Optimal Placement of Web Proxies in the Internet with link Capacity Constraints

Fathi Tenzakhti
Faculty of Applied Sciences, Sohar University
Sohar, Oman
tenzakhti@yahoo.com

ABSTRACT: This paper considers the problem of placing a fixed number of proxies in the Web modelled as a tree network. The study assumes that each link in the network has a fixed capacity, and that the total load passing through a link cannot exceed this capacity. We are interested in finding a minimum cost replication set that optimises the cost of servicing access requests in a read-only environment taking into account the capacity constraints of the links.

Categories and Subject Descriptors
H.3 [Information Storage and Retrieval] Web based services; H.3.5 [Online Information Services]

General Terms
Web services, Web proxies, Internet programming

Keywords: Web Server, Replication, Dynamic Programming, Tree Network, Link Capacity.

Received 8 April 2006; Reviewed and accepted 24 April 2006

Introduction
As the explosive growth in the use of the Web in the past few years, the Web related accesses have constituted the major traffic in the Internet. More and more people use the Web for searching information, downloading software, shopping, or entertainment. Due to the large traffic generated by the Web users, most of us have experienced the frustration of very slow response when accessing some Web sites. To alleviate traffic congestion, and to increase fault tolerance, many popular Web servers are replicated at several locations, such as the Web sites of Netscape, Yahoo, American Online and so on.

A proxy server is associated with a Web server. It stores objects of a Web server and acts as the “front end” of this Web server to clients requesting objects, e.g. Web pages. When a client issues a request to access an object on the Web server, the request is intercepted by the proxy on its way to the server. The request is directly serviced by the proxy if the requested object is available at this proxy; otherwise it is forwarded to the server. One of the main objectives of using proxies is to help distributing traffic more evenly inside the network and reducing load on the Web servers.

2. Contributions and organization of the paper
In this paper, we investigate the optimal placement policy for Web proxies in the Internet under the transparent data access model [17, 24]. The objective is to minimize the overall latency of accessing the target Web server. The main contribution is to formulate the optimal proxy placement problem as a Dynamic Programming problem [8].

The rest of the paper is organised as follows. Section 3 reviews and discusses some existing research work related to object replication in the Web. Section 4 describes the system model and formulates the placement problem. Section 5 discusses the optimal placement of proxies in a tree network, presents a recursive formulation, and suggests a new dynamic programming solution for this placement problem. Finally, Section 6 concludes the paper.

3. Related Work
The optimal replica placement in the Web has been widely studied in the literature [2, 13, 15, 18, 23]. In [15], Li et al. have argued that the placement of Web proxies is critical to the performance of the Web and investigated the optimal placement policy of proxies for a target Web server in the Internet for a read-only environment. They have shown that the problem could be modelled and solved using dynamic programming to Optimise the mean response time for accessing Web servers. They have presented an algorithm with a time complexity of $O(m^n)$ where $m$ is the size of the tree and $n$ is the number of proxies. However, the authors did not consider the loads of the clients and the capacity of the proxy servers in their problem formulation and analysis.

In [26], the authors have investigated the optimal placement of proxies of a Web server on the Internet, with the consideration of both read and update operations to the data on the Web server. First, the problem of optimal placement of $k$ proxies in a system to minimize the total access cost to the Web server has been studied. Then, for an unknown number of proxies, the optimal number of proxies required in the system has been found. The problems have been formulated using the dynamic programming method and the optimal solutions have been obtained. Simulations have been conducted to evaluate the performance of the proposed algorithms and to demonstrate how the effectiveness of proxy placement is affected by various factors, such as network traffic load, number of proxies, read–write ratio and proxy hit ratio.

In [19], the authors have considered the communication complexity of maintaining the replicas of a logical data-item, in a database distributed over a computer network. They have proposed a new method, called the minimum spanning tree write, by which a given processor in the network should multicast a write of a logical data-item, to all the processors that store replicas of the items. Then they have shown that the minimum spanning tree write is optimal from the requirements of the Web requiring extensive high computational and storage capacity [6, 16, 19, 20]. This has motivated service providers to optimise delivery distances as well as contents allocation while considering the capacity of the links. Unfortunately, there has been so far a little work to investigate the impact of link capacity on the existing solutions suggested for replica placement.
the communication cost point of view. They have also demonstrated that the method by which a write is multicast to all the replicas of a data-item affects the optimal residence set of the item, i.e., at which processors in the network the replicas should be located. They have then considered the problem of determining an optimal residence set for a data item, assuming that each processor employs the minimum spanning tree write at run-time.

Cidon et al [6] have presented an O(hn) algorithm for electronic content allocation on hierarchical servers where the storage cost has been considered. The server placement problem in a tree network has also been briefly discussed. However, there has not been any constraint imposed on the number of replication servers.

4. The System Model

For the purpose of this paper, the replicated Web consists of a number of sites interconnected by a communication network. Objects are replicated at a number of sites and managed by a group of processes called replicas, executing at the replica sites. The network topology is represented by a graph \( G = (V, E) \) where \( m = |V| \) is the number of nodes and \( E \) is the set of edges and represent physical links connecting these nodes. Nodes are routers, Web servers or a combination of both (servers provide the information a client is looking for). Routers are connected via wide-area links to form the communication network. Some routers, called gateways, provide connections to the outside Internet. These are the gateways through which all requests enter the system.

The effectiveness of any placement algorithm depends on the stability of the Internet routing methods. If the routes on the Internet are stable, the routes used by clients to access the Web server will form a shortest path tree (or routing tree) rooted at the server. Existing studies in [21] have pointed out that in practice most routes in the Internet are stable. It has been found that 80% of routes change at a frequency lower than once a day. Krishnan et al [16] have traced the routes from Bell Lab’s Web server to 13,533 destinations. They have found that almost 93% of the routes are stable during their experiments. Therefore the stability of Internet routing is a realistic assumption that can reduce the placement problem of placing proxies on an arbitrary topology to that on a tree network.

Given the stability of Internet routing [21], an object requested by a client \( c \) and located at server \( s \) travels through a path \( s \rightarrow r_1 \rightarrow r_2 \rightarrow \ldots \rightarrow r_i \rightarrow c \) called a preference path and is denoted by \( \pi(s, c) \). The preference path consists of a sequence of nodes with the corresponding routers. Routes from \( s \) to the various clients form a routing tree along which requests are propagated. Consequently, for each server, a tree rooted at \( s \) could be constructed to depict the routing tree (Fig. 1), and the entire Web could be represented as a collection of such routing trees, each rooted at a given Web server. Formally, a routing tree is the union of the preference paths. Since an object from server \( s \) to client \( c \) passes through the nodes on the preference path, it is advantageous if the request is serviced by one of the internal nodes along the path. In fact, the closer the data is to \( \pi(s, c) \) to the server, the greater is the benefit.

Each server \( s \) knows the preference path from itself to any client. This information can be extracted and periodically refreshed from the routing database kept by the routers [14]. The routing information allows the comparison of network distances (e.g., number of hops) among servers within a given platform. A user issues one request at a time for a Web page, which is fetched to the user as a single unit. In the tree, when a client sends a request to access a server \( s \), the request is always sent to the root along the preference path. If the request meets a proxy on its way and the requested object is available, it is served by the proxy. Otherwise it has to travel all the way to the root where it is serviced by \( s \). Note that if there is a proxy closer to the client but not enroute between \( c \) and \( s \), it is ignored. Table 1 summarizes the notation used in the paper.

\[ C(T_s, n, k) \]

Table 1: Notations

<table>
<thead>
<tr>
<th>Table</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C(T_s, n, k) )</td>
<td>cost of placing ( n ) proxies in tree ( T_s ), considering the capacity ( k ) of the nodes.</td>
</tr>
<tr>
<td>( d(i, j) )</td>
<td>distance between site ( i ) and site ( j ) in number of hops.</td>
</tr>
<tr>
<td>( E )</td>
<td>set of edges.</td>
</tr>
<tr>
<td>( f_v )</td>
<td>access frequency at node ( v ).</td>
</tr>
<tr>
<td>( G )</td>
<td>the graph representing the Internet.</td>
</tr>
<tr>
<td>( i )</td>
<td>( v^i ) site.</td>
</tr>
<tr>
<td>( m )</td>
<td>number of sites in the network.</td>
</tr>
<tr>
<td>( n )</td>
<td>number of proxies.</td>
</tr>
</tbody>
</table>

\( e(v, z) \) optimal proxy for node \( v \) in the tree rooted at the target server \( s \).

\( P_u \) set of nodes whose proxy is \( u \).

\( R_s \) residence set.

\( R_k \) residence set of the \( k^\text{th} \) object (set of sites holding a copy of object \( k \)).

\( S \) target server to be replicated.

\( T_i \) subtree of tree \( T_s \) rooted at \( i \).

\( T_s \) the tree rooted at the target server.

\( P_k \) replication tree for the target server \( S \).

\( v \) set of vertices of the graph \( G \).

\( k \) capacity vector representing the capacity of the nodes.

\( \lambda_v \) capacity of a node \( v \).

\( \rho_i \) parent link of node \( i \).

\( \pi(s, c) \) sequence of nodes to get a request from the client to the server.

![Fig 1. Two routing trees \( T_i \) and \( T_s \) for server\( i \) and \( s \) (dark) each with 2 proxies](image-url)
4.1 Assumptions
We make the following assumptions, which have been widely used in similar existing studies [6, 10, 13, 25]:

- The Web is structured as a set of routing trees.
- Each node v in tree except the root has a parent link \( \lambda \), that links it to its parent node.
- The proxies contain an integral copy of the target server.
- Every request is serviced at the first proxy it meets while travelling in the direction of the target server.
- The requests are read-only.
- Each node \( v \) in the tree has a load \( \lambda \) (number of requests/sec).
- The parent link of any node \( (p, v) \) has a capacity \( (\text{number of requests/sec}) \) and the total loads of the clients passing through the parent link of the node should not exceed.
- Objects accessed by clients are always available at proxies.
- The placement of proxies is static. Once a proxy is placed somewhere in the network, it would stay there for a long period of time.
- Each client request is a router-aware request.
- Each proxy is en-route proxy collocated with a router.

It is expected that servers and proxies would not change locations in the Web for a relatively long period of time. Studies in [17] have revealed that although the client population changes significantly from time to time, the route from the Web servers to the clients remain stable in the branches that carry most of the traffic. This indicates that the optimal locations for the proxies do not change frequently over time. Consequently, it is realistic to use the tree topology to model the structure of the Web and assume that the traffic inside the network is stable.

4.2 Routing Mechanism
To understand the routing mechanism, we first have to distinguish between two types of requests: router-aware and normal requests [27]. Router-aware requests are those that require interception by the routers whereas normal requests are those that do not require any interception by the routers.

In general, there are three approaches to transparent object access for replication services on proxies: en-route replication [10, 17], hierarchical replication [6] and hybrid replication [27]. In the en-route model, each client request is represented by a router-aware request which is intercepted by a router along the regular routing path to the server. If a copy of the requested object is found on the local proxy, the router services the request; otherwise the request is forwarded to the next router. In the hierarchical model, the proxies are organised in a hierarchical manner such that the target server resides at the top of the hierarchy and every proxy knows the location of its parent proxy. When a request is issued by a given client, it is routed to a predetermined proxy. A proxy either services the request if a replica of the requested object is stored locally; otherwise forwards it to the parent.

In the hybrid model, the proxies are organised in a hierarchical manner but the client request are router-aware. The request does not need to know the address (e.g. IP address) of the proxy. It just follows the path to the target server and when it gets to a router, the request is interpreted by the router until a replication proxy is reached. If the proxy has the object, it is serviced there and the value read is sent back to the requesting client (Fig.2); otherwise the proxy directly forwards this request to its parent proxy. As in [27] we use the hybrid model for the purpose of the present study. Each client request is router-aware and the client does not need to know the address of the proxy server. Since this study assumes whole site replication, the requested object is always found in the first proxy reached by the request. So, an en-route proxy intercepts any request that passes through it satisfies it. That is, if a client’s request meets an en-route proxy it will be served directly by the proxy. Otherwise, the request has to go all the way to the server and is served by the server. It is important to note that a client may not be served by a nearby proxy if the proxy is not on the route between the client and the server, even though the client is physically close to the proxy.

5. Optimal Placement of Proxies in the Tree Network
The proxies discussed in this paper are transparent enroute proxies [24, 27]. They are only placed along the routes from clients to the server and are transparent to the clients. A typical configuration is that proxies are co-located with routers and are maintained by network providers [24]. This configuration has significant operational benefits because proxies can be introduced easily into the existing network infrastructure [24]. It is worth noting that almost all existing caching products include such a transparent operation mode [24].

An efficient placement of proxies would result in most client requests to be served at proxies without having them travel further to the server. Since the client access patterns as well as the tree structures are different, the placement of proxies has significant impact in the overall system performance. To formally define the problem of placing a set of proxies in a tree network taking the capacity of the links as a constraint, we introduce the following definition and notation.

**Definition 1:** a residence set of a graph is a set of vertices at which copies of the object are placed. The minimum residence set is a residence set that provides the minimum cost (e.g. minimum average response time) among all possible residence sets in the graph. An \( n \)-minimum residence set is a minimum residence set containing \( n \) vertices.

Let \( d(u, v) \) be the distance between any two nodes \( u \) and \( v \) in the tree. \( d(u, v) \) is equal to the length of the path, \( \pi(u, v) \), between \( u \) and \( v \). In other words,

\[
d(u, v) = \sum_{(w, v) \in \pi(u, v)} d(w, v)
\]

Let \( p(v, s) \) be the first proxy encountered by a request while travelling from \( v \) to \( s \) in the tree \( T \). We call \( \pi(v, s) \) the optimal proxy. This could be \( v \) itself if \( v \) is a proxy or \( s \) if no proxy is encountered along the way to the root server. Let \( f_i \) be the access frequency from client \( v \) to server \( s \) during a period of time \( \phi \), the period between two successive invocations of the proxy placement algorithm, and \( k_i \) the load that node \( v \) imposes on the parent link of the proxy \( p(v, s) \). If \( R \) is the residence set (the set of proxies for the tree \( T \), associated with \( p(v, s) \) function), then the total distance to access the proxies is \( d(R) = \sum_{v \in R} d(v, p(v, s)) \) and the total cost of accessing the object is given by

\[
C(T, R, k) = \sum_{v \in R} f_i d(v, p(v, s))
\]

Any node \( v \) whose optimal proxy is \( u = p(v, s) \) imposes a load \( \lambda_i \) on \( p(v, s) \) the parent link of \( u \). Letting \( e_i \) be a vector that stores the capacities of all the links in the tree and \( k_i \) be the capacity of the parent link of node \( v = p(v, s) \). With the capacity constraints, the set of nodes whose optimal proxy is \( u \) should not impose a load that is greater than the capacity \( K_v \) of \( p(v, s) \), the parent link of \( u \). Consequently, if \( p(v, s) \) is a node in the tree, then the inequality \( \sum_{v \in R} k_i e_i \leq k_v \) must hold. Now, for a fixed number of proxies, say \( K \geq 1 \), let us find the \( n \)-minimum residence set \( R \) that minimises the total access cost \( C(T, R, k) \) over the tree, taking into consideration the capacity constraints \( k \) of the links. The problem thus reduces to finding the minimum access cost \( C(T, R, k) \) given by

\[
C(T, R, k) = \min_{R \subseteq V} \left\{ C(T, R, k) \right\}
\]
In general, the problem of placing replicas in the tree while imposing a constraint on the capacity of the links is an NP complete problem [8]. However, when we consider proxies where the direction of the read requests is always directed towards the target server, the problem is not NP complete any more.

The optimization problem is to minimize the average response time with the following constraints:
- The number of proxies is $n$.
- The load of each link does not exceed its capacity.

This problem exhibits the optimal substructure condition and lends itself to be solved by the dynamic programming approach which is known to give an optimal solution to the problems that exhibit the optimal substructure. This requires the optimal solution to the problem to embed optimal solution to the subproblems. Given that this can be justified by the fact that given an optimal placement of replicas in a routing tree, it must include optimal placements in the left subtree, the subtree rooted at $u$ and the write subtree. Otherwise, if a placement in one of these subtrees is not optimal, a better overall placement for the full tree can be obtained by a cut-and paste technique which is a contradiction [7].

5.1 Formulation of the Proposed Placement Solution
Consider the tree $T_v$ rooted at $s$ with a set $V$ of vertices. Assume that the children of each non-leaf vertex are ordered from left-to-right so that given any two siblings $u$ and $v$, we are able to determine that is to the left of or vice versa. Generally, given $x$ and $y$ in $T_v$, $x$ is said to be to the left of $y$ if there exist $u$ and $v$ such that $x \in T_u, y \in T_v$ and $u$ and $v$ are siblings with $u$ being to the left of $v$. For $v \in T_v$, let $T_u$ be the subtree of $T_v$ rooted at $v$. For any $u \in T_v$, we can partition $T_u$ into 3 sub-trees (see Fig 3):

- Sub-tree $L_{uv}$ containing all nodes to the left of $u$.
- Subtree containing all nodes in $T_u$.
- Sub-tree $R_{uv}$ containing the rest of the nodes.

Formally, we have

- $L_{uv} = \{x : x \in T_v : x \text{ is to the left of } u\}$
- $R_{uv} = \{x : x \in T_v : x \text{ is to the right of } u\}$

For all the $n$ proxies, we need to check all possible partitioning points $u \in T_v$ and all possible values $n_i$. Recursively, we put the proxies in $T_u$ and $R_{uv}$ the same way as in $T_v$. The dynamic programming approach can be formulated by the following equation:

$$C(T_v, n, \bar{R}) = \begin{cases} 
\sum_{\forall x \in L_{uv}} \mu(\bar{x}, v) & \text{if } n = 1 \text{ and } \sum_{\forall x \in L_{uv}} \lambda_x \leq \kappa_v \\
\min_{\forall x \in L_{uv}} \left( C(L_{uv}, n', \bar{R}) + C(R_{uv}, n-n', \bar{R}) \right) & \text{if } n > 1 \text{ and } \sum_{\forall x \in L_{uv}} \lambda_x \leq \kappa_v \\
+ \infty & \text{Otherwise}
\end{cases}$$
In equation (9), \((L_{uv}, \kappa_u)\) is equal to the total cost of accessing the node \(v\) from all nodes in \(L_u\). This cost is undefined if the total load of the nodes in \(L_u\) is greater than the capacity \(\kappa_u\) of the parent link \(P_v\). 

\[ C(T_{uv}, t, \kappa) \]

is recursively defined in \(T_u\) with the capacity constraint \(\kappa\) relative to the parent links of nodes \(T_v\). The capacity constraint of \(R_{uv}\) with respect to parent link of the proxy \(v\) is the capacity \(\kappa\) obtained by subtracting from \(\kappa_u\) the total load imposed on the parent link \(P_v\) of node \(v\) by the nodes in \(L_{-uv}\).

**Theorem 2**: The cost of the proxy placement computed by equation (9) is the minimum cost.

**Proof**: We start by numbering the tree nodes according to a post order traversal of the tree (see Fig. 4). When there is only one proxy placed at the root \(u\) and the cost computed by equation (9) is the minimum cost if the capacity of the links leading to the root node is not exceeded by the load of the nodes in \(L_u\). If \(u > 1\) we assume that we know \(u\), the highest numbered proxy in \(T\), different from \(v\). The minimum cost of placing \(n\) proxies in \(T\) is found by assuming that \(T_v\) contains \(n\) proxies, \(R_{uv} = n-1\) proxies, and \(L_{-uv}\) no proxies. Calculating the best possible cost under this assumption and then minimizing over \(n\). Since we have no a priori information about \(u\), we must minimize over all possible values of \(u\). In doing so, we will have generated the proxy placement with minimum cost.

6. Conclusions

This paper has considered the problem of proxy server placement with capacity constraints in a read-only Web environment. The Web has been modelled as a set of trees rooted at the target servers to be replicated and treated the optimal placement of \(n\) proxies in a tree network consisting of \(m\) nodes. The dynamic programming approach has been used to formulate the problem of placing \(n\) proxies in a tree of \(m\) nodes taking into consideration the capacities of the links.

**References**
