versão impressa ISSN 0101-7438 / versão online ISSN 1678-5142

WEB CACHE LOCATION

Brian Boffey *
Department of Mathematical Sciences
University of Liverpool
Liverpool, L69 7ZL – UK
boffey@liv.ac.uk

Pirooz Saeidi
Department of Computer Science
Staffordshire University
Stafford, ST18 0DG – UK
ps1@staffs.ac.uk

* *Corresponding author*/autor para quem as correspondências devem ser encaminhadas

Recebido em 06/2003; aceito em 03/2004
Received June 2003; accepted March 2004

Abstract

Stress placed on network infrastructure by the popularity of the World Wide Web may be partially relieved by keeping multiple copies of Web documents at geographically dispersed locations. In particular, use of proxy caches and replication provide a means of storing information 'nearer to end users'. This paper concentrates on the locational aspects of Web caching giving both an overview, from an operational research point of view, of existing research and putting forward avenues for possible further research. This area of research is in its infancy and the emphasis will be on themes and trends rather than on algorithm construction. Finally, Web caching problems are briefly related to referral systems more generally.

Keywords: proxy cache; replication; location.

1. Introduction

The World Wide Web (or simply Web) is extremely popular. It has experienced an extraordinary growth rate and, with newer applications such as the resource intensive video-on-demand, yet further growth may be expected. This growth has placed considerable stress on the network infrastructure giving rise to the need for a scalable way in which it may be upgraded in order to keep latencies (response times) within a tolerable limit.

One general way in which Web performance may be improved is to keep multiple copies of Web documents at geographically dispersed locations. Then, on average, retrieval of a 'nearby' copy of a document results in a reduction in latency, network congestion and server load. This dispersal may be achieved by suitable location of *Web proxy caches* (which we term simply 'proxy caches', or even just 'caches') or by means of *replication*. Replication can involve replicating the server which may be modelled as the well-known *Simple Location* problem or the *p-Median* problem (Mirchandani & Francis, 1990) or, if server load is a concern, the capacitated versions of these. However, replication may mean the use of *reverse proxies* (see later) which retain only copies of the more popular documents. In this case there is also an *Assignment Problem* for deciding which objects (documents) to store at which site. A reverse proxy connects one server with very many users Internet wide and represents a *server-side* solution.

Unlike replication, (forward) proxy caches represent a *client-side* solution in which copies of popular documents are 'pulled' towards users, with each cache connecting a group of users to many servers. It is a viable approach since the request rate for documents is empirically found to approximate a Zipf distribution (Baentsch *et al.*, 1997b) for which

$$\text{P(request for } i\text{-th most popular document)} = C/i^{\alpha}$$

where $C = (\sum_i i^{-\alpha})^{-1}$ is just a normalisation constant and α is a parameter whose value is typically around $0.7 - 0.8$, although values outside this range are not uncommon. This distribution has the consequence that around 40% of requests may be met by a proxy cache storing a very small fraction of one per cent of all the documents on the Web using a cache with a few Gigabytes of memory.

Of course, caching in general is not new and is well established where memory hierarchies are involved. What is new about Web caching is that the proxy caches are geographically dispersed over network(s) thus giving rise to a location problem: where should cache(s) be located so as to optimise some objective (Krishnan *et al.*, 2000)? Very little research has been published regarding this, Krishnan *et al.* (2000) being a notable contribution. What can definitely be stated is that there are many practical complications regarding actual networks, not least as regards the protocols involved. Because of this it does not seem particularly helpful to devote a lot of effort developing exact algorithms for the general theoretical models. Rather, we shall here consider simplified models to investigate general themes and trends. To a large extent this will be achieved by means of illustrative examples. As well as giving a brief overview of existing research from an operational researcher's point of view, some possibilities for further research are also given.

For an excellent general reference on Web caching and replication the interested may consult the text by Rabinovich & Spatscheck (2002). Other relevant references are Luotonen (1998) and Wessels (2001). For an account of relevant models in locational analysis, see Mirchandani & Francis (1990), especially the first three chapters.

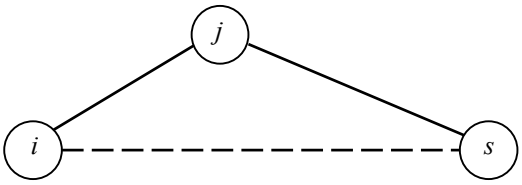

Figure 1 – Client i may be assigned directly to server s (broken line), or assigned to the cache at j which in turn is connected to the server (full lines).

2. Proxy Location Problems and Previous Research

Before proceeding further, it is convenient to introduce some terminology relating to the area of computer networks. A *user* works at a computer running a *browser* (e.g. Netscape Navigator or Microsoft Explorer). Requests can be sent to a remote computer, the *origin server*, which holds the desired information. This information may, or may not be, in the form of a Web document and will be referred to as an *object*.

A *cache* is a place for storing information. The basic idea of a *(forward) Web proxy cache* is that a request for an object is sent from the client to a specified 'nearby' cache, where there is a check to see whether the cache holds the desired object – if it does, then the object is returned directly to the client, otherwise a *cache miss* occurs and the request is forwarded on to the origin server with the object being returned to the client via the cache **which also keeps a copy of the object**. The proportion of requests which can be satisfied by the cache is called the cache *hit rate*. This situation is portrayed in Figure 1. It is interesting to note at this point that the problem of minimising latency is abstractly similar to *referral systems* more generally. Thus, for example, the server may correspond to a hospital and the caches to local clinics which will refer patients on to the hospital if more specialist treatment is required (see eg. Narula & Ogbu, 1985; Gerrard & Church, 1994 and Galvão *et al.*, 2002).

2.1 The basic problems

A mathematical formulation can now be given of the *Minimal Latency Problem* (or MLP) in which there is one fixed server and precisely i **equal sized** caches are to be located. It is:

MLP: minimise $\tau = \Sigma_{i\in I}\Sigma_{j\in C^+} f_i[d_{ij} + (1-\alpha_i)d_{js}]x_{ij} = \Sigma_{i\in I}\Sigma_{j\in C^+} f_i c_{ij} x_{ij}$

subject to

$$\Sigma_{j\in C^+} x_{ij} = 1, \qquad \forall i \in I \tag{1}$$

$$x_{ij} \le y_j, \qquad \forall i \in I,\ \forall j \in C^+ \tag{2}$$

$$\Sigma_{j\in C^+} y_j = p+1, \tag{3}$$

$$x_{ij}, y_j \in \{0,1\}, \qquad \forall i \in I,\ \forall j \in C^+ \tag{4}$$

$$y_s = 1 \tag{5}$$

where

τ is the total latency per unit time

s is the (site of) the server

I is the set of client sites – this includes all nodes other than s

$C \subset I$ is the set of possible cache sites

$C^+ = C \cup \{s\}$

f_i is the demand per unit time from client I

d_{ij} is the distance between $i \in I$ and $j \in C^+$

α_i is the hit rate of a cache for demand from $i \in I$

c_{ij} is written in place of $d_{ij} + (1-\alpha_i)d_{js}$

$x_{ij} = 1$ if client $i \in I$ is assigned to cache (or server) at $j \in C^+$; $x_{ij} = 0$ otherwise.
(These are the *assignment* variables and $\boldsymbol{x} = (x_{ij})$ is the *assignment*.)

$y_j = 1$ if there is a cache at $j \in C$; $y_j = 0$ otherwise

p is the number of caches to be located.

Constraints (1) state that each client must be assigned to precisely one cache or else to the server. Constraints in set (2) are the logical constraints that a client i cannot be assigned to j unless there is a cache at j or else $j = s$ (the server). Exactly p caches must be located which, together with the server at s, gives the term $p+1$ on the right hand side of (3). (4) are the usual binary constraints and (5) expresses the fact that the server is fixed at s. Actually, the latency includes both request travel time and travel time for retrieved objects, with the latter dominating. Since distances d_{ij} are symmetric the expression for τ can refer to traffic in either direction.

It is readily seen that the feasible set of problem MLP is that for a $(p+1)$-median problem in which one facility location (the server's) is fixed ($y_s = 1$). A simple transformation then shows that this is just the feasible set of a p-median problem in which the p facilities (the caches) are to be located at sites in C. In the special, but unrealistic, case of *perfect* caching (i.e. $\alpha_i = 1, \ \forall i \in I$) the 'forwarding terms' $(1-\alpha_i)d_{js}x_{ij}$ are all zero and the objective also takes the usual form for a p-median problem. The equivalence of this special case to a p-median problem establishes that MLP belongs to the class *NP* of computationally difficult problems (Garey & Johnson, 1979 and Mirchandani & Francis, 1990).

In practice, caching is far from perfect since: the caches do not have sufficient memory to store all required objects simultaneously; and, not all objects are cacheable anyway. Another deficiency is that browsers need to be *explicitly assigned* to the optimal cache (or server) and this might not always be done satisfactorily. However, there is a form of caching, called *transparent caching*, for which a p-median equivalence can be demonstrated more naturally.

With transparent caching, all requests are dispatched to the server. When no cache is encountered by the request the server returns the object in the normal way. On the other

hand, if such a request passes through a node with a cache, then the request is *intercepted* and a check made to see whether the cache has the required object. If it has, it responds as though it were the server (hence the term 'transparent') returning a copy of the object directly to the client; otherwise, the request is forwarded on to the server which then deals with the request. With transparent caching the assignment is *implicit* with assignment being to the first cache encountered or, if there are no caches on the shortest path to *s*, assignment is to *s* itself.

If a transparent cache at *j* is encountered when a request is sent from *i* to server at *s* then *j* is on a shortest path from *i* to *s* (with respect to the routing table metric) and so $d_{ij} + d_{js} = d_{is}$. As a result of this the objective function can be simplified.

Theorem 1 If all caches are transparent and of equal size, then MLP may be solved by minimising $\Sigma_{i\in I}\Sigma_{j\in C^+} f_i d_{ij} x_{ij}$.

Proof Since $c_{ij} = d_{ij} + (1-\alpha_i)d_{js}$:

(a) if $x_{ij} = 1,\ j = s$ then $c_{ij} = d_{ij} + 0 = d_{ij} = (1-\alpha_i)d_{ij} + \alpha_i d_{ij} = (1-\alpha_i)d_{is} + \alpha_i d_{ij}$

(b) if $x_{ij} = 1,\ j \neq s$ then $c_{ij} = d_{ij} + (1-\alpha_i)d_{js} = (1-\alpha_i)(d_{ij} + d_{js}) + \alpha_i d_{ij} = (1-\alpha_i)d_{is} + \alpha_i d_{ij}$

The objective function can hence be modified to

$$\Sigma_{i\in I}\Sigma_{j\in C^+} f_i c_{ij} x_{ij} = \Sigma_{i\in I}\Sigma_{j\in C^+} (1-\alpha_i) f_i d_{is} x_{ij} + \Sigma_{i\in I}\Sigma_{j\in C^+} \alpha_i f_i d_{ij} x_{ij}$$

But the first term is just a constant since it is independent of cache location ($\Sigma_{j\in C^+} x_{ij} = 1$, $\forall i \in I$ by (1)) and so may be ignored as far as the minimisation is concerned. Also, without loss of generality, the factor α_i may be absorbed into f_i and the objective $\Sigma_{i\in I}\Sigma_{j\in C^+} f_i d_{ij} x_{ij}$ used.

This theorem means that we have again obtained the usual objective for a *p*-median problem. Since a cache is likely to have very many clients assigned to it, explicitly or implicitly, the average hit rate should not differ too much from those of other groups and so we shall proceed with hit rates depending only on cache size (i.e. for equal sized caches $\alpha_i = \alpha, \forall i$).

Example 1 The above ideas will now be illustrated by the very small example network in Figure 2 in which the demands at *i*, *j* and *k* are equal, say $f_i = f_j = f_k = 1$. A **single** cache (i.e. $p = 1$) with hit rate α is to be located at *i*, *j* or *k* so as to minimise the total expected delay in a request being serviced.

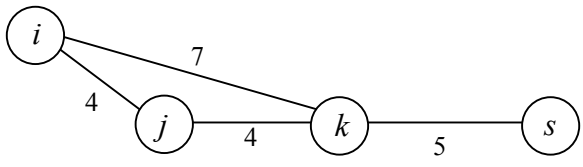

Figure 2 – A small example network. The length d_{ij} of each edge *i*-*j* is shown beside that edge.

Solution (a) First, we shall suppose that clients are assigned explicitly to the cache or the server. Suppose that the cache is at i then

$$\tau(i) = \min\{12,\ 0+12(1-\alpha)\} + \min\{9,\ 4+12(1-\alpha)\} + \min\{5, 7+12(1-\alpha)\}$$

where the three minima respectively correspond to demands from i, j and k. For each minimisation, the first argument refers to flow directed to s and the second to flow directed to the cache plus the expected flow forward from the cache to the server. That is,

$$\tau(i) = \begin{cases} 26-12\alpha, & 0 \le \alpha \le \frac{7}{12} \\ 33-24\alpha, & \frac{7}{12} \le \alpha \le 1 \end{cases} \quad \text{and} \quad \tau^{\#}(i) = \begin{cases} (c,s,s), & 0 \le \alpha \le \frac{7}{12} \\ (c,c,s), & \frac{7}{12} \le \alpha \le 1 \end{cases}$$

Here, $\tau^{\#}(i)$ denotes the vector of assignments of nodes; thus, for $\alpha > 7/12$, nodes i and j are assigned to the cache at i whereas node k is assigned directly to the *s*erver. The values of $\tau(j), \tau^{\#}(j), \tau(k)$ and $\tau^{\#}(k)$ are readily found to be

$$\tau(j) = \begin{cases} 26-9\alpha, & 0 \le \alpha \le \frac{1}{9} \\ 27-18\alpha, & \frac{1}{9} \le \alpha \le \frac{8}{9} \\ 35-27\alpha, & \frac{8}{9} \le \alpha \le 1 \end{cases} \quad \text{and} \quad \tau^{\#}(j) = \begin{cases} (s,c,s), & 0 \le \alpha \le \frac{1}{9} \\ (c,c,s), & \frac{1}{9} \le \alpha \le \frac{8}{9} \\ (c,c,c), & \frac{8}{9} \le \alpha \le 1 \end{cases}$$

$$\tau(k) = 26-15\alpha, \quad 0 \le \alpha \le 1 \quad \text{and} \quad \tau^{\#}(k) = (c,c,c), \quad 0 \le \alpha \le 1$$

from which the minimal value $\tau_{\min}$ of τ is

$$\tau_{\min} = \begin{cases} 26-15\alpha, \text{ cache at } k, & 0 \le \alpha \le \frac{1}{3} \\ 27-18\alpha, \text{ cache at } j, & \frac{1}{3} \le \alpha \le \frac{8}{9} \\ 35-27\alpha, \text{ cache at } j, & \frac{8}{9} \le \alpha \le 1 \end{cases} \quad \text{and} \quad \tau^{\#}_{\min} = \begin{cases} (c,c,c), & 0 \le \alpha \le \frac{1}{3} \\ (c,c,s), & \frac{1}{3} \le \alpha \le \frac{8}{9} \\ (c,c,c), & \frac{8}{9} \le \alpha \le 1 \end{cases}$$

It is thus seen to be optimal to have the cache at k or j, in the latter case the assignment of k being to the server or cache depending on whether α is greater than, or less than 8/9. Note also that for α = 1/3 the cache may equally well be placed at j or k.

Next we consider transparent caching. For this, shortest path routing to s is used so that link $i-j$ plays no part and hence may be removed. After a little calculation the solution is found to be

$$\tau(i) = 26-12\alpha, \quad 0 \le \alpha \le 1 \quad \text{and} \quad \tau^{\#}(i) = (c,s,s), \quad 0 \le \alpha \le 1$$
$$\tau(j) = 26-9\alpha, \quad 0 \le \alpha \le 1 \quad \text{and} \quad \tau^{\#}(j) = (s,c,s), \quad 0 \le \alpha \le 1$$
$$\tau(k) = 26-15\alpha, \quad 0 \le \alpha \le 1 \quad \text{and} \quad \tau^{\#}(k) = (c,c,c), \quad 0 \le \alpha \le 1$$

and it is clearly optimal to place the cache at k for all α.

The above example shows that it can be non-trivial to solve MLP for all $0 \le \alpha \le 1$. In practice however, a solution will be required for a single representative value of α, say $\alpha = 0.4$. In this case, with equal sized transparent caches MLP is effectively a p-median problem on a tree and so can be solved in polynomial time (Tamir, 1996; Krishnan *et al.*, 2000). In the general case, however, MLP is *NP*-hard even for a tree network (Krishnan *et al.*, 2000)!

Latency (or response time) is the measure of effectiveness of direct interest to the client (actually the human user) but it is only indirectly important to the 'system managers'. On the other hand, network congestion is of direct interest to the system but only of indirect interest to the client (user) in so far as it impacts on latency.

One measure of congestion appropriate when all links have the same capacity is the flow in a most congested link and it is sensible to try to minimise this. Since this objective is system oriented, it seems to make sense to use it with regard to transparent caching only. Server overload may be a problem but this is also connected to minimising congestion and so will not be considered further here. The *Minimal Congestion Problem* (or MCP) may now be formulated as

MCP: minimise σ

subject to constraints $(1)-(5)$, and

$$flow_{ij} \le \sigma, \qquad \forall i \in I,\ j \in I \cup \{s\} \tag{6}$$

where $flow_{ij}$ is the flow in link $i-j$. It is easily seen that, for the network of example 1, the optimal location for the cache is now at k for all $0 \le \alpha \le 1$ and that the minimal value of σ is $3-3\alpha$.

2.2 The uniform linear segment

Since the optimal solutions of MLP and MCP generally do not coincide, it is interesting to see how they differ. To gain a feeling for this we investigate the case of a linear chain with a single server and n client nodes each with the same demand. For n sufficiently large a chain may conveniently be approximated by a *uniform linear segment* (Figure 3) which, without loss of generality, may be assumed to extend from $x = 0$ to $x = 1$ (at the server) and with unit demand spread continuously over the unit interval (i.e. $\rho(\xi) = 1,\ 0 \le \xi \le 1$, where $\rho(\xi)$ is the demand density and $\int_0^1 \rho(\xi)\xi = 1$ as required).

Example 2 (Uniform linear segment) The optimal cache location for a single transparent cache is sought (see Figure 3). The results for the discrete case will be much the same and will not be given here (but see Krishnan *et al.*, 2000).

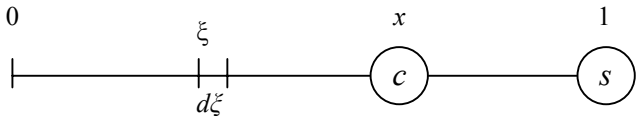

Figure 3 – A uniform linear segment. The numbers above the line are the respective distances from the left end of the segment.

Solution Denoting by $\tau(x)$ the total latency when the cache is at a distance x along the segment and considering all elements of demand $d\xi$ along the segment,

$$\tau(x) = \int_0^x (x-\xi)\,d\xi + (1-\alpha)(1-x)\int_0^x d\xi + \int_x^1 (1-\xi)\,d\xi = \frac{1}{2} - \alpha x(1-x)$$

where the first integral refers to flow arising from demand before the cache (i.e. from demand to the left of the cache in Figure 3), the second to demand from before the cache with regard to travel between cache and server, and the third to demand arising 'between the cache and server'. $\tau(x)$ is minimised at $x = 1/2$. Notice that for transparent caching, requests can only *travel to the right* (though of course the retrieved objects must travel to the left) but with explicit assignment a request is permitted to travel in either direction and the optimal cache location would be changed. (Note that the problem could have been solved slightly more easily by using the continuous equivalent of theorem 1 and treating this as a 2-median problem with the server being fixed.)

When total demand remains at 1 but is spread with density $\rho(\xi)$, $0 \le \xi \le 1$, where $\rho(x)$ is differentiable, then it may easily be shown that τ is minimised when

$$\int_0^x \rho(\xi) d\xi = (1-x)\rho(x), \tag{7}$$

which yields $x = 1/2$ when $\rho(\xi) = 1, \ \forall \xi$ agreeing with the result obtained above. Notice that, as for the uniform case, the optimal value of x does not depend on α.

Turning now to congestion minimisation, we note that σ will be equal to the minimum of the flows immediately prior to the cache and to the server; these are, respectively, x and $(1-\alpha)x + (1-x) = 1 - \alpha x$. The first of these (i.e. x) is monotonic increasing in x and $1 - \alpha x$ is monotonic decreasing. It follows immediately that minimum σ occurs when these two terms are equal, that is $x = 1 - \alpha x$, or

$$x = 1/(1+\alpha).$$

This time as α varies from zero to one, the optimal x varies from $1/2$ (the same as for MLP) to 1 (at the server). At $\alpha = 0.4$ (a typical value for a small cache), $x = 0.71$ and the solutions for MLP and MCP differ by over 20% of the segment length!

When total demand remains at 1 but is spread with density $\rho(\xi)$, $0 \le \xi \le 1$, then $\sigma(x)$ is minimised when $\int_0^x \rho(\xi) d\xi = 1/(1+\alpha)$, that is, when a proportion $1/(1+\alpha)$ of the total demand is 'to the left of the cache'.

One feature of the actual (discrete) case that is worth mentioning is that not all links need have the same capacity and this could have a major impact on congestion minimisation. What is important is the *proportion* of capacity of a link that is used rather than the flow in that link. Consequently, constraint (6) of MCP should be amended to

$$flow_{ij}/cap_{ij} \le \sigma, \quad \forall i \in I, \ \ j \in I \cup \{s\}$$

where cap_{ij} is the capacity of link $i - j$.

Since it is desirable to minimise both latency and congestion it is interesting to pose the *best compromise solution* problem (BCSP) which is to minimise $(1-w)\tau(x) + w\sigma(x), \ 0 \le w \le 1$. This problem is reminiscent of the *Centdian Problem* in locational analysis but there the analogue of $\sigma(x)$ would pull the optimal location towards the centre of the segment whereas here $\sigma(x)$ is pulling towards the server.

Some work has been done by Krishnan *et al.* (2000) on MLP. They obtained results for a uniform linear chain these being essentially rounded versions of those for a uniform linear segment and also provided an algorithm for a multiple server non-uniform linear chain. An efficient algorithm was presented for the general (discrete) problem of locating p equally sized caches on a tree in the case of a single server.

For congestion minimisation the authors are aware of no previous work and, to the best of their knowledge, the 'bicriterion' BCSP has not previously been posed.

2.3 Hierarchical cache location

In examples so far, only the placement of a single cache has been considered though in reality there will be multiple caches to be located. To study this latter situation it is convenient to revisit the continuous approximation for a uniform linear chain.

The hit rate of cache c_1 (for requests arising to the left of c_1 (see Figure 4)) is α, and the hit rate of cache c_2 for requests arising *between the caches* is β. But what is the relevant hit rate for cache c_2 for requests from the left of the first cache c_1 which have been forwarded on (because c_1 did not have the relevant objects)?

Caches c_1 and c_2 will have many objects in common since a copy of an object supplied by the server is dropped off at each transparent cache on the return journey. It follows that satisfaction of such requests at c_2 is **not** independent of satisfaction at c_1. In fact c_2 is likely to have almost all the objects stored in c_1 provided its capacity is at least that of c_1. With this in mind, Krishnan *et al.* (2000) imposed their so-called *Full Independence Assumption* under which a cache has an *effective hit rate* of zero after having encountered an **equal sized** cache previously. Rodriguez *et al.* (2001) extended this to the case of two successive caches on a shortest path to the server having hit rates of α and β where $\beta > \alpha$, the effective cache of the second (larger) cache now being $\beta - \alpha$. To account for the case of a larger cache being encountered first we further extend this slightly by assigning an effective hit rate of

$$\max(0,\ \beta - \alpha)$$

for the second encountered cache. (This extends in a natural way to three or more caches on a shortest path to the server.) Intuitively it seems preferable to have caches increasing in size towards the server and we now test this for a uniform linear segment.

Example 3 Caches c_1 and c_2 are, respectively, at distances x and y from the non-server end of a uniform linear segment and have hit rates α and β where $\beta > \alpha$. Determine optimal values of σ and τ under the conditions: (a) $y > x$; (b) $x > y$.

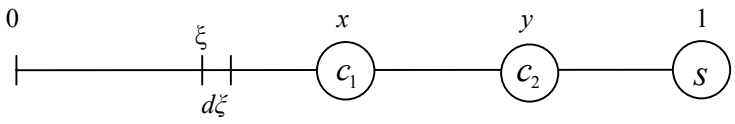

Figure 4 – A uniform linear segment with two caches at distances x and y from the left end of the segment. Caches c_1 and c_2 have hit rates α and β respectively.

Solution Referring to Figure 4 we investigate σ first.

(a) With $x < y$,

$$\sigma(x,y) = \max(x,\ (1-\alpha)x + (y-x),\ (1-\beta)x + (1-\beta)(y-x) + (1-y))$$
$$= \max(x,\ y-\alpha x,\ 1-\beta y)\ .$$

As in example 2, we set the three terms in the maximisation to be equal which leads to

$$x = S\ ,\qquad y = (1+\alpha)S\ ,\qquad \sigma_{\min} = S\ ,\qquad \text{where}\ \ S = (1+\beta+\alpha\beta)^{-1}$$

(b) With $y < x$,

$$\sigma(y,x) = \max(y,\ (1-\beta)y + (x-y),\ (1-\beta)y + (1-\alpha)(x-y) + (1-x))$$
$$= \max(y,\ x-\beta y,\ 1-\alpha x + (\alpha-\beta)y)\ .$$

Again setting the three terms in the maximisation to be equal gives

$$y = S\ ,\qquad x = (1+\beta)S\ ,\qquad \sigma_{\min} = S\ ,\qquad \text{where}\ \ S = (1+\beta+\alpha\beta)^{-1}$$

Although the **position** of the cache nearer to the server is affected the near-server cache stays in the same place and the minimal value of σ is **unchanged**. This indifference of σ to the order of the two caches is somewhat surprising and may be expected to be approximately the case for a linear chain. For a tree the situation is more complicated though it seems that, at least in some cases, it is preferable to have caches increasing in size towards the server.

For the latency τ we cannot now use theorem 1 since the caches are of different sizes.

(a) When $x < y$,

$$\tau(x,y) = \int_0^x (x-\xi)d\xi + (1-\alpha)(y-x)\int_0^x d\xi + (1-\beta)(1-y)\int_0^x d\xi$$
$$+\int_x^y (y-\xi)d\xi + (1-\beta)(1-y)\int_x^y d\xi + \int_y^1 (1-\xi)d\xi$$

where the first line gives the terms for traffic from the left of cache c_1 for the subsegments $(0,x)$, (x,y) and $(y,1)$ respectively. After some straightforward, but tedious manipulation, this simplifies to

$$\tau(x,y) = \frac{1}{2} - \beta y + \{\alpha x^2 - \alpha xy + \beta y^2\}\ .$$

Setting partial derivatives equal to zero yields

$$x = \beta/D\ ,\qquad y = 2\beta/D\ ,\qquad \text{where}\ \ D = 4\beta - \alpha \qquad (8)$$

from which

$$\tau_{\min} = \frac{1}{2} - \frac{\beta^2}{D}$$

(b) When $y < x$,

$$\tau(x,y) = \int_0^y (y-\xi)d\xi + (1-\beta)(x-y)\int_0^y d\xi + (1-\beta)(1-x)\int_0^y d\xi$$
$$+\int_y^x (x-\xi)d\xi + (1-\alpha)(1-x)\int_y^x d\xi + \int_x^1 (1-\xi)d\xi$$

where the first line again gives the terms for the traffic from the left of cache but this time for the three subsegments $(0,y)$, (y,x) and $(x,1)$. This simplifies to

$$\tau(x,y) = \frac{1}{2} - \alpha x + (\alpha-\beta)y + \{\alpha x^2 - \alpha xy + \beta y^2\}$$

leading to

$$y = (2\beta - \alpha)/D, \qquad x = (3\beta - \alpha)/D, \qquad \text{where } D = 4\beta - \alpha \tag{9}$$

and hence

$$\tau_{\min} = \frac{1}{2} - \frac{\beta^2}{D}$$

Thus we have shown that for τ it is immaterial which way round the two caches are. However, it should not be expected to carry through to trees where it would seem sensible to have the larger caches towards the server.

Finally, we present two general results for the location of multiple identical sized caches on a uniform linear segment.

Theorem 2 n identical sized caches are to be located on a uniform linear segment (Figure 5). Then: (a) τ is minimised when the caches are placed at $j/(n+1)$, $j = 1,\ldots n$; and, (b) σ is minimised when the caches are placed at

$$\frac{1+\alpha+\alpha^2+\ldots+\alpha^{j-1}}{1+\alpha+\alpha^2+\ldots+\alpha^n} \qquad \text{for } j = 1, 2, \ldots, n \tag{10}$$

Proof (a) Since the caches are identical, the fact that it is a *p*-median problem (established in subsection 2.1) can be used to obtain the desired result straightforwardly.

(b) Here we again equate flows immediately prior to caches and the server. Let these, from the non-server end of the segment be $F_1, F_2, \ldots, F_{n+1}$, where F_{n+1} is the demand that is actually satisfied by the server. Referring to Figure 5,

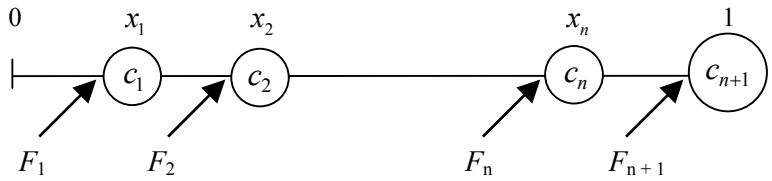

Figure 5 – A uniform linear segment with n caches at distances $x_1, x_2, \ldots, x_n$ from the left end of the segment. The flows immediately prior to the caches and the server are denoted $F_1, F_2, \ldots, F_{n+1}$.

$$F_1 = x_1$$
$$F_2 = (x_2 - x_1) + (1-\alpha)x_1 = x_2 - \alpha x_1$$
$$F_3 = (x_3 - x_2) + (1-\alpha)x_2 = x_3 - \alpha x_2$$
$$\vdots$$
$$F_{n+1} = 1 - x_n + (1-\alpha)x_n = 1 - \alpha x_n$$

It is now a simple matter to verify that (10) satisfies these equations. Notice that (8) and (9) give caches at 1/3 and 2/3 in agreement with theorem 2.

Rodriguez *et al.* (2001) have studied hierarchical networks but from a different point of view. Their underlying model is that of h_1-level *institutional* tree networks being connected at their root nodes to the tips of h_2-level *regional* tree networks which in turn are connected at their root nodes to the tips of an h_3-level *national* tree network. In their study they took $h_1 = h_2 = h_3 = 3$ and all trees to have a branching factor of 4. It was assumed that there were institutional caches, regional caches and a national cache at the roots of each of the three levels of tree and that at each level the caches were of the same size. They investigated theoretically the impact of this arrangement on latency, bandwidth usage, cache load and disk space requirements with regard to requests for objects which had to traverse an 'international path' from the server(s) prior to the root node of the national cache. This was an interesting and informative study with relevance to hierarchical caching but is not directly comparable to the present work since it was not **locational** in nature (because the sites of the caches were fixed although their sizes were not).

2.4 Real networks

The work above concentrates on uniform linear chains as approximated by uniform linear segments. But real networks are somewhat more complicated than this, so how can the above results be used? We now answer this in general terms.

First, there is the question of discreteness. When n is large, making a continuous approximation should give the right approximate locations for caches, and when n is small, complete or extensive enumeration should be viable.

Secondly, uniformity is no great restriction. Where (10) gives the distances from the end of the segment at which caches should be placed, it is in fact dividing the **demand** in these proportions. For τ, the situation is slightly more complex as dividing demand equally, while it should give a reasonably good solution, need not be optimal (see case $n = 1$ in subsection 2.2).

Thirdly, trees are relevant to real situations rather than linear chains, so what difference may be expected? For τ we can get some indication by using a linear segment with the density set appropriately to reflect the exponential increase in numbers of nodes with distance from the server. Let

$$\rho(\xi) = e^{k(1-\xi)},\ \xi \in [0,1].$$

Using (7) leads, after some manipulation, to

$$e^{kx} + kx = 1 + k.$$

A series approximation of x up to the quadratic term in k is

$$x = \frac{1}{2} - \frac{k}{16} + \frac{k^2}{192} + \cdots \qquad (11)$$

To correspond to a **binary tree** with **depth** three, it is appropriate to take $k = 3\ln 2 = 2.08$ (this corresponds to the correct number of nodes at level j in the tree, $j = 0, 1, 2, 3$). Substituting into (11) this gives $x = 0.39$. However, this 'cache' would translate to several real caches on the tree. Thus for a uniform binary tree with depth three, $x = 0.39$ corresponds to being a little less than two links from the server or between $2^1 = 2$ and $2^2 = 4$ caches. This suggests three caches, one situated one link from the server and the other two on the other branch at a distance two links from the server. The reader may verify that this is indeed a good solution. Similarly for a binary tree of depth six we arrive at $x = 0.33$ corresponding to exactly 16 caches at a distance of 4 links from the server. Of course these results are very approximate but they give some feeling for the behaviour.

For σ, the authors are aware of no algorithm but on intuitive grounds it is clear that junctions play a special role. This is because, as long as the cache has sufficient processing capability, σ cannot exceed the maximum flow on any incoming link however many in number, but if there is not a cache at the junction the flow in the outgoing link (towards the server) is at least the sum of the flows on incoming links. Depending on the tree, this could have the effect of either pulling a cache further towards the server. or away from it, but overall we may say that caches should be 'close to' the server for uniform sized links.

It seems that there are at least two possibilities for using tree models as developed: within a very large institutional network; and, near the server. Krishnan *et al.* (2000) investigated the latter with regard to a Bell Labs network containing one server. On a larger scale the deployment of caches becomes less simple. An Internet Service Provider (or ISP) is likely to prefer to satisfy as much demand as possible by caching within its own network for at least two reasons: it reduces the amount of bandwidth that needs to be purchased from upstream (serverwards) ISPs; and, most packet delays and losses occur at ISP exchange points.

2.5 Some practical considerations

The above treatment has discussed only the most basic models and so can only provide an idea of how a system behaves. Roughly speaking it may be said that there is a case for spreading caches around with regard to latency reduction, but that reducing congestion tends to require caches to be sited more towards server(s), though this is ameliorated somewhat since there may well be higher capacity links in the region of the server. The case of finding a best compromise solution is clearly intermediate. To take matters much further it is necessary to look at practical considerations which have so far been neglected. Some of these will now be mentioned though not all of these may be expected to have any substantial effect on what is an appropriate model.

Clearly, transparent caching possesses some advantages but there is a potential snag. While the routing pattern does form a tree at a given instant, the particular routing tree can change as network traffic conditions vary. In confronting this potential difficulty Krishnan *et al.* (2000) studied the routing pattern from Bell Labs and concluded that the routing patterns were quite stable and compared with the time scale of Web requests it was reasonable to assume a constant tree and that even over longer periods a good placing of caches remained

good. Despite this, though, the question of routing stability has to be borne in mind in practical situations.

It is common to speak of hit rates but it should not be expected that a cache with a hit rate α will reduce latency by a corresponding amount! A reason for this is that, upon a request, a straightforward implementation will involve setting up a *connection* from client to cache and, if the cache does not hold the requested object, the setting up of another connection between cache and server. The time taken up by this on a cache miss can be significant and on average can degrade latency savings substantially. Rabinovich & Spatscheck (2002) suggest that *connection caching* can be as important. This makes use of the fact that the HTTP(1.1) protocol permits *persistent* TCP (Transmission Control Protocol) connections (Petersen & Davie, 2000). That is, after a request has been satisfied a connection is not torn down but kept and reused for a later request. Incidently, object hit rates have been considered here but these take no account of object size. An alternative measure that does this is the *byte hit rate* which gives the proportion of bytes of requested data that a cache can return directly to the client.

One question that arises relates to the fact that objects may be updated from time to time even though they are considered to be *static*. Consequently, a cache may hold an out of date copy of a requested object. This is the problem of *cache consistency* and one scheme for tackling it is as follows. When a server delivers an object it attaches a *time-to-live* beyond which a cache must assume the item is *stale* (i.e. not *fresh*). When a cache receives a request for a stale object it can pass the request on to the appropriate server as usual. The server will then transmit the object or, if no changes have been made to the version held by the cache, the server can merely send a short message to this effect thus saving bandwidth. It may be noted that this does not **guarantee** fresh data since it is possible that an object has in fact been modified during the time-to-live period.

Even when an object is subject to change, some account of this can be taken by *prefetching* or using *delta encoding*. With prefetching, the cache uses some heuristic to anticipate objects which will be requested next, or at least soon. These can then be retrieved and stored ready in case a client should actually make a request for them. Clearly this will not always work. On the other hand, delta encoding attempts to reduce the amount of data sent from servers by transmitting just the **changes**, or *delta*, that have occurred to an object held by a cache since the cache received its copy. The cache incorporates the changes and thus can supply an up to date version of the object to a client. Prefetching has the potential to increase hit rates and delta encoding can increase byte hit rates.

Upon a cache miss some systems do not immediately forward a request on to the server but rather 'investigate' whether the required object is held by another nearby cache. This is *cooperative caching* and the reader is referred to Rabinovich & Spatscheck (2002) for more details. However, we will briefly mention one idea akin to cooperative caching, namely *cache routing*. With this, upon a cache miss, the cache sends a request on to another cache which is nearer to, but not necessarily on a shortest path to, the server. This results in a *routing tree* **among caches**. The 'nearer to server' condition ensures that requests will ultimately reach the server if they have not already been satisfied.

Caching works as well as it does largely because the most popular objects are **much** more popular than other objects. Indeed, it has been noted earlier that frequency of requests for objects is well modelled by the Zipf distribution. However, after a request has been through a proxy cache on a cache miss the distribution 'has been changed'. Specifically, if a cache has

the N most popular objects, the request stream for those requests that are passed on to another cache (see subsection 2.3) or to the server will have a distribution

$$\text{P(request for } i\text{-th most popular document)} = C'/(N+i)^{\alpha}$$

(C' here being a normalisation constant) which is 'much flatter' than the original Zipf distribution. Doyle *et al.* (2001) call this the *trickle down effect*. Not only would it be interesting to investigate what effect this might have, but it would also be of interest to investigate further the validity of the Full Dependence Assumption and the consequences of it only being approximately true.

Other factors of interest are replacement policies (Jin & Bestavros, 2001) and choosing cache size (Kelly & Reeves, 2001).

3. Discussion and Conclusions

The models MLP, MCP and BCSP have been introduced. The first two of these have been investigated for very simple model networks. This is not unreasonable in the circumstances since we are concerned with looking at general features and trends. It is not really appropriate to develop sophisticated exact algorithms without further modelling effort and in any case the area is developing rapidly so the internet may look quite different in a few years time. Also, there are so many different network structures and different factors involved that it would indeed be difficult to proceed much further while still maintaining generality.

The most severe limitation of our work is probably the restriction to one server and extensions along these lines would be welcome. Other main lines for further development are improved modelling, specialised models for particular cases and further investigation into minimising congestion, in particular studying the effects of varying link capacities and developing an algorithm to solve MCP over a tree. It would also be interesting, from an algorithmic point of view, to study BCSP. One line of development that is potentially profitable is to look at reverse proxies (also called *surrogates*) and at *Content Delivery Networks* (or CDNs) which occupy a position between the servers and clients or client-side caches. In both these cases data is 'pushed from' the server and so is more under system control, thus enabling better consistency control. This results in a *data assignment problem* (see Kangasharju *et al.*, 2002). Also there may be an impact on the Full Dependancy Assumption. Baentsch *et al.* (1997a) argue for a hybrid proxy caching / replication scheme. Further consideration of such ideas is beyond the scope of the present work.

Finally, as observed earlier, the question of cache location is related to referral systems more generally. This might provide the stimulus to develop some of the above ideas further. The authors hope that other researchers will follow this up.

Acknowledgements

An early version of this paper was presented in November 2002 at SBPO XXXIV in Rio de Janeiro.

In contributing to this Special Issue, the first named author has pleasure in expressing his gratitude to Professor Roberto Galvão for a fruitful working relationship and for his many kindnesses during a long friendship.

References

(1) Baentsch, M.; Baum, L.; Molter, G.; Rothkugel, S. & Sturm, P. (1997a). Enhancing the Web infrastructure – from caching to replication. *IEEE Internet Computing*, **1**, 18-27.

(2) Baentsch, M.; Baum, L.; Molter, G.; Rothkugel, S. & Sturm, P. (1997b). World Wide Web caching: The application-level view of the Internet. *IEEE Communications Magazine*, **35**(6), 170-178.

(3) Doyle, R.P.; Chase, J.S.; Gadde, S. & Vahdat, A.M. (2001). The trickle down effect: Web caching and server request distribution. **In:** *Proceedings of the Sixth Web Caching and Content Delivery Conference* [edited by M. Rabinovich and A. Bestavros], Elsevier, 1-18.

(4) Galvão, R.D., Espejo, L.G.A. & Boffey, T.B. (2002). A hierarchical model for the location of perinatal facilities in the municipality of Rio de Janeiro. *European Journal of Operational Research*, **138**, 495-517.

(5) Garey, M.J. & Johnson, D.S. (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

(6) Gerrard, R.A. & Church, R.L. (1994). A generalized approach to modeling the hierarchical maximal covering location problem with referral. *Papers in Regional Science*, **73**, 425-453.

(7) Jin, S. & Bestavros, A. (2001). GreedyDual*Web caching algorithm: exploiting the two sources of temporal locality in web requests streams. *Computer Communications*, **24**, 174-183.

(8) Kangasharju, J.; Roberts, J. & Ross, K.W. (2001). Object replication strategies in content distribution networks. **In:** *Proceedings of the Sixth Web Caching and Content Delivery Conference* [edited by M. Rabinovich and A. Bestavros], Elsevier, 39-53.

(9) Kelly, T. & Reeves, D. (2001). Optimal Web cache sizing – scalable methods for exact solutions. *Computer Communications*, **24**, 163-173.

(10) Krishnan, P.; Raz, D. & Shavitt, Y. (2000). The cache location problem. *IEEE/ACM Transactions on Networking*, **8**, 568-582.

(11) Luotonen, A. (1998). *Web Proxy Servers*. Prentice-Hall, Upper Saddle River, NJ.

(12) Mirchandani, P.B. & Francis, R.L. (1990). *Discrete Location Theory*. Wiley-Interscience, New York.

(13) Narula, S.C. & Ogbu, U.I. (1985). Lagrangean relaxation and decomposition in an uncapacitated 2-hierarchical location-allocation problem. *Computers & Operations Research*, 169-180.

(14) Petersen, L.L. & Davie, B.S. (2000). *Computer Networks: a systems approach*. Morgan Kaufmann Publishers, San Francisco.

(15) Rabinovich, M. & Spatscheck, O. (2002). *Web Caching and Replication*. Addison-Wesley, New York.

(16) Rodriguez, P.; Spanner, C. & Biersack, E.W. (2001). Analysis of Web caching architectures: hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, **9**, 404-418.

(17) Tamir, A. (1996). An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, **19**, 59-64.

(18) Wessels, D. (2001). *Web Caching*. O'Reilly, Sebastopol, CA.