) with precision at least p and recall at least r.

Lemma: The optimization problems defined above are NP-complete.

Proof: We provide a reduction from 3SAT [17] to the first decision problem. A similar reduction works for the second problem, and is omitted for brevity. Given an instance of 3-SAT, construct a log as follows: Create a resource for each variable and its negation. Create another "special" resource R. The log consists of many "independent subsequences" that capture the constraints of the 3SAT instance. Two requests are independent if made by different clients or are far enough in time (more than T seconds apart) so that predictions made using the volume for the first resource will expire before the other request arrives. Below we use $\tau = 0.1$ and $T = 2$. We elaborate on the subsequences. Let M be the number of clauses and N the number of variables in the 3SAT instance. We use the same symbol for each variable and its corresponding resource.

- For each variable x and its negation $\overline{x}$ we create $3M$ subsequences of the form: "$x, \overline{x}$ at time t and R at time $t+1$".
- For each clause of the form $(x \vee y \vee z)$ we create a subsequence of the form "x, y, z at time t, R at time $t+1$".

We claim that there exist a volumes set with recall of $r \geq M(3N+1)/(9MN+4M)$ and average hintsize of $h \leq (3MN+3M)/(9MN+4M)$, if and only if if the 3SAT instance is satisfiable.

Consider a set of volumes that solves the problem above. Observe that only requests for the resource R can be predicted, since other requests have no preceding requests during the previous $[\tau, T]$ seconds. There are $M(3N+1)$ requests for R, and $(9MN+4M)$ requests in total. Hence, recall of $M(3N+1)/(9MN+4M)$ means that the volumes predict all the requests for R. Now, for each variable x, there are requests for R that are preceded (and hence, can be predicted) only by x and $\overline{x}$. Hence, our volumes must be such that for each variable x, at least one of v_x or $v_{\overline{x}}$ contains R. Note, however, that each variable (or negation) with nonempty volume generates at least $3M$ hints. We had allowed for $(3MN+3M)$ hints in total, M of them must be used to predict the requests for R in the "clause" subsequences. Hence, at most N of the volumes can be non empty. By combining the above we obtain that exactly one of v_x and $v_{\overline{x}}$ equals $\{R\}$ and the other is empty.

Consider now the M subsequences that correspond to clauses. Each subsequence contains one request for R, and all these requests need to be predicted to achieve the desired recall. Consider a subsequence corresponding to the clause $(x \vee y \vee z)$. The request for R in the subsequence is predicted, if and only if at least one of v_x, v_y, and v_z equals $\{R\}$.

It follows that for any set of volumes with the stated recall and precision, a satisfying assignment can be obtained as follows: For each variable x we set x to be true if and only if the volume of x is equal to $\{R\}$. The considerations above establish that this is well defined and that all clauses are satisfied.

Conversely, given a satisfying assignment, the following set of volumes has the desired recall and hintsize: $v_R = \emptyset$, for each true variable x, $v_x = \{R\}$, and for each false variable y, $v_y = \emptyset$. ∎

B Performance Bounds for the Greedy Heuristic

We considered using the greedy heuristic with respect to two optimization criteria. We show that greedy has a performance ratio of $O(\log n)$ for the recall vs. hintsize trade-offs, where n is the number of requests.

Recall vs. Hintsize Trade-offs

Lemma: Greedy heuristic with respect to recall vs. hintsize trade-offs has performance ratio of $O(\log n)$.

Proof: We write each instance of our problem as an instance of the *weighted set-cover problem* [18,19] . The input to the weighted set cover problem is a set of elements, and a set of subsets over the elements. Each subset has an associated *weight.* The goal is to find a collection of subsets of minimum combined weight such that their union contains all elements. An important variant of the problem which is relevant to us is to find a collection of subsets of minimum weight which covers *at least a fraction of the elements.* Given an instance of our problem, we view each request as an "element." Each placement of a resource s in a volume v_r constitutes a "subset." The weight of the subset is the total number of request for r (the number of hints produced when s is placed in v_r). The subset contains all elements corresponding to requests for s occurring within $[\tau, T]$ time units following some request for r. It is easy to establish that a solution for the weighted set cover instance with a certain coverage and weight corresponds to volumes with recall that equals the fraction of covered elements and weight that corresponds to the total hintsize. For example, a set cover that covers 50% of the elements and has weight (combined weights of all participating subsets) 1000, corresponds to a solution of our problem with recall 50% and (total) hintsize 1000 (average hintsize is $1000/n$). Therefore, there is correspondence between the optimal solution of the greedy instance and an optimal solution for our problem. Furthermore, a solution to the set cover instance with approximation ratio of k (k times worse than optimal) corresponds to a volume set with the same performance ratio.

The greedy algorithm has a logarithmic performance ratio [19] for the weighted set cover problem. The well-known greedy set cover algorithm reduces to our greedy heuristic on instances of our problem. To see that, observe that predictions made for different resources are independent (each "subset" covers only requests to a single resource and hence, subsets covering requests to different resources have empty intersection.) Thus, in constructing a set cover it suffices to consider covering requests of each resource r idependently (that is, choosing a set of volumes to place r), as our greedy algorithm does. It follows that our greedy heuristic has a logarithmic performance ratio. ∎

C Performance of the Pairwise Heuristic

The plain pairwise heuristic is the heart of much previous work on web prefetching [1,2,5,10] . We outline a family of sequences where pairwise performs poorly. The patterns of these sequences is common in actual request sequences, explaining the relatively poor performance we experimentally observed for the unimproved pairwise heuristic.

Lemma: The pairwise heuristic has performance ratio of $\Omega(n)$, where n is the number of requests.

Proof: We construct a family of logs, parameterized with k. The log has a set of resources $p_1, \ldots, p_q, q, R$. The request sequence consists of many "independent" subsequences (subsequences are independent if more than T seconds separate the last request in one subsequence and the first request in another):

- 9 repetitions of the subsequence: simultaneous (within τ seconds) requests for $p_1, \ldots, p_k$ followed (within (τ, T) seconds) by a request for R.
- 8 repetitions of the subsequence: request for q, followed (within (τ, T) seconds) by a request for R.
- For $i = 1, \ldots, k$, a single-request subsequence "p_i."
- 2 single-request subsequences "q."

The pairwise probabilities are $p_{R|p_i} = 0.9$ and $p_{R|q} = 0.8$. Hence, a pairwise volume set includes R in the volumes $v_{p_1}, \ldots, v_{p_k}$ before including it in the volume v_q.

Hence, a pairwise volume set that predicts all requests for R have total hintsize $10(k+1)$ (number of requests for $p_1, \ldots, p_k$ and q). This volume set predicts all $9 + 8 = 17$ requests for R, but makes $k + 2$ "false" predictions. Hence, it has precision of $17/(19 + k)$.

The optimal set of volumes contains two non-empty volumes: $v_{p_1} = v_q = \{R\}$. It predicts all requests for R with total hintsize 20 and precision 17/20 (it generates only 3 "false predictions").

It follows that the performance ratio of pairwise is $\Omega(k)$. Since the number fo requests n is proportional to k, the performance ratio is $\Omega(n)$. ∎

To conclude we note that repeated sequences of requests of the form $p_1, \ldots, p_k$ are common as web request sequences (due to embedded images or following hyperlinks), and pairwise volumes perform suboptimally on such sequences.

References

[1] A. Bestavros, "Using speculation to reduce server load and service time on the WWW," in *Proc. ACM Inter. Conf. on Information and Knowledge Management*, 1995. `http://www.cs.bu.edu/faculty/best/res/papers/cikm95.ps`.

[2] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve World Wide Web latency," *Computer Communication Review*, vol. 26, no. 3, pp. 22–36, 1996. `http://daedalus.cs.berkeley.edu/publications/ccr-july96.ps.gz`.

[3] T. M. Kroeger, D. E. Long, and J. C. Mogul, "Exploring the bounds of Web latency reduction from caching and prefetching," in *Proc. USENIX Symp. on Internet Technologies and Systems*, pp. 13–22, December 1997. `http://www.usenix.org/events/usits97`.

[4] 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`.

[5] Z. Jiang and L. Kleinrock, "An adaptive prefetching scheme," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 358–368, April 1998. `millennium.cs.ucla.edu/~jiang/Research/Publication/extended.ps`.

[6] C. Liu and P. Cao, "Maintaining strong cache consistency in the World Wide Web," in *Proc. IEEE Inter. Conf. on Distributed Computing Systems*, pp. 326–334, May 1997. `http://www.cs.wisc.edu/~cao/papers/icache.html`.

[7] B. Krishnamurthy and C. E. Wills, "Piggyback server invalidation for proxy cache coherency," in *Proc. World Wide Web Conference*, April 1998. `http://www.research.att.com/~bala/papers/psi-www7.ps.gz`.

[8] J. C. Mogul, "Hinted caching in the Web," in *Proc. ACM SIGOPS European Workshop*, 1996. `http://mosquitonet.stanford.edu/sigops96/papers/mogul.ps`.

[9] E. Cohen, B. Krishnamurthy, and J. Rexford, "Evaluating server-assisted cache replacement in the web," in *Proc. European Symposium on Algorithms*, August 1998. `http://www.research.att.com/~bala/papers/esa98.ps.gz`.

[10] E. Cohen, B. Krishnamurthy, and J. Rexford, "Improving end-to-end performance of the web using server volumes and proxy filters," in *Proc. ACM SIGCOMM*, pp. 241–253, September 1998. `http://www.research.att.com/~bala/papers/sigcomm98.ps.gz`.

[11] A. Woodruff, P. Aoki, E. Brewer, P. Gauthier, and L. Rowe, "An investigation of documents on the world world web," in *Proc. 5th Int'l Conf. on the World Wide Web*, pp. 963–980, May 1996. `http://www5conf.inria.fr/fich_html/papers/P7/Overview.html`.

[12] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul, "Rate of change and other metrics: A live study of the World Wide Web," in *Proc. USENIX Symp. on Internet Technologies and Systems*, pp. 147–158, December 1997. `http://www.research.att.com/~bala/papers/roc-usits97.ps.gz`.

[13] AT&T Easy World Wide Web `http://www.att.com/work-net/easywww`.

[14] B. Krishnamurthy and J. Rexford, "Software issues in characterizing web server logs," in *Proc. Web Characterisation Workshop*, November 1998.
`http://www.w3.org/1998/11/05/WC-workshop`.

[15] G. Fowler, D. Korn, and K.-P. Vo, "Libraries and file system architecture," in *Practical Reusable UNIX Software* (B. Krishnamurthy, ed.), ch. 2, New York, NY: John Wiley & Sons, 1995.
`http://www.research.att.com/library/books/reuse`.

[16] D. Korn and K.-P. Vo, "Sfio: Safe/fast string/file IO," in *Proceedings of Summer 1991 USENIX Conference*, pp. 235–256, 1991.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability.* San Francisco, CA: W. H. Freeman and Co., 1979.

[18] D. S. Hochbaum, "Approximating covering and packing problems," in *Approximation Algorithms for NP-Hard Problems* (D. S. Hochbaum, ed.), Boston, MA: PWS Publishing Company, 1995.

[19] V. Chvátal, "A greedy heuristic for the set-covering problem," *Mathematics of Operations Research*, vol. 4, pp. 233–235, 1979.