Web Caching using Access Statistics

Adam Meyerson* Kamesh Munagala† Serge Plotkin‡

May 12, 2000

Abstract

We present the problem of caching web pages under the assumption that each user has a fixed, known demand vector for the pages. Such demands could be computed using access statistics. We wish to place web pages in the caches in order to optimize the latency from user to page, under the constraints that each cache has limited memory and can support a limited total number of requests. When C caches are present with fixed locations, we present a constant factor approximation to the latency while exceeding capacity constraints by $O(\log C)$. We improve this result to a constant factor provided no replication of web pages is allowed. We present a constant factor approximation where the goal is to minimize the maximum latency. We also consider the case where we can place our own caches in the network for a cost, and produce a constant approximation to the sum of cache cost plus weighted average latency. Finally, we extend our results to incorporate page update latency, temporal variation in request rates, and economies of scale in cache costs.

*Supported by ARO DAAG-55-97-1-0221. Department of Computer Science, Stanford University CA 94305. Email: awm@cs.stanford.edu.

†Supported by ONR N00014-98-1-0589. Department of Computer Science, Stanford University CA 94305. Email: kamesh@cs.stanford.edu.

‡Supported by ARO Grants DAAG55-98-1-0170 and ONR Grant N00014-98-1-0589. Department of Computer Science, Stanford University CA 94305. Email: plotkin@cs.stanford.edu.

1 Introduction

As web traffic increases, the added congestion makes accessing pages more difficult. Caching of web data is quickly becoming key in reducing this congestion, particularly for more popular pages. In this paper, we consider the natural theoretical problems of assigning pages to caches and determining the optimal cache locations on the web.

Our caching paradigm assumes that every web domain (like "yahoo" or "cnn") is statically cached in certain caches, and user requests are routed to the appropriate cache. Static caching of domains has been shown to work in practice [20, 5]. We also assume that accurate data on user preferences is available to our algorithm, and remains relatively static over time. We can merge all the users in a single region of the network for these purposes, and experimental observations [2, 21, 20, 6] tend to support our assumptions.

Our assignments will attempt to accomplish four things: we want to minimize the internet distance from users to the pages they wish to view, we want to keep the traffic on any given cache below a certain threshold, we want to keep the total number of caches opened from growing too large, and we should not exceed the bounded memory of any cache. With these goals, we consider three different problems related to web caching. These problems are motivated by different optimization requirements for the caches. Typically, such optimization would be used by Internet Service Providers who place these caches in their networks, or by independent entities who manage the storage requirements for the content providers of the web pages.

The problems we present below are generalizations of the classical facility location [18] and k-median [10, 4] problems, which are NP-hard. We therefore are interested in solving them approximately. The approximation technique we use is rounding of linear programs. The rounding schemes combine ideas from randomized rounding [14, 15, 19, 13] and path filtering [10, 18] with many new techniques.

First, we look at the Page-Placement problem. We assume that caches already exist; our goal is to allocate a set of web pages to these caches in order to optimize the average user's quality-of-service. If we were to assume that every cache contains a copy of every page, this would become a minimum cost flow problem. However, we will assume that cache memory is relatively small, creating an upper bound on the number of pages allowed in a cache. We would like to allocate pages without exceeding the cache memory, in such a way that average distance from a user to a cache containing the page the user wishes to view is small. Our algorithm, based on linear program rounding, finds an allocation which cannot exceed the cache memory by more than an $O(\log C)$ factor, while making sure that no more than $O(\log C)$ times a threshold number of users access any given cache. In these ratios, C is the number of caches. We make sure that the average user's network distance from his assigned cache is within a factor of 5 of the optimum. If we are allowed to assume that every page is cached only once, we can improve these bounds to exceed cache memory and the user threshold by only a factor of 4 while obtaining optimum average distance-to-cache.

Second, we consider the problem of Cache-Placement. We suppose that we are allowed to open our own caches at a variety of locations, but we must pay for each cache opened. Once we have selected caches to open, we allocate pages and users to caches, optimizing the weighted sum of average user-to-cache distance with the cost of opening the caches. While this problem might seem more difficult than Page-Placement because of the additional decision to open caches, it turns out we can attain better performance because we are allowed to open multiple caches at the same location. This problem is related to capacitated facility location [18, 9], with additional constraints created by the limited size of cache memory. Our LP-rounding algorithm provides a 13.325 approximation on the total cost while guaranteeing that no cache exceeds its cache memory or its threshold number of users.

Third, we consider the problem of minimizing Maximum-Latency. We suppose that we have C caches placed at fixed locations in a network, and we wish to cache pages to minimize the

distance between any user and the closest cache containing the page the user requests. For this problem, we assume that any cache can serve an arbitrary number of users. This is a variant of PAGE-PLACEMENT where we are interested in maximum distance rather than average distance, and is closely related to the K-CENTER problem [7]. We show how to guarantee that latency is within a factor of 6 of optimal, while exceeding cache memories by at most a factor of 2.

Finally, we mention various straightforward extensions of our algorithms, accounting for situations where pages have different memory sizes, where users have different bandwidth demands, and where we need to guarantee that pages have small latency of update. None of these situations will effect our performance bounds. We also show how our algorithms can exploit economies of scale in cache cost, *i.e.*, the cost of the caches grow sub-linearly with the cache capacity.

1.1 Previous Work

Previous theoretical work on web caching [6, 1, 8] has focussed on online page replacement policies. We differ from these works in that we assume pages are statically cached in certain caches, and we know the steady state access patterns for these pages by different users. Our algorithms are therefore offline, and need to be run only when there is a significant shift in usage patterns. There has been some previous work on adapting existing online single cache replacement policies to multiple caches [11, 12], but these approaches work only for certain classes of networks. The performance of our approach is independent of the network topology, and the number of users and pages.

The problems we solve and the techniques we use are closely related to three different types of problems. The problem of packing pages into caches so that capacity constraints of the caches are satisfied is similar in flavor to *covering and packing* problems [15, 19, 13]. The standard technique used for solving these problems is randomized rounding [14]. The *Generalized Assignment Problem*(GAP) [17] is a special packing problem for which constant approximations are known.

The problem of placing caches in a network and deciding how many times and where to replicate the pages is similar to the classical facility location and k-median problems [10, 18]. The algorithms in [18, 3, 9] give efficient constant factor algorithms to the facility location problem, while the algorithms in [4, 9, 3] give a constant approximation to the k-median problem.

Finally, we also consider scenarios in which we take advantage of economies of scale while buying and placing caches. This is related to the buy-at-bulk network design problem [16]. The facility location algorithm in [9] gives a 4-approximation for the case where the facilities have capacities and we can place multiple facility copies at any location.

1.2 Organization of the Paper

The next section presents the PAGE-PLACEMENT problem, where the cache locations are fixed. The main technique here is randomized rounding of an LP after consolidating user demands at cache locations. We augment the LP by a cutting plane to improve the quality of approximation. In Section 3, we present the CACHE-PLACEMENT problem and give a constant approximation for it. Our technique involves applications of the filtering technique in [10] combined with new ideas. We show how we can extend the framework to incorporate economies of scale in Appendix C. In Section 4, we present the MAXIMUM-LATENCY problem and give a constant approximation for it. Our technique here involves a two stage rounding of an LP, where we ensure the first stage rounding preserves the feasibility of the LP. The two rounding stages use techniques from [18] and [17] respectively. We conclude by mentioning extensions and open problems.

2 The PAGE-PLACEMENT Problem

We are given a set P of pages, and a set C of caches. The caches are placed at known locations in the network represented by a graph $G(V, E)$ with a distance(latency) function $d(e)$ on the edges. Each user in the network requests a page. We must assign pages and users to caches, guaranteeing that at most cap_u users and at most cap_p pages are assigned to any one cache. We must also guarantee that if a user is assigned to a cache, the page he requested is present at that cache. Given these restrictions, we wish to minimize the sum over users of the distance along the network from the user to his assigned cache.

Our algorithm will obtain within constant factors of the optimal distance sum, while guaranteeing at most $O(\log |C|)cap_u$ users and $O(\log |C|)cap_p$ pages on each cache. Since the algorithm is randomized, we will obtain these bounds to high probability; repeated executions of the algorithm makes the probabilities arbitrarily good.

2.1 Algorithm

First, we assign users to caches without exceeding cap_u users on any cache, ignoring the page bound. This is simply a min-cost flow problem, and we can solve it exactly. Since ignoring the page bound can only help our cost, we guarantee $cost_1 \leq OPT$.

We now merge users who were mapped to the same cache and request the same page. This gives us a total of $|C| * |P|$ users, each with a "size" equal to the number of real users it represents. These sizes are between 1 and cap_u, since no more than cap_u users were mapped to any cache. We define U to be this new weighted set of users. We let $W(u)$ represent the "size" of user u , and $d(u, c)$ represent the distance between user u and cache c. $G_c(u)$ represents the fraction by which user u receives from cache c, while $I_c(p)$ represents the fraction of page p present in cache c. Call the page wanted by user u page p_u. We will solve the linear relaxation of the following integer program:

$$\text{Minimize} \sum_u \sum_c G_c(u) W(u) d(u, c)$$

$$\begin{array}{rcll} G_c(u) & \leq & I_c(p_u) & \forall u \in U, c \in C \\ \sum_c G_c(u) & = & 1 & \forall u \in U \\ \sum_p I_c(p) & \leq & cap_p & \forall c \in C \\ \sum_u W(u) G_c(u) & \leq & cap_u & \forall c \in C \\ \sum_{u:p_u=p} W(u) G_c(u) & \leq & cap_u I_c(p) & \forall p \in P, c \in C \\ G_c(u), I_c(p) & \in & \{0, 1\} & \forall u \in U, c \in C, p \in P \end{array}$$

Note that the constraint $\sum_{u:p_u=p} W(u) G_c(u) \leq cap_u I_c(p)$ is a cutting plane required to make the rounding scheme outlined below go through.

Suppose we assign each page to the caches where the optimal solution places the page. We then map each user fractionally back to the locations from which the requests originated, and from there to their optimal caches. It follows that $frac \leq OPT + cost_1$, where $frac$ represents the cost of the fractional solution to this linear program.

We call our optimal fractional solution to this program G_c^* and I_c^*. We scale these variables as follows.

$$I_c^D(p) = min(1, 2I_c^*(p))$$

If $d(u, c) \geq 2 \sum_c G_c^*(u) d(u, c)$ then $G_c^D(u) = 0$, otherwise $G_c^D(u) = min(1, 2G_c^*(u))$.

This scaling removes long-distance paths from consideration, while exceeding the capacity equations by at most a factor of two. We now scale again in preparation for randomized rounding.

- $I_c^S(p) = min(1, 2(\log |C|) I_c^D(p))$
- $G_c^S(u) = min(1, \frac{2(\log |C|) G_c^D(u) I_c^D(p_u)}{I_c^S(p_u)})$

We now perform a randomized rounding on these scaled variables. Notice that all the variables will be rounded independently.

- $I_c^R(p) = 1$ with probability equal to $I_c^S(p)$.
- $G_c^R(u) = 1$ with probability equal to $G_c^S(u)/I_c^D(p_u)$.

Finally, we eliminate G_c variables which would send a user to a cache which does not have the page he's looking for, yielding:

- $I_c^F(p) = I_c^R(p)$
- $G_c^F(u) = I_c^R(p_u) G_c^R(u)$.

These final values for $G_c^F(u)$ are not independent, since if two users want the same page from the same cache, they will both have the same $I_c^R(p)$ term in their formula. Some users may not have been assigned to any cache ($G_c(u) = 0$ for all c). These users will be assigned to the closest cache; of course this is the cache where they were placed in the first stage of the algorithm. If the proper page is not in the cache, we will place it there. Clearly this algorithm assigns every user to a cache, and makes sure that each cache has the corresponding page for the users assigned there. We will show in the next section that the total cost is within a constant factor of optimum, and that no capacity (in terms of pages or users) is exceeded by more than $O(\log |C|)$.

2.2 Analysis

We will begin by proving that most users are satisfied in the LP-rounding stage, leaving a relatively small number of users to be satisfied in the final stage by sending them to their local cache.

Lemma 1. *For any choice of user, $u \in U$, $Pr[\sum_c G_c^F(u) = 0] < e/|C|$.*

Proof. Refer Appendix A □

We note that for users desiring different pages, the decision as to where to send those users is independent.

Lemma 2. *For users u_1, u_2 desiring different pages, the values of the random variables $G(u_1) = \sum_c G_c^F(u_1)$ and $G(u_2) = \sum_c G_c^F(u_2)$ are independent.*

We continue to prove that the probability of placing a lot of pages on a single cache in the LP-rounding stage is small. This indicates that, with high probability, the LP-rounding part of the algorithm does not exceed page capacity bounds on caches by too much.

Lemma 3. *For any cache $c \in C$, $Pr[\sum_p I_c^F(p) > (6 \log |C|) cap_p] < 1/|C|^2$.*

Proof. Refer Appendix A. □

We also need to prove that the additional pages added to caches by the final stage of the algorithm don't cause us to overflow page capacities by too much. This depends on our previous lemmas indicating that most pages were satisfied by the LP-rounding and the satisfying of users wanting different pages is independent.

Lemma 4. *The probability that there exists a cache with more than* $(3e+6)(\log|C|)cap_p$ *pages in the final assignment is at most* $(1/|C|)+(1/|C|^5)$.

Proof. Consider the pages assigned to caches during the final step of the algorithm. Originally, we assigned at most $|P|$ pages to each cache. Lemma 1 proves at most $e/|C|$ probability to remain unsatisfied after the LP rounding phase. Whether each page remains is, according to lemma 2, an independent Bernoulli variable. The expected number of pages remaining is $|P|e/|C| \leq e(cap_p)$ since there must be enough space to place each page on some cache. Let X_c be the number of pages assigned to cache c in the last step.

Using Chernoff bounds, and assuming that $cap_p > 2\log|C|$, for any given cache we have X_c pages assigned in the last step:

$$Pr[X_c > e(3\log|C|)cap_p] < (e^2/27)^{3e\log|C|} < 1/|C|^6$$

Now applying lemma 3, the probability that any given cache has at least $(3e+6)(\log|C|)cap_p$ pages is bounded above by $(1/|C|^2)+(1/|C|^6)$. The probability that some cache has at least that many pages is bounded by $(1/|C|)+(1/|C|^5)$. □

We also need to show that we don't overflow the capacities based on the number of users by too much. We first show that the LP-rounding part of the algorithm is unlikely to cause more than $O(\log|C|)$ overflow.

Lemma 5. *For any cache* $c \in C$, $Pr[\sum_u W(u)G_c^F(u) > (18\log|C|)cap_u] \leq (1/|C|^2)+(1/|C|^6)$.

Proof. We divide the users into users wanting different pages. We define $S(p)$ to be the sum over users wanting page p of $W(u)G_c^R(u)$. It follows that $\sum_u W(u)G_c^F(u) = \sum_p S(p)I_c^R(p)$. We know from the cutting plane equation that for any chosen page where $I_c^S(p) < 1$, $E[S(p)] \leq 2cap_u$. Suppose we assign the random values of $I_c^R(p)$. What is the new expected value, based on the choice of $G_c^R(p)$, for the sum, ignoring pages for which $I_c^S(p) = 1$? This expected value is a sum of independent Bernoulli variables, each of which is at most $2cap_u$. The expected value (based on choices of I) of this expected value is at most $2cap_u$. Using Chernoff bounds, the probability that this expected value is above $(4\log|C|)cap_u$ is at most $1/|C|^2$ for large $|C|$. The sum of $S(p)$ where $I_c^S(p) < 1$ has expected value at most $(2\log|C|)cap_u$. Given that the overall expected value is below $(6\log|C|)cap_u$, we now determine the values for $W(u)G_c^R(u)$. Since each weight is at most cap_u, this is a sum of bounded Bernoulli variables. It follows that the probability of this sum exceeding $(18\log|C|)cap_u$ is at most $(e^2/27)^{6\log|C|} \leq 1/|C|^6$. The overall probability that a given cache serves more than $(18\log|C|)cap_u$ users is bounded by $(1/|C|^2)+(1/|C|^6)$. □

We notice that at most cap_u weight of users are assigned to a cache in the final stage, even if all users from the first stage are assigned there. It follows that the total number of users assigned to a cache are at most those assigned by randomized rounding plus cap_u. Applying lemma 5 gives us:

Lemma 6. *The probability that there exists a cache with more than* $(1+18\log|C|)cap_u$ *users in the final assignment is at most* $(1/|C|)+(1/|C|^5)$.

What's the overall chance that "something goes wrong" when we run this algorithm? We define "something" to be overflowing either the user capacity or the page capacity on some cache by more than $O(\log|C|)$. We combine the results of lemma 4 and lemma 6 to get that the chance of this occurring is polynomially small.

Theorem 1. *The probability that there exists a cache with more than* $(3e+6)(\log|C|)cap_p$ *pages or more than* $(1+18\log|C|)cap_u$ *users in the final assignment is at most* $(2/|C|)+(2/|C|^5)$.

Finally, we show that the total cost of our solution is good. This will hold regardless of the random choices made.

Theorem 2. *The total cost is at most five times optimal.*

Proof. The distance from the first stage of the algorithm is at most optimal. The LP-rounding stage guarantees that a user can only be sent to a cache within a factor of two of its average distance in the fractional optimal; thus the rounded solution we get has at most twice the total distance of the fractional optimal which is at most twice the optimal distance. The final stage of the algorithm simply places users in caches to which they've already been moved, incurring no extra distance. Overall total distance is at most five times optimal. □

We've shown that the probability of violating a user or page capacity constraint on some cache by more than $O(\log |C|)$ is polynomially small, while guaranteeing that the total distance is within 5 times optimal. By repeating our randomized algorithm several times, we can guarantee that we find a "good" solution which does not exceed any constraint by more than $O(\log |C|)$.

The main ideas in the rounding scheme are the consolidation of users at the cache location using min-cost flow, the cutting plane equation to guarantee that the rounding of users does not blow up capacity constraints and the bounding of the number of users left over after the randomized rounding stage.

2.3 Page-Placement Without Replication

If we impose the additional constraint that each page can be cached exactly once, we can obtain a constant factor approximation for Page-Placement. We can formulate this as an extension of the Generalized Assignment Problem(GAP) [17] and obtain the following theorem.

Theorem 3. *We can solve* Page-Placement *without page replication to obtain the optimum latency, provided the packing constraints are violated by a factor of* 4.

Proof. Refer Appendix A. □

3 The Cache-Placement Problem

We consider the alternate problem where we need to select locations for caches on the network. We are given a set P of pages and a set L of possible cache locations. Each user in the network requests a page. We must choose caches to open, assign pages to open caches, and then assign users to caches which are open and contain the requested page. We must guarantee that the total number of pages assigned to any open cache is at most cap_p and the total number of users assigned to any open cache is at most cap_u. We wish to minimize the sum over users of the distance from user to assigned cache, plus the sum over open caches of the cost of opening the cache. We call this problem Cache-Placement. Our algorithm obtains a constant-approximation.

Intuitively, this problem is easier than Page-Placement because we are allowed to place multiple caches at a single location. This means we can overflow capacity in a few locations by a lot, instead of having to make sure no location overflows capacity by more than a log factor.

3.1 Algorithm

We define x to be the cost of placing a cache, and $d(u, c)$ to be the distance from user u to location c. We let p_u represent the page requested by user u. We define O_c to be the number of caches we open at location c, $I_c(p)$ is one if page p is placed at location c, and $G_c(u)$ is one if user u is satisfied from location c. We solve the linear relaxation of the following integer program:

$$\text{Minimize} \sum_u \sum_c G_c(u) d(u,c) + \sum_c x O_c$$

$$\begin{array}{rcll} G_c(u) & \leq & I_c(p_u) & \forall u \in U, c \in C \\ I_c(p) & \leq & O_c & \forall p \in P, c \in C \\ \sum_c G_c(u) & = & 1 & \forall u \in U \\ \sum_p I_c(p) & \leq & cap_p O_c & \forall c \in C \\ \sum_u G_c(u) & \leq & cap_u O_c & \forall c \in C \\ G_c(u), I_c(p) & \in & \{0,1\} & \forall u \in U, p \in P, c \in C \\ O_c & \in & \mathbf{Z}^+ \cup \{0\} & \forall c \in C \end{array}$$

If we were to restrict all variables to be integer, then the optimal solution of the integer program is equal to the optimal problem solution. Suppose our fractional solution has total distance $d = \sum_u \sum_c G_c(u) d(u,c)$ and opens $k = \sum_c O_c$ caches. We know that $d + xk \leq OPT$.

It is useful to have comparable distances between a user and the caches from which his demand is satisfied. We describe a rounding scheme which can guarantee such a condition. We use the filtering technique of Lin and Vitter [10].

Lemma 7. *Given a fractional solution with k caches and total distance d, we can produce a new fractional solution which guarantees that if $d(u,c) > \alpha \sum_c G_c(u) d(u,c)$, then $G'_c(u) = 0$. This new solution guarantees that $O'_c \geq O_c$ for all $c \in L$. The new fractional solution uses at most $\frac{\alpha}{\alpha - 1} k$ caches and has total distance at most d.*

Proof. Refer Appendix B. □

We now explain how to round a fractional solution to guarantee that the O_c variables are all integral, while losing at most a constant factor on the cost function.

Lemma 8. *Suppose we have a fractional solution which guarantees that if $G_c(u) > 0$ then $d(u,c) \leq \alpha f(u)$ (for some function f), we can produce a new fractional solution in which all O_c are either zero or at least one. This solution uses at most 2 times the number of caches in the given solution, and guarantees that if $G_c(u) > 0$ then $d(u,c) \leq 3\alpha f(u)$. This of course also guarantees total distance at most $3\alpha \sum_u f(u)$.*

Proof. We consider user u_0 with the minimum value of $f(u_0)$. Some fraction of this user's demand is satisfied by caches which are less than half open ($\sum_{c:O_c<0.5} G_c(u_0)$). If this fraction is less than one half, we set all those $G_c(u_0)$ to zero. This can only decrease the total distance; note that the user still has $\sum_c G_c(u_0) \geq 0.5$; all other LP equations still hold. Otherwise, we select a cache c_0 such that $d(u_0, c_0)$ is minimum. We close every cache $c \neq c_0$ with $O_c < 0.5$ and $G_c(u_0) > 0$. The pages contained in these caches and the users satisfied by the caches are moved to c_0. We thus set $G_{c_0}(u) = \sum_{c:closed} G_c(u) + G_{c_0}(u)$ for all u, and $I_{c_0}(p) = \sum_{c:closed} I_c(p) + I_{c_0}(p)$ for all p. In order to accommodate the increased number of pages and users, we also set $O_{c_0} = \sum_{c:closed} O_c + O_{c_0}$. Notice that $O_{c_0} \geq \sum_{c:O_c<0.5} G_c(u_0) \geq 0.5$. This modified solution still satisfies all LP equations which were previously satisfied. The total $\sum_c O_c$ remains unchanged. The total distance for the new solution has increased. However, each u, c pair which was moved to cache c_0 had its distance increased by at most the sum of the distance from u_0 to the original cache with the distance from u_0 to the c_0. This is an increase of at most $2\alpha f(u_0)$. Since we selected the user with minimum $f(u_0)$, this means that no path increases by more than $2\alpha f(u)$ for the relevant user u.

We continue this process for each user, in order from least $f(u)$ to most. No path is ever changed more than once (after one change, it goes to a cache which is at least half open). Moving a path

increases its length by at most $2\alpha f(u)$. When we drop $G_c(u)$ values to zero, we note that the paths we keep have length at most $3\alpha f(u)$. We double the values of $I_c(p)$ and O_c in order to scale up the $G_c(u)$ values to sum to one. Our final solution satisfies all equations with total distance at most $3\alpha \sum_u f(u)$ and at most 2 times the caches of the previously given fractional solution. Suppose we have some cache location with $O_c < 1$. It follows that before doubling, we had $O_c < 0.5$. If any user has $G_c(u) > 0$, then we would have either set $G_c(u) = 0$ or else $O_c > 0.5$ would have been guaranteed. It follows that $G_c(u) = 0$ for all users, so we may as well close the cache and set $O_c = 0$. □

We now have a solution where the O_c are at least one, but the variables need not be integer. We show how to round to get O_c and $I_c(p)$ variables to be integers.

Lemma 9. *Given a fractional solution which guarantees that all O_c are zero or at least one, and that if $G_c(u) > 0$ then $d(u,c) \leq 3\alpha f(u)$, we can produce a new solution which satisfies all equations except $\sum_u G_c(u) \leq cap_u O_c$ and in which all O_c and $I_c(p)$ are integer. This solution uses at most 2 times the number of caches in the previous solution and has total distance at most $9\alpha \sum_u f(u)$.*

Proof. We consider user u_0 with the minimum value of $f(u_0)$. We let p_0 be the page requested by this user. If there exists a cache c_0 with $G_{c_0}(u_0) > 0$ and $I_{c_0}(p_0) = 1$, then we set $G_{c_0}(u_0) = 1$ and reduce $G_c(u_0)$ to zero for all other caches $c \neq c_0$. Otherwise, we choose c_0 to be the closest cache to user u_0. We remove all partial copies of page p_0 from caches with $G_c(u_0) > 0$, setting $I_c(p_0) = 0$ at these caches. We set $G_{c_0}(u_0) = I_{c_0}(p_0) = 1$. We also move all the other users which were retrieving page p_0 from one of the caches which no longer has the page, so that they are retrieving p_0 from c_0. This step may increase the distance along those paths. However, these other users have $f(u) \geq f(u_0)$, so the maximum possible increase of the distance is bounded by $6\alpha f(u)$.

We continue this process for each user, from least $f(u)$ to most. No path is ever changed more than once (after one change, it goes to a cache which has the full page). Moving a path increases its length by at most $6\alpha f(u)$. When we drop $G_c(u)$ values to zero, we note that the path we keep has length at most $9\alpha f(u)$. By setting O_c to the sum over pages of $I_c(p)/cap_p$, we guarantee that pages don't overflow bounds. Since the total of $\sum_c \sum_p I_c(p)$ has not changed, we know that this solution does not increase the number of caches. However, we still may not have satisfied $I_c(p) \leq O_c$ and we still have fractional O_c values. We round up all fractions to correct this problem. This introduces at most one extra cache at each location where the previous solution had $O_c > 0$. However, the previous solution had $O_c \geq 1$ in these locations, so we end up at most doubling the previous number of caches. Our final solution satisfies all equations except $\sum_u G_c(u) \leq cap_u O_c$ with total distance at most $9\alpha \sum_u f(u)$ and at most 2 times the caches of the previously given fractional solution. □

Combining the rounding schemes given above, we can obtain an integer solution with cost at most a constant factor more than the fractional, while guaranteeing O_c and $I_c(p)$ variables are integral. We show how to finish the rounding by guaranteeing integer $G_c(u)$.

Theorem 4. *We can find a constant-approximation to* Cache-Placement.

Proof. We apply the rounding techniques of lemma 7, lemma 8, and lemma 9 in order. This produces a solution with at most $4\frac{\alpha}{\alpha-1}k$ caches and total distance at most $9\alpha d$. However, we failed to satisfy the equation $\sum_u G_c(u) \leq cap_u O_c$. We notice that the original fractional solution must have had $k \geq |U|/cap_u$. We set $G_c(u) = 1$ for the cache c which minimizes $d(u,c)$ subject to $I_c(p_u) = 1$. We can introduce at most k additional caches to cover for the overflows of cap_u. Our final solution, satisfying all equations, has at most $(1 + 4\frac{\alpha}{\alpha-1})k$ caches and total distance at most $9\alpha d$. We set the two approximation ratios to be equal and solve for $\alpha = \frac{14+\sqrt{160}}{18}$. This yields a 13.325 approximation to OPT.

□

3.2 CACHE-PLACEMENT without Page Replication

Using similar techniques to those used in Section 2.3, but with a capacitated facility location algorithm [9] instead of GAP, we can obtain an 8-approximation to the problem when each page is cached exactly once.

4 The MAXIMUM-LATENCY Problem

In this variant of PAGE-PLACEMENT, we are given C caches placed at fixed locations in a network, each of which can hold cap_p pages. There is no user bound on the caches. There are N users who wish to access one or more of P distinct pages. Page p has maximum latency of access L_p. We wish to cache the pages so that any user accessing page p sees a latency of at most L_p. As before, we assume that each user has demand for only one page. We call this problem MAXIMUM-LATENCY.

We model this problem in two stages. In the first stage, we open facilities for each page and route the user demands to the facilities. In the next stage, we route the facilities to the C caches.

Let $M(p,i)$ denote whether we open a facility for page p at location $i \in F$. If we open a facility here, let us denote it as (p,i). Let $G_i(u)$ denote whether user u accesses its page p_u from facility (p_u,i). Let $I_c(p,i)$ denote whether we pack (p,i) in cache $c \in F$. Let p_u denote the page accessed by user u.

We can write an integer program for MAXIMUM-LATENCY as follows:

$$\begin{array}{rcll}
\sum_{i=1}^{F} G_i(u) & = & 1 & \forall u \\
G_i(u) & \leq & M(p_u,i) & \forall u,i \\
d(i,u) > L_{p_u} & \Rightarrow & G_i(u) = 0 & \forall u,i \\
 & & & \\
\sum_{c=1}^{F} I_c(p,i) & \geq & M(p,i) & \forall i,p \\
\sum_{p=1}^{P} \sum_{i=1}^{F} I_c(p,i) & \leq & cap_p & \forall c \\
d(i,c) > L_p & \Rightarrow & I_c(p,i) = 0 & \forall p,i,c \\
 & & & \\
G_i(u), M(p,i), I_c(p,i) & \in & \{0,1\} &
\end{array}$$

Note that the optimal solution is a feasible solution of this integer program where $M(p,c) = I_c(p,c) = 1$ if page p is in cache c, and $G_c(u) = 1$ if user u accesses page $p = p_u$ from cache c.

We first solve the LP relaxation of the integer program to obtain a fractional solution. Let us call the first three constraints the facility location constraints and the next three constraints the assignment constraints. We now round $G_i(u)$ and $M(p,i)$ as a facility location problem, and then round the $I_c(p,i)$ as a GAP problem [17].

The first stage rounding scheme uses the facility location constraints and is outlined below:

- Pick any fractional $G_i(u)$, and round up $G_i(u)$ and the corresponding $M(p,i)$ to 1, where $p = p_u$.
- For all other facilities q from which user u fractionally obtains page p, set $M(p,q)$ to 0. If user l had $G_q(l) > 0$, set $G_i(l) = 1$ and $G_q(l) = 0$.
- If any $G_i(u)$ remains un-rounded, goto step 1.

In [18], it is shown that this rounding scheme guarantees that the latency in user u going to facility (p,i) is no more than $3L_p$. Furthermore, the total fractional value of all facilities (p,q), including (p,i) in step 2 of the algorithm is at least 1, guaranteeing that we open no more than the optimum number of facilities per page.

The assignment constraints effectively pack the fractional $M(p,i)$ into the caches. Note that the first stage rounding transfers the fractional value $M(p,q)$ at facility (p,q) to a value of one at facility (p,i) in step 2.

We set $I_c(p,i) = \sum_q I_c(p,q) + I_c(p,i)$ and $I_c(p,q) = 0$ for all (p,q) rounded down to 0 in step 2. The resulting values form a feasible solution to the assignment constraints, provided that the latency bound for page p is $3L_p$ instead of L_p.

Note now that the assignment constraints are just a GAP problem. Demand point (p,i) has size 1, and the packing capacities are cap_p. This can be rounded using the technique in [17] so that the cache size is increased by a factor of 2 and the latency in this stage $3L_p$ for page p. We therefore obtain the following theorem:

Theorem 5. *We can solve* Maximum-Latency *so that the latency for page p is no more than $6L_p$ while increasing the cache size by a factor of 2.*

We note that for the Cache-Placement version of this problem, the techniques in Section 3 give a 4 approximation on the cache placement cost, provided the latency bound for page p is $9L_p$.

5 Extensions

In the previous sections we ignored many parameters for simplicity. We can easily extend the algorithms mentioned to handle different page sizes, as well as capacity constraints based on user demands. The approximation ratios remain the same. Furthermore, if request patterns vary temporally, we can solve one instance for each time period where the pattern is fixed (say for example, one instance for accesses during mornings, one for weekends, etc).

5.1 Update Latency

In addition to ensuring that users see small latency in page access, many pages need to have small latency of update. To model this, we assume we have a server for every page which updates the copies of the page in the caches. Page p has a latency of update J_p. We have to place the pages so that both the user latency bounds and the update latency are satisfied.

It is an easy exercise to see that the approximation bounds for Page-Placement, Cache-Placement and Maximum-Latency are the same with this additional constraint. The server latency bound J_p for page p can be exactly satisfied. The basic idea is to have the additional constraint that if the distance from the server to the cached copy of page p is more than J_p, then $I_c(p) = 0$.

5.2 Economies of Scale

In Appendix C, we present an algorithm for Cache-Placement when we have caches of different capacities and costs, and the cost grows sub-linearly with capacity.

6 Open Problems

Many interesting problems arise from the caching framework described above. One direction is to ask if there is a lower bound on the blowup in cache capacity required in Page-Placement, or if we can algorithmically obtain a constant blowup. Another interesting direction is to incorporate bandwidth constraints on the links into the model. Finally, from an implementation point of view, it would be interesting to come up with more practical schemes than the ones mentioned here which are based on solving linear programs.

References

[1] S. Albers, S. Arora, and S. Khanna. Page replacement for general caching problems. *Proceedings of Tenth Annual SIAM-ACM Symposium on Discrete Algorithms*, 1999.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. *Proceedings of INFOCOM*, 1999.

[3] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location and k-median problems. *Proceedings of the Twenty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[4] Moses Charikar, Sudipto Guha, David B. Shmoys, and Éva Tardos. A constant factor approximation algorithm for the k-median problem. *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[5] C. Chiang, M. Liu, and M. Muller. Caching neighbourhood protocol: A foundation for building dynamic web caching hierarchies with proxy servers. *Proceedings of International Conference on Parallel Processing*, 1999.

[6] Edith Cohen and Haim Kaplan. Exploiting regularities in web traffic patterns for cache replacement. *Proceedings of Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[7] Dorit Hochbaum and David Shmoys. A best possible heuristic for the k-center problem. *Math of Operations Research*, 10(2):180–184, 1985.

[8] S. Irani. Page replacement with multi-sized pages and applications to web caching. *Proceedings of Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.

[9] Kamal Jain and Vijay Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *Proceedings of the Twenty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[10] J.-H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violations. *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, 1992.

[11] B. Maggs, F. Meyer auf der Heide, B. Vocking, and M. Westermann. Exploiting locality for networks of limited bandwidth. *Proceedings of Thirty-Eighth Annual IEEE Symposium on Foundations of Computer Science*, 1997.

[12] F. Meyer auf der Heide, B. Vocking, and M. Westermann. Caching in networks. *Proceedings of Eleventh Annual SIAM-ACM Symposium on Discrete Algorithms*, 2000.

[13] R. Motwani, S. Naor, and P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[14] P. Raghavan and C.D. Tompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 37:365–374, 1987.

[15] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *JCSS*, 37:130–143, 1988.

[16] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Buy-at-bulk network design: Approximating the single-sink edge installation problem. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 619–628, 1997.

[17] David B. Shmoys and Éva Tardos. Scheduling unrelated machines with costs. *Proceedings of the Fourth Annual SIAM-ACM Symposium on Discrete Algorithms*, pages 448–454, 1993.

[18] David B. Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[19] Aravind Srinivasan. Improved approximations for packing and covering problems. *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, 1995.

[20] I. Tatarinov, A. Rousskov, and V. Soloviev. Static caching of web servers. *Proceedings of the Sixth International Conference on Computer Communications and Networks*, 1997.

[21] J. Zhang, R. Izmailov, D. Reininger, and M. Ott. Web caching framework: Analytical models and beyond. *IEEE workshop on Internet Applications*, 1999.

A Proofs for Section 2

A.1 Lemma 1

For a particular user, the $G_c^F(u)$ variables are independent from cache to cache. Their sum is a sum of independent $0-1$ Bernoulli variables. If for some cache, $G_c^S(u) = 1$, then we know $I_c^S(p_u) = 1$ and after rounding we will guarantee that the sum over caches is at least one. Assuming $G_c^S(u)$ is not equal to one for any cache, we know that $G_c^F(u) = I_c^R(u)G_c^R(u) = 1$ with probability $2(\log|C|)G_c^D(u)$. This means that either the sum is always at least one or else the sum has expected value at least $\mu = 2\log|C|$. We apply Chernoff bounds to get:

$$Pr[\sum_c G_c^F(u) < (1-\delta)\mu] < exp(-\mu\delta^2/2)$$

where in this case $\mu = 2\log|C|$ and $\delta = 1 - (1/2\log|C|)$. Solving the inequality yields:

$$Pr[\sum_c G_c^F(u) < 1] < exp(-\mu\delta^2/2) < e/|C|$$

A.2 Lemma 3

The sum $\sum_p I_c^F(p)$ is a sum of independent $0-1$ Bernoulli variables. The expected value of this sum is at most $(2\log|C|)cap_p$. Using Chernoff bounds:

$$Pr[\sum_p I_c^F(p) > (6\log|C|)cap_p] < (e^2/27)^{2\log|C|} < 1/|C|^2.$$

A.3 Theorem 3

For page p, let $W(p)$ denote the total user size for that page and $d(p,c)$ denote the total latency of accessing this page if it is cached in cache c. If $I_c(p)$ denotes whether we cache page p in cache c, then the resulting integer program is shown below.

$$\text{Min.} \sum_{p=1}^{P}\sum_{c=1}^{C} W(p)d(p,c)I_c(p)$$

$$\begin{array}{rcll} \sum_{c=1}^{C} I_c(p) & = & 1 & \forall p \\ \sum_{p=1}^{P} I_c(p) & \leq & cap_p & \forall c \\ \sum_{p=1}^{P} W(p)I_c(p) & \leq & cap_u & \forall c \\ I_c(p) & \in & \{0,1\} & \end{array}$$

Let $M(p) = \frac{1}{cap_p} + \frac{W(p)}{cap_u}$. We replace the two packing constraints for each cache c by the following single constraint:

$$\sum_{i=1}^{P} M(p)I_c(p) \leq 2 \quad \forall c$$

Note that the optimum solution satisfies the new packing constraint. With this new packing constraint, the problem reduces to the Generalized Assignment Problem (GAP) [17], and we can round the cost optimally, while violating the packing constraint by a factor of 2. This means that the original packing constraints are satisfied, provided we blow up the cap_u and the cap_p by a factor of 4.

B Proofs for Section 3

B.1 Lemma 7

For every cache location c and user u, we set $G'_c(u) = 0$ if $d(u,c) > \alpha \sum_c G_c(u)d(u,c)$. Otherwise we set $G'_c(u) = G_c(u)$. Notice that we have $\sum_c G'_c(u) \geq 1 - \frac{1}{\alpha}$. We next scale $G'_c(u)$ by multiplying it by $1/\sum_c G'_c(u)$. Since we have only discarded the longest-distance caches, the total distance for this new set of $G'_c(u)$ cannot be more than the distance in the original fractional solution. However, we may have $G'_c(u) > I_c(p_u)$. Since no $G'_c(u)$ exceeds the old (pre-rounding) value by more than a factor of $\frac{\alpha}{\alpha-1}$, we can set $I'_c(p) = min(1, \frac{\alpha}{\alpha-1} I_c(p))$ and $O'_c = \frac{\alpha}{\alpha-1} O_c$. This increases the number of caches to $\frac{\alpha}{\alpha-1}k$. All the linear program constraints are satisfied.

C Utilizing Economies of Scale for Cache-Placement

As in the Cache-Placement problem, we are given a set P of pages and a set L of possible cache locations. Each user in the network requests a page. We must choose caches to open, assign pages to open caches, and then assign users to caches which are open and contain the requested page.

We are given caches of K types. A cache of type i has cost x_i, page capacity cap_{pi} and user capacity cap_{ui}.

We must guarantee that the total number of pages assigned to any open cache of type i is at most cap_{pi} and the total number of users assigned to it is at most cap_{ui}. We wish to minimize the sum over users of the distance from user to assigned cache, plus the sum over open caches of the cost of opening the cache. Our algorithm obtains a $O(K)$ approximation.

We create K copies of each location c, one for each cache type, and treat them as distinct locations. Note that the cache cost and capacity depend on location now. We first solve a linear relaxation of the following integer program (the variables have the same meaning as before):

$$\text{Minimize} \sum_u \sum_c G_c(u)d(u,c) + \sum_c x_c O_c$$

$$\begin{array}{rcll}
G_c(u) & \leq & I_c(p_u) & \forall u \in U, c \in C \\
I_c(p) & \leq & O_c & \forall p \in P, c \in C \\
\sum_c G_c(u) & = & 1 & \forall u \in U \\
\sum_p I_c(p) & \leq & cap_{pc} O_c & \forall c \in C \\
\sum_u G_c(u) & \leq & cap_{uc} O_c & \forall c \in C \\
G_c(u), I_c(p) & \in & \{0,1\} & \forall u \in U, p \in P, c \in C \\
O_c & \in & \mathbf{Z}^+ \cup \{0\} & \forall c \in C
\end{array}$$

The rounding scheme follows the scheme for Cache-Placement, but with some modifications. Firstly, for each user, we identify the cache type that sinks in most of its demand, and send all its demand there. This increases the cost of the solution by at most a factor K. Now, no user goes to more than one cache type.

For each cache type, we round the fractional solution separately using the rounding scheme for Cache-Placement. The resulting integer solution is within a factor of 13.325 from the fractional solution before rounding. We therefore have a $O(K)$ approximation for the problem.