Optimal Methods for Coordinated En-Route Web Caching

by

Keqiu Li

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Professor Hong Shen

School of Information Science
Japan Advanced Institute of Science and Technology

September 27, 2005

Abstract

Web caching is an important technology for improving the scalability of web services. This dissertation investigates some key problems for coordinated en-route web caching, such as (multimedia) object caching, (transcoding) proxy placement, cache replacement. The main contributions of this dissertation are outlined as follows:

- We address the problem of coordinated en-route web object caching for tree networks (i.e., determining the locations where a copy of the same object should be cached in a network such that the specified objectives are achieved). A dynamic programming-based optimal solution and its analysis are presented. We also extend this solution to solve the problem of coordinated en-route object caching for autonomous systems and the problem of proxy placement for coordinated en-route web caching in tree networks and autonomous systems (i.e., determining the locations where a cache/proxy should be placed in a network such that the specified objectives are achieved). Extensive simulation experiments are conducted to evaluate our proposed solutions over a wide range of performance metrics in comparison with existing solutions proposed in the literature.

- We address the problem of coordinated en-route multimedia object caching for transcoding proxies for linear and tree networks (i.e., deciding the locations where an exact version of the same multimedia object should be stored in a network so that the specified objective is arrived). Dynamic programming-based optimal solutions and their analysis are also presented. We further extend these solutions to solve the problem of proxy placement for coordinated en-route transcoding proxy caching (i.e., deciding the locations where a transcoding proxy should be placed in a network so that the specified objective is arrived). We compare the performance of our solutions with other solutions over various performance metrics through extensive simulation experiments. The simulation results show that our solutions outperform existing solutions proposed in the literature.

- We address the problem of multimedia object placement for transparent data replication, i.e., choosing a specific version of the same multimedia object to be cached at each node in a network such that the specified objectives are reached. The performance objective is to minimize the total access cost by considering both transmission cost and transcoding cost. We present optimal solutions for different cases for this problem. The performance of the proposed solutions is evaluated with a set of carefully designed simulation experiments for various performance metrics over a wide range of system parameters. The simulation results show that our solution consistently outperforms existing solutions in terms of all the performance metrics considered.

- We address the problem of cache replacement for transcoding proxy caching (i.e., selecting the objects that should be removed from the cache to accommodate a new

object to be cached when the cache space is not enough). We first present a cache replacement algorithm for transcoding proxy caching that computes the aggregate profit of caching multiple versions of the same multimedia object with considering cache consistency, which has not been widely considered in the literature. We also address coordinated cache replacement in transcoding proxies by formulating this problem as an optimization problem which determines cache replacement candidates on all candidate nodes in a coordinated fashion for the objective of minimizing the total cost loss. Moreover, we conduct extensive simulation experiments to compare the performance of our algorithms with some existing algorithms. The results show that our algorithms outperform others in terms of various performance metrics.

Key words: Web caching, Internet, multimedia object, transcoding, optimization, proxy placement, mobile network, simulation, performance evaluation.

Acknowledgments

First of all, I would like to express my deepest gratitude to my supervisor, Professor Hong Shen, for his inspiring and encouraging way to guide me to a deeper understanding of research work, and his invaluable comments during the whole course with this dissertation. His dedication, prudence, and work of ethics have been a never-ending source of inspiration. Without his patience and understanding in difficult times, it would probably not be possible to complete this dissertation.

Many thanks to Professor Francis Y. L. Chin of the University of Hong Kong for his constructive comments and helpful discussion on some contents of this dissertation.

I am very grateful to Associate Professor Keishi Tajima, my sub-theme supervisor, for his valuable instruction on my sub-theme research.

I would like also to give thanks to all other members in SHEN laboratory, especially to Dr. Xiaohong Jiang and Dr. Haibin Kan for their many useful suggestions on my research.

I am deeply indebted to the Japanese Ministry of Education, Culture, Sports, Science and Technology for granting me a scholarship, which makes my study in Japan feasible. Thanks also go to Foundation for C&C Promotion, International Information Science Foundation, Japan Association for Mathematical Sciences, Research Foundation for the Electrotechnology of Chubu, JAIST Foundation Research Grant for Students, Inoue Foundation for Science, and The Telecommunications Advancement Foundation for supporting me to attend and present our work at some international conferences.

I am grateful to my wife and my daughter who have sacrificed many things for me and given me their fullest support.

I would like to give special thanks to my parents and parents-in-law, who give me great encouragement on my study. Particularly, for my parents-in-law's taking care of my daughter, I could concentrate on my research and finish my Ph.D study.

Finally, I would like to thank all the people who either directly or indirectly provide me knowledge, experience, and support.

Contents

Chapter 1

Introduction

1.1 Background

The World Wide Web has become the most successful application on the Internet since it provides a simple way to access a wide range of information and services. However, due to the dramatic growth in demand, considerable access latency is often experienced in retrieving web objects from the Internet, and popular web sites are suffering from overload. An efficient way to overcome such deficiencies is web caching, by which multiple copies of the same object are stored in geographically dispersed caches. An overview of web caching can be found in [31, 79, 98]. Although web caching is similar to memory caching, since they both store objects at different locations for future requests, significant differences between them result from the non-uniformity of web object sizes, access frequencies, retrieval costs, and cacheability. In addition, the performance of web caching depends greatly on the network distance from the user to the server, since users are geographically distributed over the entire internet.

To obtain the full benefits of web caching, different architectures have been employed, such as hierarchical caching [79, 85] and distributed caching [85, 93]. En-route caching is a new caching architecture developed recently [49, 83, 90] in which caches are placed on the access path from the user to the server. Each en-route cache intercepts any request that passes through its associated node, and either satisfies the request by sending the requested object to the client or forwards the request upstream along the path to the server until it can be satisfied. En-route web caching can be implemented in several ways, such as by using light-weight techniques [80, 84], active network [10, 92], etc. En-route web caching has a couple of advantages. First, cache are only located along the routes from clients to servers, and are placed transparently to the clients and servers. Second, since the requests are routed along the regular path, a lot of extra bandwidth and network delay for cache miss are saved. Finally, the additional overhead caused by locating the objects such as sending broadcast queries [99] and maintaining directories [35] are not necessary. Therefore, en-route web caching is a system with good scalability and flexible management. Cooperative caching, in which caches cooperate in serving each other's requests and making storage decisions, is a powerful paradigm to improve cache effectiveness [30, 47, 48].

In [90], the authors studied the problem of coordinated en-route web caching for linear networks and proposed an optimal solution for this problem. In [45], the authors presented an example showing that this solution cannot be directly applied to solve the

same problem for tree networks, and proposed an en-route web caching algorithm applying dynamic programming for placing web files in tree networks. However, the model they established has not considered the impact of caching a copy of a web file at a node on the requests for this file from all downstream nodes of this node. This model can only result in an optimal solution for placing web file at only one node on the path from client to server. In this dissertation, we consider the general case in which several copies of an object can be cached on the path from the client to the server and a cached copy at a node shall benefit retrievals to the object from all downstream nodes of the node. We also address the problem of proxy placement for coordinated en-route web caching for tree networks.

As many mobile appliances are divergent in size, weight, I/O capabilities, network connectivity, and computing power, differentiated devices should be employed to satisfy their diverse requirements. In addition, different presentation preferences from users make this problem more serious. Transcoding, used to transform a multimedia object from one form to another, frequently through trading off object fidelity for size, is a technology that can meet these needs [21,22,41,66,87,97]. There are three main kinds of such media transcoding: transcoding between the same formats of the same media type (e.g. from high-resolution JPEG to low-resolution JPEG), transcoding between different formats of the same media type (e.g. from JPEG to GIF), and transcoding between different media types (e.g. from text to image format). Transcoding can be executed by various components in the network such as server, proxy, and client. In the case of the client, it can preserve the original semantics of system architecture and transport protocols. However, this solution is extremely expensive when the clients are mobile users, due to connection bandwidth and power limitations. In the case of the server, it is not necessary to perform transcoding during the time between the issuance of a client's request and the response from the server; thus, no additional transcoding delay will incur. At the same time, it will take too much storage space to keep all the versions of the same media object on the server. Further, this method is not flexible in dealing with changing clients' needs. For these reasons, it will be better to transcode the media objects in intermediate proxies. Much research has been focused on exploring the advantages of this approach [21,25,37,41], in which an intermediate proxy is capable of transcoding the requested media object to a proper version according to the client's specification before sending the media object to the client. As audio and video applications have proliferated on the Internet, caching media objects in transcoding proxies (transcoding proxy caching for short) has become an important research topic.

In [23], the authors proposed an effective cache replacement algorithm for transcoding proxies by exploring the aggregate effect of caching multiple versions of the same multimedia object in transcoding proxies. However, this algorithm considered only static objects. We further investigate this problem when dynamic objects are involved. Moreover, we address coordinated cache replacement in transcoding proxies. To the best of our knowledge, there is little work done on coordinated en-route transcoding proxy caching. In this dissertation, we address two important factors that affect the performance of transcoding proxy caching, i.e., multimedia object caching and transcoding proxy placement.

1.2 Related Work

Web caching has been recognized as one of the effective schemes to alleviate the server load, reduce the network traffic, and minimize the user access latency over the Internet. There have been a lot of research done on several topics related to web caching, including cache architectures [65,75,78], protocols [35,95,100,101], replacement policies [3,102,104], prefetching [27,29,35,50,64], cache coherency [9,16,19,40], user access prediction [26,68, 70], web traffic characteristics [11,33,34], and dynamic data caching [13,17,20,36,52].

Efficient replacement algorithms for a single web cache can great improve the performance of web caching [46,86,102]. However, these algorithms store copies of an object at each node through which the object passes, without checking whether it is beneficial to do so. This may cause ineffective use of the limited cache space, since there are numerous objects to be distributed in the network. Therefore, it is necessary and important to find methods that can optimally determine the locations to place the copies of an object. In [30,51], the placement and replacement algorithms for local-area networks were studied. However, this problem is relatively unexplored in wide-area networks, which are very different from local-area networks in regard to the number of users and objects.

Cache cooperation is an important approach to improve web performance. Recent studies have focused on the benefits of cooperative caching for distributed systems and large-scale systems [4, 24, 51, 81, 93]. In [107], wide-area cache cooperation was studied under a simple model, in which distances among all nodes in the network are assumed to be the same. In [48], the authors examined three practical cooperative placement algorithms for large-scale distributed caches and showed that cooperative object placement could significantly improve web performance compared to local replacement algorithms, particularly when the sizes of individual caches were small compared to those of the objects. In [47], the object placement problem was formulated as an instance of the facility location problem and solved by reducing it to the minimum cost problem. The time complexity of this object placement problem is very high, at least quadratic of the product of the number of nodes and the number of objects in the network. In [47], two approximation algorithms, greedy placement algorithm and amortizing placement algorithm, were proposed. Although the greedy algorithm looks simple and is easy to implement, it has been shown that the performance of its worst case solution can be arbitrarily far from optimal, i.e. the approximation ratio is relevant to the number of nodes concerned. The amortizing placement algorithm is a constant-factor approximation algorithm. The problem studied in [90] considered the coordinated en-route web caching problem for linear topology, deciding the optimal locations for placing copies of an object among the en-route caches. This scheme, which optimizes the placement of objects on the path from the user to the server, has been shown to perform significantly better than other schemes that considered either object placement or replacement in individual caches only. An en-route web caching algorithm applying dynamic programming for placing web files in the tree network was proposed in [45]. They established a model which does not consider the impact of caching a copy of a web file at a node on the requests for this file from all downstream nodes of this node. This model can result in optimal solution for placing web file at only one node on the path from client to server. In [105], the authors the problem of replication proxy placement in the network and data replica placement on the installed proxies given that a maximum number of proxies are allowed with considering both read and write applications. There is little work done on coordinated en-route transcoding

proxy caching. In [23], the authors proposed an effective cache replacement algorithm in transcoding proxies by exploring the aggregate effect of caching multiple versions of the same multimedia object at one one node. This problem becomes more complicated when a network with many nodes are considered. The methods for coordinated en-route web caching cannot be directly applied to solve this problem since different versions of the same multimedia object cannot be simply dealt with different objects due to the relationship among all the versions of the same multimedia object. There is little work done on coordinated en-route transcoding proxy caching. In [71], the authors proposed a proxy server management scheme for continuous media objects based on object partitioning.

Proxy placement is also an important factor that affects the performance of web caching. Traditionally, web proxies are placed in some "obvious" important places, such as the router of a LAN, gateway in the internet and some strategic locations in the network of an institute or organization [67]. In addition, it has been shown that multiple web proxies are necessary to improve the overall web performance. Furthermore, this is also evident that numerous web sites have employed a replication of the sites. For example, users can select among 15 different servers to access Yahoo within the US, 101 different servers to access the Netscape server. Generally, the problem of web proxy placement can be formulated into the p median problem [38] or the k center problem [96]. Both of them are NP hard for general networks. Heuristic algorithms and dynamic programming-based algorithms have been proposed to find the optimal or sub-optimal solutions. In [43], the placement problem for Internet instrumentation was addressed. Both graph theoretic methods and heuristics for instrumenting the Internet to obtain distance maps were investigated. The authors showed that an Internet distance map service based on their placement techniques can provide useful outlines for server selection by users. In [42], the same authors compared a 2-approximation algorithm for the k median problem, with a greedy approach, a random algorithm and a heuristic which favored nodes of higher out-degree for proxy placement. In [53], the problem of proxy placement for tree networks was studied and the authors modelled the problem as an optimization problem of minimizing the overall access latency for accessing the server. The optimal solution was obtained by using a dynamic programming-based algorithm and the time complexity of the algorithm is $O(n^3k^2)$, where n is the number of nodes in the network and k is the number of proxies to be placed. In [44], the authors further considered the problem of placing web proxies for replicated web servers in the internet. They also presented a dynamic programming-based algorithm to solve the problem. The time complexity of the algorithm is $O(n^3k^3)$. In [49], the authors presented optimal algorithms for line and ring networks. They also presented a dynamic programming algorithm for tree networks. The time complexity of this algorithm is $O(nhk)$, where h is the height of the tree. In [77], several placement algorithms were proposed, using workload information, such as client latency and request rates. In [105], the authors presented two schemes for the replication proxy placement problem. The authors proposed a set of heuristic models for the problem of proxy placement in [63], among which the *k-maximum shortest path count* ($KMPC$) heuristic model obtains the best results. There are also some work done on solving the problem of proxy placement for content distribution networks (CDNs) [6, 12, 77]. CDNs have also emerged as a powerful solution to improve the client response time and to reduce the traffic in the Internet. CDNs consist of a number of distributed proxy servers replicating the contents for better performance and availability than centralized servers. The placement of proxy servers is a key factor in determining the effectiveness of a content

distribution network. Transcoding proxy placement is also an important factor on the performance of transcoding proxy caching. Unfortunately, little work has been done on this topic. The key observation here is that the above proxy placement methods cannot be simply used to solve the problem of transcoding proxy placement although only one word is different because transcoding proxies has the functionality of transcoding, which is an important technique for executing transformation among different versions of the same multimedia object.

Cache replacement plays a significant role on the functionality of web caching as well. A number of cache replacement algorithms have been proposed in the literature with the purpose of attempting to minimize various cost metrics, such as hit rate, byte hit rate, average access latency, and total access cost. All these algorithms can be generally classified into such categories as deterministic policies [15, 102, 106], greedydual-based policies [2, 32], hynrid policies [28, 82], randomized policies [76, 89]. An overview of web caching algorithms can be found in [5]. However, all these algorithms consider the case in which web objects are independent. The objects addressed in this paper are multimedia objects that are dependent because of the relationship among all the versions of the same multimedia object through the technology of transcoding. There is little work done on finding effective cache replacement algorithms for transcoding proxy caching. In [87, 91], the authors studied several caching strategies or architectures for transcoding proxies. However, all these strategies or architectures are evolved from the algorithms mentioned above and the authors have not considered the aggregate effect of caching multiple versions of the same multimedia object at the same time. The algorithm proposed by Chang et al. in [23] is on the similar line with our algorithm. However, this algorithm is not effective since it has not considered the aggregate effect of removing several versions of the same multimedia object at the same time.

1.3 Main Contributions

In this dissertation, we address some key problems for coordinated en-route web caching, e.g., (multimedia) object caching, (transcoding) proxy placement, cache replacement. The main contributions are summarized as follows:

- We present a dynamic programming-based optimal solution and its analysis for the problem of coordinated en-route object caching for tree networks. We extended this solution to solve the problem of coordinated en-route object caching for autonomous systems and the problem of proxy placement for coordinated en-route web caching for tree network. Extensive simulation experiments are conducted to evaluate our proposed solutions over a wide range of performance metrics. The implementation results show that our solutions outperform existing solutions proposed in the literature.

- We present dynamic programming-based optimal solutions and their analysis for the problem of coordinated en-route multimedia object caching for transcoding proxies for linear and tree networks. We extended these solutions to solve the problem of proxy placement for coordinated en-route transcoding proxy caching. We compare the performance of our solutions with other solutions over various performance metrics through extensive simulation experiments. The simulation results show that our solutions outperform existing solutions proposed in the literature.

- We present optimal solutions for different cases of problem of multimedia object placement for transparent data replication with performance objective to minimize the total access cost by considering both transmission cost and transcoding cost. The performance of the proposed solutions is evaluated with a set of carefully designed simulation experiments for various performance metrics over a wide range of system parameters. The simulation results show that our solution consistently and significantly outperforms comparison solutions in terms of all the performance metrics considered.

- For the problem of cache replacement for transcoding proxy caching, we present a cache replacement algorithm for transcoding proxy caching that computes the aggregate profit of caching multiple versions of the same multimedia object with considering cache consistency, which is not considered in the existing researches. We also present a solution for coordinated cache replacement in transcoding proxies by formulating this problem as an optimization problem which determines cache replacement candidates on all candidate nodes in a coordinated fashion with the objective of minimizing the total cost loss. Moreover, we conduct extensive simulation experiments to compare the performance of our algorithms with some existing algorithms. The results show that our algorithms outperform others in terms of various performance metrics.

1.4 Dissertation Outline

For easy understanding and reading, we organize each chapter in a self-contained format as follows:

Chapter 2 addresses some key problems of coordinated en-route web caching. First, a dynamic programming-based optimal solution and its analysis are presented for the problem of coordinated en-route web object caching for tree networks. Second, we extend this solution to solve the problem of coordinated en-route web object caching for autonomous systems and the problem of proxy placement for coordinated en-route web caching for tree networks and autonomous systems. Finally, extensive simulation experiments are conducted to evaluate our solutions.

Chapter 3 concentrates on some major problems of coordinated en-route transcoding proxy caching. In this chapter, we first address the problem of coordinated en-route multimedia object caching for transcoding proxies for linear and tree networks. Based on this solution, we present an optimal solution for the problem of transcoding placement. We then consider the problem of multimedia object placement for transparent data replication. Finally, we compare the performance of our solutions with existing solutions proposed in the literature with simulation experiments.

Chapter 4 focuses on the problem of cache replacement. We propose two effective cache replacement algorithms for transcoding proxy caching and present an optimal solution for coordinated cache replacement in transcoding proxies. We then present some simulation experiments to compare the performance of our algorithms with other algorithms in the literature.

Chapter 5 summarizes our work and discusses some future work.

Chapter 2

Coordinated En-Route Web Caching

2.1 Coordinated En-Route Web Object Caching

2.1.1 Problem Formulation

Before formulating the problem for coordinated en-route web object caching ($CERWOC$), we introduce the notations and definitions used in this section. We model the network as a tree $T = (V, E)$ [1], where V is the set of nodes, each of which is associated with an en-route cache [2], and E is the set of network links [3]. Figure 2.1 shows an example of such a tree topology. In this section, we use T_w to denote a tree whose root is w.

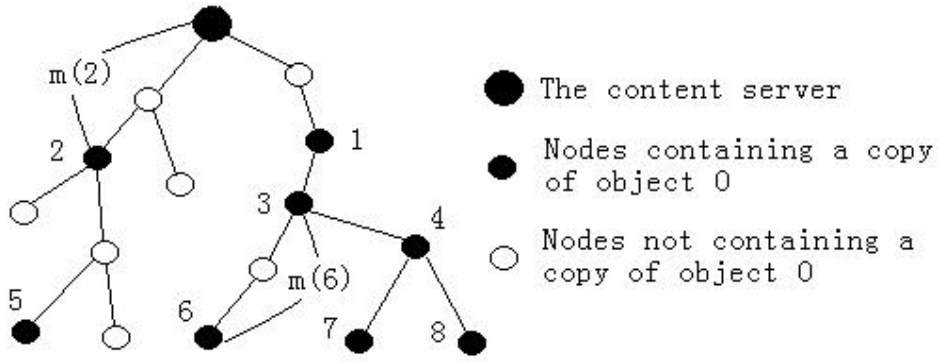

Figure 2.1: Coordinated En-Route Web Object Caching for Tree Networks

Let $P \subseteq V$ be a subset of nodes, at each of which a copy of an object is cached. For every node $v \in V$, $D(v)$ denotes the set of all nodes that are the descendants of node v, and $C(v)$ denotes the set of all nodes that are the children of node v. For any two nodes $u, v \in V$, $E[u \rightarrow v]$ denotes the set of all edges on the path between u and v, and $V[u \rightarrow v]$ denotes the set of all nodes on the path between u and v, including u and v. Let O be the given object for caching. For notational tidiness, we omit argument O in all parameters and functions throughout this section. Let $f(v)$ denote the access frequency of

[1] Without loss of generality, we assume there is only one content server at the root of a tree, at which the objects requested by users are maintained.

[2] Our analysis can be easily extended to the case in which en-routes are associated with a subset of nodes only if we include in the graph the nodes with en-route caches.

[3] In this section, we assume that the network links are bi-directional.

object O which is defined by the number of requests to access object O that pass through node v during a certain period of time; obviously, $f(v) \geq \sum_{w \in C(v)} f(w)$. Estimation of $f(v)$ can be tedious and will be discussed in Section 2.1.4. Let $c(u, v)$ be a nonnegative cost assigned to edge $(u, v) \in E$ for object O, which is defined as network latency, bandwidth consumption, and processing cost at the cache, or some combination of these measures, incurring on (u, v) for accessing O. The cost of a path for object O is the summation of all edge costs on the path.

Due to the limitation of cache sizes, it is necessary and important to find methods to optimally distribute copies of an object among the en-route caches. Accordingly, when a new object is stored in a cache, one or more objects may need to be removed from the cache to make room for it, if necessary. Storing an object at a node enables all the requests for object O previously passing it now to be satisfied at it; hence, its access cost, which is defined in this section as cost saving, is decreased. Similarly, removing the copy of an object from a node increases its access cost, which is defined as cost loss. In this section, we consider cost saving and cost loss in a coordinated way.

Let $m(v)$ be the miss penalty of object O with respect to node v, which is given by

$$m(v) = \sum_{(u_1, u_2) \in E[v \rightarrow v']} c(u_1, u_2) \tag{2.1}$$

where v' is the nearest higher level node of v that stores a copy of object O (see Figure 2.1). Therefore, the *cost saving* for node $v \in P$ denoted by $s(v)$ is defined as

$$s(v) = \Big(f(v) - f'(v)\Big)\, m(v) \tag{2.2}$$

where $f'(v)$ is the total access frequency of object O that can still be served by the original caches on the downstream of node v if the copy of object O stored at node v is removed. For instance, in Figure 2.1, $f'(1) = f(3)$, $f'(2) = f(5)$, $f'(3) = f(4) + f(6)$, $f'(4) = f(7) + f(8)$, $f'(5) = f'(6) = f'(7) = f'(8) = 0$.

Let $l(v)$ be the cost loss for storing a copy of object O at node v. Computing $l(v)$ is a bit more complicated. Suppose that $O_1, O_2, \cdots, O_\alpha$ are cached at node v. Obviously, the removed objects should introduce the least total cost loss while making enough room to accommodate the object to be cached. We apply the following greedy heuristic to decide replacement candidates. Note that the normalized cost loss (NCL, i.e., the cost loss introduced by creating one unit of free space) of ejecting O_i at v is $\frac{f_i(v) m_i(v)}{e_i}$, where $f_i(v)$ is the access frequency for object O_i observed at node v, $m_i(v)$ is the miss penalty of object O_i with respect to node v, and e_i is the size of object O_i. The objects in the cache are ordered by their NCLs and are selected sequentially, starting from the object with the smallest NCL, until enough space is created. The cost loss of caching an object at a node is calculated by summing the cost losses caused by all the selected objects. Thus, the *cost gain* for a single node v denoted by $g(v)$ is defined as

$$g(v) = s(v) - l(v) = \Big(f(v) - f'(v)\Big)\, m(v) - l(v) \tag{2.3}$$

Based on the cost gain for a single node, we formulate the general problem for

$CERWOC$ as an optimization problem as follows:

$$\begin{cases} \max\limits_{P} G(T,P) = \max\limits_{P} \left\{ \sum\limits_{v \in P} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \right\} \\ s.t.\ \mathcal{C} \end{cases} \tag{2.4}$$

where $\mathcal{C}$ is called the constraint space.

For unconstrained coordinated en-route web object caching ($U-CERWOC$), $\mathcal{C}$ is null, i.e., there is no constraint. For constrained coordinated en-route web object caching ($C-CERWOC$), we consider the following descriptions of $\mathcal{C}$:

- Non-negative cost gain per node
  $\mathcal{C}$: $\left(f(v) - f^{'}(v) \right) m(v) - l(v) \geq \alpha_v \geq 0 \ \ (\forall\, v \in P)$
  This constraint states that an object may be cached at node v if the cost gain for caching it at v is above a predefined threshold. Clearly, this threshold should be non-negative to avoid any possible cost loss, making the caching practically beneficial.

- Placing exactly k copies
  $\mathcal{C}$: $|P| = k$
  This constraint is to restrict the number of copies to be distributed. Since in practice caching an object will also generate overheads, such as maintaining consistency between caches and the content server, it is necessary to discuss the case of placing a fixed number of copies.

- Placing at most k copies
  $\mathcal{C}$: $|P| \leq k$
  This constraint sets an up-bound on the number of copies to be placed, and allows freedom of placement within this bound to maximize the total cost gain. It can be viewed as an extension of the previous constraint, but requires different technical treatment.

In Equation (2.4), $G(T,P)$ is the overall cost gain for placing copies of an object at each node in P in the constraint space. Regarding the solution to the constrained cases, we give the following definitions. A placement P is called a feasible solution if and only if P satisfies the relevant constraints. For example, if a placement P is a feasible solution to the constrained problem of non-negative cost gain per node, then we have $\left(f(v) - f^{'}(v) \right) m(v) - l(v) \geq \alpha_v \ (\forall\, v \in P)$. On the contrary, if we have $\left(f(v) - f^{'}(v) \right) m(v) - l(v) \geq \alpha_v \ (\forall\, v \in P)$, then the placement P is a feasible solution to the constrained problem of non-negative cost gain per node. A placement P^* is called an optimal solution if and only if P^* is a feasible solution and satisfies that $G(T,P^*) = \max\limits_{P}\{G(T,P)\}$.

2.1.2 Optimal Solutions for Tree Networks

In the following, we focus on unconstrained coordinated en-route web object caching ($U-CERWOC$) and constrained coordinated en-route web object caching ($C-CERWOC$) for tree networks.

- $U-CERWOC$

Now we begin to present an optimal solution for the problem for $U-CERWOC$ for tree networks. Based on Equation (2.4), the problem for $U-CERWOC$ for tree T_w is defined as follows:

$$\max_{A_w} G(T_w, A_w) = \max_{A_w} \left\{ \sum_{v \in A_w} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \right\} \tag{2.5}$$

where $A_w \subseteq D(w)$ and $f(v)$, $f^{'}(v)$, $m(v)$, and $l(v)$ are the same as defined in Section 2.1.1. Suppose that A_w^* is an optimal solution to Equation (2.5) with respect to tree T_w, then we should store a copy of an object at each node in A_w^*, and $G(T_w, A_w^*)$ represents the relevant maximum overall cost gain.

Let $T_{w,x}$ be a subtree of T_w, whose node set is $V[w \rightarrow x] \cup D(x)$, where $x \in D(w)$. Similarly, we define the problem for $U-CERWOC$ for tree $T_{w,x}$ as follows:

$$\max_{A_{w,x}} G(T_{w,x}, A_{w,x}) = \max_{A_{w,x}} \left\{ \sum_{v \in A_{w,x}} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \right\} \tag{2.6}$$

where $A_{w,x} \subseteq D(x) \cup \{x\}$. Suppose that $A_{w,x}^*$ is an optimal solution to Equation (2.6) with respect to tree $T_{w,x}$, then we should store a copy of an object at each node in $A_{w,x}^*$, and $G(T_{w,x}, A_{w,x}^*)$ represents the relevant maximum overall cost gain.

Before presenting our dynamic programming-based algorithm for solving Equation (2.5), we give the following lemmas or theorems.

Lemma 1 *For tree T_w, if $C(w) = \{w_1, w_2, \cdots, w_m\}$, then we have*

$$G(T_w, \cup_{i=1}^m A_{w,w_i}) = \sum_{i=1}^m G(T_{w,w_i}, A_{w,w_i}) \tag{2.7}$$

where $A_{w,w_i} \subseteq D(w_i) \cup \{w_i\}$, $i = 1, 2, \cdots, m$.

Proof Since $A_{w,w_i} \subseteq D(w_i) \cup \{w_i\}$, we have $\cup_{i=1}^m A_{w,w_i} \subseteq \cup_{i=1}^m (D(w_i) \cup \{w_i\}) = D(w)$. Since $A_{w,w_i} \cap A_{w,w_j} = \phi$ for $i \neq j$, by the definition of $G(T_w, A_w)$, we have

$$\begin{aligned} G(T_w, \cup_{i=1}^m A_{w,w_i}) &= \sum_{v \in \cup_{i=1}^m A_{w,w_i}} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\ &= \sum_{i=1}^m \sum_{v \in A_{w,w_i}} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\ &= \sum_{i=1}^m G(T_{w,w_i}, A_{w,w_i}). \end{aligned}$$

Hence, the lemma is proven. □

Lemma 2 *If $C(w) = \{w_1, w_2, \cdots, w_m\}$ and $A_{w,w_i}^{'} = A_w^* \cap (D(w_i) \cup \{w_i\})$, then we have $A_w^* = \cup_{i=1}^m A_{w,w_i}^{'}$, where $A_w^* \subseteq D(w)$ is an optimal solution to Equation(2.5) with respect to tree T_w.*

Proof Since $C(w) = \{w_1, w_2, \cdots, w_m\}$ and $A'_{w,w_i} = A^*_w \cap (D(w_i) \cup \{w_i\})$, we have

$$\begin{aligned} \cup_{i=1}^{k} A'_{w,w_i} &= \cup_{i=1}^{k} (A^*_w \cap (D(w_i) \cup \{w_i\})) \\ &= A^*_w \cap \cup_{i=1}^{k} (D(w_i) \cup \{w_i\}) \\ &= A^*_w \cap D(w) = A^*_w. \end{aligned}$$

Hence, the lemma is proven. □

Theorem 1 *For tree T_w, if $C(w) = \{w_1, w_2, \cdots, w_m\}$, then we have*

$$A^*_w = \cup_{i=1}^{m} A^*_{w,w_i} \tag{2.8}$$

where $A^*_w \subseteq D(w)$ is an optimal solution to Equation(2.5) with respect to tree T_w, and $A^*_{w,w_i} \subseteq D(w_i) \cup \{w_i\}$ is an optimal solution to Equation(2.6) with respect to tree T_{w,w_i}, $i = 1, 2, \cdots, m$.

Proof For $A^*_{w,w_i} \subseteq D(w_i) \cup \{w_i\}$, we have $\cup_{i=1}^{m} A^*_{w,w_i} \subseteq \cup_{i=1}^{m} (D(w_i) \cup \{w_i\}) = D(w)$. Since A^*_w is an optimal solution to Equation(2.5) with respect to tree T_w, we have $G(T_w, A^*_w) \geq G(T_w, \cup_{i=1}^{m} A^*_{w,w_i})$. Let $A'_{w,w_i} = A^*_w \cap (D(w_i) \cup \{w_i\})$, by Lemma 2, then we have $A^*_w = \cup_{i=1}^{m} A'_{w,w_i}$. Obviously, $A'_{w,w_i} \subseteq D(w_i) \cup \{w_i\}$, so we have $G(T_{w,w_i}, A'_{w,w_i}) \leq G(T_{w,w_i}, A^*_{w,w_i})$ since $A^*_{w,w_i} \subseteq D(w_i) \cup \{w_i\}$ is an optimal solution to Equation (2.6) with respect to tree T_{w,w_i}. By Lemma 1, we have

$$\begin{aligned} G(T_w, A^*_w) &= G(T_w, \cup_{i=1}^{m} A'_{w,w_i}) = \sum_{i=1}^{m} G(T_{w,w_i}, A'_{w,w_i}) \\ &\leq \sum_{i=1}^{m} G(T_{w,w_i}, A^*_{w,w_i}) = G(T_w, \cup_{i=1}^{m} A^*_{w,w_i}). \end{aligned}$$

Therefore, we have $G(T_w, A^*_w) = G(T_w, \cup_{i=1}^{m} A^*_{w,w_i})$, so we have $A^*_w = \cup_{i=1}^{m} A^*_{w,w_i}$. Hence, the theorem is proven. □

Theorem 2 *For tree $T_{w,x}$, if $C(x) = \{x_1, x_2, \cdots, x_k\}$, then we have*

$$A^*_{w,x} = \begin{cases} \cup_{i=1}^{k} A^*_{w,x_i} & G(T_{w,x}, \cup_{i=1}^{k} A^*_{w,x_i}) \geq G(T_{w,x}, A^*_x \cup \{x\}) \\ A^*_x \cup \{x\} & G(T_{w,x}, \cup_{i=1}^{k} A^*_{w,x_i}) < G(T_{w,x}, A^*_x \cup \{x\}) \end{cases} \tag{2.9}$$

where $A^*_{w,x} \subseteq D(x) \cup \{x\}$ is an optimal solution to Equation(2.6) with respect to tree $T_{w,x}$, $A^*_x \subseteq D(x)$ is an optimal solution to Equation(2.5) with respect to tree T_x, and $A^*_{w,x_i} \subseteq D(x_i) \cup \{x_i\}$ is an optimal solution to Equation(2.6) with respect to tree T_{w,x_i}, $i = 1, 2, \cdots, k$.

Proof It is easy to see that $A^*_x \cup \{x\} \subseteq (D(x) \cup \{x\})$ and $\cup_{i=1}^{m} A^*_{w,x_i} \subseteq (D(x) \cup \{x\})$. For node x, we consider the following two cases. One case is that a copy of an object is stored at node x, i.e. $x \in A^*_{w,x}$ and the other case is that no copy of that object is placed there, i.e. $x \notin A^*_{w,x}$.

(1) First, we prove $A^*_{w,x} = A^*_x \cup \{x\}$ for $x \in A^*_{w,x}$. Let $A^{'}_{x,x_i} = A^*_{w,x} \cap (D(x_i) \cup \{x_i\})$, by Lemma 2, then we have $A^*_{w,x} = \cup_{i=1}^k A^{'}_{x,x_i} \cup \{x\}$. Therefore, we have

$$
\begin{aligned}
&G(T_{w,x}, A^*_{w,x}) \\
&\quad = G(T_{w,x}, \cup_{i=1}^k A^{'}_{x,x_i} \cup \{x\}) \\
&\quad = \sum_{v \in (\cup_{i=1}^k A^{'}_{x,x_i} \cup \{x\})} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\
&\quad = \sum_{v \in \cup_{i=1}^k A^{'}_{x,x_i}} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right] \\
&\quad = \sum_{i=1}^k \sum_{v \in A^{'}_{x,x_i}} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right] \\
&\quad = \sum_{i=1}^k G(T_{x,x_i}, A^{'}_{x,x_i}) + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right] \\
&\quad = G(T_x, \cup_{i=1}^k A^{'}_{w,x_i}) + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right] \\
&\quad \leq G(T_x, A^*_x) + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right]
\end{aligned}
$$

On the other hand, we have

$$
\begin{aligned}
G(T_{w,x}, A^*_{w,x}) &\geq G(T_{w,x}, A^*_x \cup \{x\}) \\
&= \sum_{v \in A^*_x} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right] \\
&= G(T_x, A^*_x) + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right].
\end{aligned}
$$

Therefore, we have $G(T_{w,x}, A^*_{w,x}) = G(T_{w,x}, A^*_x \cup \{x\})$, so we have $A^*_{w,x} = A^*_x \cup \{x\}$ for $x \in A^*_{w,x}$.

(2) Now, we prove $A^*_{w,x} = \cup_{i=1}^m A^*_{w,x_i}$ for $x \notin A^*_{w,x}$. Let $A^{'}_{w,x_i} = A^*_{w,x} \cap (D(x_i) \cup \{x_i\})$, by Lemma 2, then we have $A^*_{w,x} = \cup_{i=1}^k A^{'}_{w,x_i}$, therefore, we have

$$
\begin{aligned}
G(T_{w,x}, A^*_{w,x}) &= G(T_{w,x}, \cup_{i=1}^k A^{'}_{w,x_i}) \\
&= \sum_{v \in \cup_{i=1}^k A^{'}_{w,x_i}} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\
&= \sum_{i=1}^k \sum_{v \in A^{'}_{w,x_i}} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\
&= \sum_{i=1}^k G(T_{w,x_i}, A^{'}_{w,x_i}) \\
&\leq \sum_{i=1}^k G(T_{w,x_i}, A^*_{w,x_i}) \\
&= G(T_{w,x}, \cup_{i=1}^k A^*_{w,x_i}).
\end{aligned}
$$

Since $G(T_{w,x}, A^*_{w,x}) \geq G(T_{w,x}, \cup_{i=1}^k A^*_{w,x_i})$, we have $G(T_{w,x}, A^*_{w,x}) = G(T_{w,x}, \cup_{i=1}^k A^*_{w,x_i})$. So we have $A^*_{w,x} = \cup_{i=1}^k A^*_{w,x_i}$ for $x \notin A^*_{w,x}$.

From (1) and (2), we can know that

$$G(T_{w,x}, A^*_{w,x}) = \max\{G(T_{w,x}, A^*_x \cup \{x\}), G(T_{w,x}, \cup_{i=1}^k A^*_{w,x_i})\}$$

Therefore, it is easy to see that the theorem is correct. □

By Theorem 2, we can see that for tree $T_{w,x}$, if $G(T_{w,x}, \cup_{i=1}^k A^*_{w,x_i}) \geq G(T_{w,x}, A^*_x \cup \{x\})$, then we do not store a copy of an object at node x and further consider the subtrees $\{T_{w,x_i}, i = 1, 2, \cdots, k\}$, where $C(x) = \{x_1, x_2, \cdots, x_k\}$. Otherwise, we store a copy at node x and further consider the subtree T_x.

Based on theorems 1 and 2, we can present the dynamic programming-based algorithm for $U - CERWOC$ for tree networks as follows:

Algorithm 1: $U - CERWOC$ for Tree T_w

Step 1. Initialization:
$A^*_w = \phi$ and $G(T_w, A^*_w) = 0$;
Step 2. End condition
if $D(w) = \phi$ then return;
Step 3. Recursive procedure
for $v \in C(w)$ do
  if $D(v) = \phi$ then
    if $f(v)c(w, v) - l(v) > 0$ then
      $A^*_{w,v} = \{v\}$
    else
      $A^*_{w,v} = \phi$
  else
    for $x \in C(v)$ do
      if $\sum_{x \in C(v)} G(T_{w,x}, A^*_{w,x}) \geq G(T_v, A^*_v) + (f(v) - f'(v))c(w, v) - l(v)$ then
        $A^*_{w,v} = \cup_{x \in C(v)} A^*_{w,x}$
      else
        $A^*_{w,v} = A^*_v \cup \{v\}$ (According to Theorem 2)
  $A^*_w = \cup_{v \in C(w)} A^*_{w,v}$ (According to Theorem 1)

Now we give in Figure 2.2 a simple example to show how Algorithm 1 works. First, we decompose tree T_0 into two subtrees, $T_{0,1}$ and $T_{0,2}$. For tree $T_{0,1}$, we can get an optimal placement by calculating $g(1)$ directly. For tree $T_{0,2}$, we can further decompose it into either subtrees $T_{0,3}$ and $T_{0,4}$ or tree T_2 according to the relationship between $G(T_{0,3}, A^*_{0,3}) + G(T_{0,4}, A^*_{0,4})$ and $G(T_2, A^*_2) + g(2)$. Obviously, $G(T_{0,3}, A^*_{0,3})$ and $G(T_{0,4}, A^*_{0,4})$ can be solved directly, therefore, what we should do is to further decompose tree T_2 until $G(T_2, A^*_2)$ can be solved directly. Accordingly, we can obtain an optimal placement for tree T_0.

From Algorithm 1, we can know that every cache should maintain some information on the objects, including size, access frequency, update frequency, and miss penalty with the associated node. Fortunately, it is not necessary to store the information of an object at all the nodes in the network. The following theorem describes an important property of Algorithm 1.

Theorem 3 *If A^*_w is an optimal solution to Equation (2.5) with respect to tree T_w, then we have $f(v)c(v, w) - l(v) \geq 0$, $\forall\, v \in A^*_w$.*

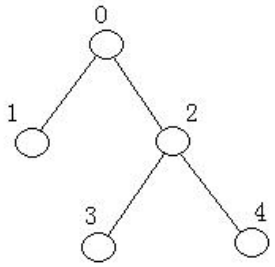

Figure 2.2: A Simple Example

Proof Suppose there exists $x \in A_w^*$ that satisfies $f(x)c(x,w) - l(x) < 0$, then we have

$$\begin{aligned}
G(T_w, A_w^*) &= \sum_{v \in A_w^*} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\
&= \sum_{v \in (A_w^* - \{x\})} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] + \left[\left(f(x) - f^{'}(x) \right) c(x, x^{'}) - l(x) \right] \\
&< \sum_{v \in (A_w^* - \{x\})} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] + [f(x)c(x,w) - l(x)] \\
&< \sum_{v \in (A_w^* - \{x\})} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\
&= G(T_w, A_w^* - \{x\}),
\end{aligned}$$

which contradicts the fact that A_w^* is an optimal solution to Equation (2.5) with respect to tree T_w. Hence, the theorem is proven. □

From Theorem 3, we easily can see that we should consider placing a copy of an object only among the caches where object caching is locally beneficial. Here, "locally beneficial" means that the cost gain is greater than zero if we put only one copy of an object among the en-route caches.

Regarding the time complexity of Algorithm 1, we have the following theorem.

Theorem 4 *If all nodes are locally beneficial, then the time complexity of Algorithm* 1 *is* $O(n^2)$*, where* n *is the total number of nodes in the network.*

Proof From Algorithm 1, we can easily know that the time complexity of it is $O(\sum_{v \in V} |D(v)|)$, where $|D(v)|$ is cardinality of the set $D(v)$. Since $|D(v)| \leq n-1$, we have $O(\sum_{v \in V} |D(v)|) \leq O(\sum_{v \in V}(n-1)) = O(n(n-1)) = O(n^2)$. Hence, the theorem is proven. □

- $C - CERWOC$

In the following, we concentrate on solving the problem for $C-CERWOC$ tree networks. The different constraints include non-negative cost gain per node, placing exactly k copies, and placing at most k copies of an object among the en-route caches.

Constraint I: Non-Negative Cost Gain Per Node

Suppose that $B_w \subseteq D(w)$ is a subset of nodes of tree T_w, based on Equation (2.4), the $C-CERWOC$ problem of non-negative cost gain per node for tree T_w is defined as follows:

$$\begin{cases} \max\limits_{B_w} G(T_w, B_w) = \max\limits_{B_w} \left\{ \sum\limits_{v \in B_w} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \right\} \\ s.t. \ \ \left(f(v) - f^{'}(v) \right) m(v) - l(v) \geq \alpha_v \ \ (\forall \, v \in B_w) \end{cases} \tag{2.10}$$

Suppose that B_w^* is an optimal solution to Equation (2.10), then we should store a copy of an object at each node in B_w^*, and $G(T_w, B_w^*)$ represents the relevant maximum overall cost gain.

Before developing the dynamic programming-based algorithm for solving Equation (2.10), we give the following mathematical proofs.

Lemma 3 *If $A_w^* \subseteq D(w)$ is an optimal solution to Equation (2.5) with respect to tree T_w, then we have $\left(f(v) - f^{'}(v) \right) m(v) - l(v) \geq 0, \ \forall \, v \in A_w^*$.*

Proof Suppose that there exists $x \in A_w^*$ that satisfies $\left(f(x) - f^{'}(x) \right) m(x) - l(x) < 0$, then we have

$$\begin{aligned} & G(T_w, A_w^* - \{x\}) \\ & \quad = \sum_{v \in A_w^* - (D(x) \cup \{x\})} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\ & \qquad + \sum_{v \in D(x) \cap A_w^*} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \\ & \quad > \sum_{v \in A_w^* - (D(x) \cup \{x\})} \left[\left(f(v) - f^{'}(v) \right) c(v, v^{'}) - l(v) \right] \\ & \qquad + \left[\left(f(x) - f^{'}(x) \right) m(x) - l(x) \right] \\ & \qquad + \sum_{v \in D(x) \cap A_w^*} \left[\left(f(v) - f^{'}(v) \right) c(v, v^{'}) - l(v) \right] \\ & \quad \geq \sum_{v \in A_w^*} \left[\left(f(v) - f^{'}(v) \right) c(v, v^{'}) - l(v) \right] \text{ [4]} \\ & \quad = G(T_w, A_w^*) \end{aligned}$$

which contradicts the fact that A_w^* is an optimal solution to Equation (2.5) with respect to tree T_w. Hence, the theorem is proven. □

Theorem 5 *If $A_w^* \subseteq D(w)$ is an optimal solution to Equation (2.5) with respect to tree T_w, then A_w^* is also an optimal solution to the following equation:*

$$\begin{cases} \max\limits_{B_w} G(T_w, B_w) = \max\limits_{B_w} \left\{ \sum\limits_{v \in B_w} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) \right] \right\} \\ s.t. \ \ \left(f(v) - f^{'}(v) \right) m(v) - l(v) \geq 0 \ \ (\forall \, v \in B_w) \end{cases} \tag{2.11}$$

[4]This is because $f^{'}(v)$ becomes larger for the nodes of upstream of node x and $c(v, v^{'})$ smaller for the nodes of downstream of node x when a copy of an object is cached at node x.

Proof Since A_w^* is an optimal solution to Equation (2.5) with respect to tree T_w, by Lemma 3, we have $\left(f(v) - f^{'}(v)\right) m(v) - l(v) \geq 0, \forall\, v \in A_w^*$, therefore, A_w^* is a feasible solution to Equation (2.11), so we have $G(T_w, A_w^*) \leq G(T_w, B_w^*)$. Now we suppose $G(T_w, A_w^*) < G(T_w, B_w^*)$. It is obvious that B_w^* is a feasible solution to Equation (2.5), therefore, we have $G(T_w, A_w^*) = G(T_w, A_w^*)$ and $G(T_w, B_w^*) = G(T_w, B_w^*)$, so we have $G(T_w, A_w^*) < G(T_w, B_w^*)$. This contradicts the fact that A_w^* is an optimal solution to Equation (2.5) with respect to tree T_w. Hence, the theorem is proven. □

By Theorem 5, we can obtain an optimal solution to Equation (2.11) by solving Equation (2.5). It is easy to see that Equation (2.10) can be transferred into the following equation:

$$\begin{cases} \max\limits_{B_w} G(T_w, B_w) = \max\limits_{B_w} \left\{ \sum\limits_{v \in B_w} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) - \alpha_v \right] + \sum\limits_{v \in B_w} \alpha_v \right\} \\ s.t.\ \ \left(f(v) - f^{'}(v) \right) m(v) - l(v) - \alpha_v \geq 0\ \ (\forall\, v \in B_w) \end{cases} \tag{2.12}$$

Furthermore, it is obvious that Equation (2.12) is equivalent to the following equation.

$$\begin{cases} \max\limits_{B_w} G(T_w, B_w) = \max\limits_{B_w} \left\{ \sum\limits_{v \in B_w} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) - \alpha_v \right] \right\} \\ s.t.\ \ \left(f(v) - f^{'}(v) \right) m(v) - l(v) - \alpha_v \geq 0\ \ (\forall\, v \in B_w) \end{cases} \tag{2.13}$$

By Theorem 5, we can get an optimal solution to Equation (2.13) by solving the following Equation:

$$\max_{A_w} G(T_w, A_w) = \max_{A_w} \left\{ \sum_{v \in A_w} \left[\left(f(v) - f^{'}(v) \right) m(v) - l(v) - \alpha_v \right] \right\} \tag{2.14}$$

Therefore, we can obtain an optimal solution to Equation (2.10) by solving Equation (2.14). By now, we have proved that the problem for $C - CERWOC$ as described in Equation (2.10) can be transformed into a problem for $U - CERWOC$ as described in Equation (2.14). Based on the algorithm proposed for $U - CERWOC$ (i.e., Algorithm 1), we present the following dynamic programming-based algorithm for $C - CERWOC$ of non-negative cost gain per node, which is described as follows:

Algorithm 2: $C - CERWOC$—Non-Negative Cost Gain Per Node

Step 1. Initialization
$A_w^* = \phi$ and $G(T_w, A_w^*) = 0$;
Step 2. End Condition
if $D(w) = \phi$ then return;
Step 3. Recursive Procedure
for $v \in C(w)$ do
  if $D(v) = \phi$ then
    if $f(v)c(w,v) - l(v) - \alpha_v > 0$ then
      $A_{w,v}^* = \{v\}$
    else
      $A_{w,v}^* = \phi$
  else

for $x \in C(v)$ do
  if $\sum_{x \in C(v)} G(T_{w,x}, A^*_{w,x}) \geq G(T_v, A^*_v) + \Big[(f(v) - f^{'}(v))c(w,v) - l(v) - \alpha_v\Big]$ then
    $A^*_{w,v} = \cup_{x \in C(v)} A^*_{w,x}$
  else
    $A^*_{w,v} = A^*_v \cup \{v\}$
$A^*_w = \cup_{v \in C(w)} A^*_{w,v}$

Similar to Algorithm 1, we can know that every cache should also maintain some information on the objects, including size, access frequency, update frequency, and miss penalty with the associated node. For Algorithm 2, the constraint for each node should also be maintained. The following corollary describes an important property of Algorithm 2.

Corollary 1 *If A^*_w is an optimal solution to Equation (2.14), then we have*

$$f(v)c(v,w) - l(v) - \alpha_v \geq 0, \ \forall\, v \in A^*_w.$$

The proof of Corollary 1 is similar to that of Theorem 3.

From Corollary 1, we can easily see that we should consider placing a copy of an object only among the caches where object caching is locally beneficial. Here, "locally beneficial" means that the cost gain for each node is greater than the constraint for that node if we put only one copy of an object among the caches.

Regarding the time complexity of Algorithm 2, we have the following corollary.

Corollary 2 *If all nodes are locally beneficial, then the time complexity of Algorithm* 2 *is $O(n^2)$, where n is the total number of nodes in the network.*

The proof of Corollary 2 is similar to that of Theorem 4.

Constraint Ⅱ: Placing Exactly k Copies of An Object

Similarly, the $C-CERWOC$ problem of placing exactly k copies of an object among the en-route caches for tree T_w is defined as follows:

$$\begin{cases} \max\limits_{B_w} G(T_w, B_w) = \max\limits_{B_w} \left\{ \sum\limits_{v \in B_w} \Big[\Big(f(v) - f^{'}(v) \Big) m(v) - l(v) \Big] \right\} \\ s.t. \ |B_w| = k \end{cases} \tag{2.15}$$

Suppose that A^*_w is an optimal solution to Equation (2.5) with respect to tree T_w, we can easily know that it is not necessary to place more than k^* copies of an object among the en-route caches, where $k^* = |A^*_w|$. Otherwise, there must be at least one node whose cost gain is negative. Therefore, k should be less than k^*. So we first compute k^* by algorithm 2 by setting $\alpha_v = 0$, and the optimal locations are all the nodes in A^*_w when $k \geq k^*$. The relationship between the overall cost gain and the number of copies can be visualized from Figure 2.3.

Before presenting the algorithm for solving the problem of placing exactly k copies of an object among the en-route caches for tree T_w, we give the following definition. The local cost gain [5] for node v denoted by $h(v)$ is defined as follows:

$$h(v) = f(v)c(v,w) - l(v) \tag{2.16}$$

[5]The local cost gain is the cost gain for placing only one copy of an object among the en-route caches.

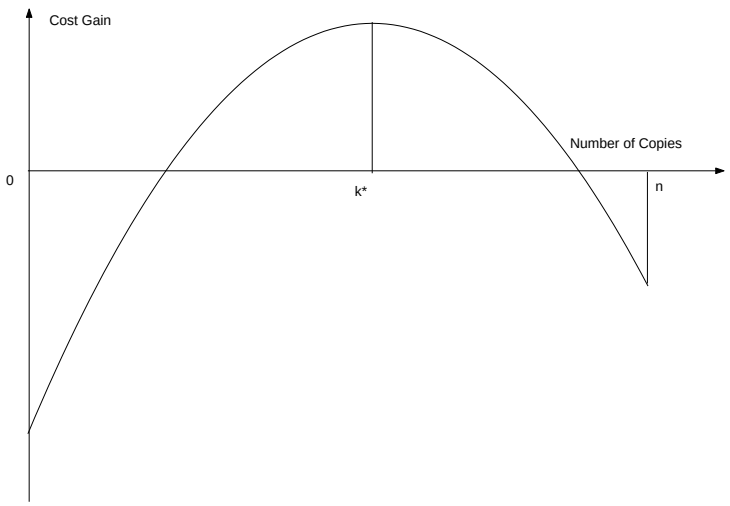

Figure 2.3: Relationship between Cost Gain and Number of Copies

Let $\alpha_{max} = \max_{v \in V}\{h(v)\}$, then we have the following theorem with respect to the feasible solution to Equation (2.10).

Theorem 6 *if $\alpha_v > \alpha_{max}$, then there is no feasible solution to Equation (2.10).*

Proof Suppose that there exists a node that satisfies $\left(f(v) - f^{'}(v)\right) m(v) - l(v) \geq \alpha_v$. On the other hand, we have

$$\begin{aligned} \left(f(v) - f^{'}(v)\right) m(v) - l(v) &\leq \left(f(v) - f^{'}(v)\right) c(v,w) - l(v) \\ &\leq f(v)c(v,w) - l(v) \leq h(v) < \alpha_v. \end{aligned}$$

So, the supposition is not correct. Hence, the theorem is proven. □

From Theorem 6, we can know that the parameter α_v in Equation (2.10) should satisfy: $0 \leq \alpha_v \leq \alpha_{max}$. It is obvious that the optimal number of copies of an object to be placed in the network is relevant to the parameter α_v. Hence, the proper selection of α_v determines this number. The crucial observation is that this number is a monotonically decreasing function of α_v, that is, as α_v increases, it decreases monotonically. Therefore, we can determine the optimal locations for placing exactly k copies of an object among the en-route caches by tuning the parameter α_v. The relationship between the optimal number of copies k^* and the parameter α_v can be visualized in Figure 2.4.

The algorithm for placing exactly k copies of an object among the en-route caches is described as follows:

Algorithm 3: $C-CERWOC$—Placing Exactly k Copies of An Object

Step 1. Initialization
$\alpha_{min} = 0$; $\alpha_{max} = \max_{v \in V}\{h(v)\}$, $k^* = |A_w^*|$.
Step 2. Recursive Procedure
while $k^* \neq k$ do
  $\alpha = (\alpha_{min} + \alpha_{max})/2$;
  Call Algorithm 2 by setting $\alpha_v = \alpha$;

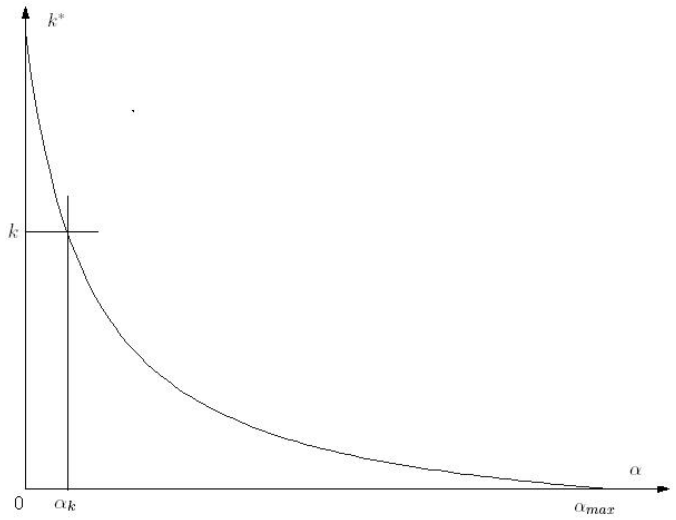

Figure 2.4: Relationship between k^* and α_v

$k^* = |A_w^*|$;
if $k^* > k$ then
    $\alpha_{min} = \alpha$
else
    $\alpha_{max} = \alpha$

We can see that Algorithm 3 converges to an optimal solution to Equation (2.15) quickly and its time complexity can be easily shown to be $O(n^2$, where n is the total number of nodes in the network.

Constraint Ⅲ: Placing at Most k Copies of An Object

Based on Equation (2.4), the $C-CERWOC$ problem of placing at most k copies of an object among the en-route caches for tree T_w is defined as follows:

$$\begin{cases} \max\limits_{B_w} G(T_w, B_w) = \max\limits_{B_w} \left\{ \sum\limits_{v \in B_w} \left[\Big(f(v) - f^{'}(v) \Big) m(v) - l(v) \right] \right\} \\ s.t.\ |B_w| \leq k \end{cases} \tag{2.17}$$

Suppose that $k^* = |A_w^*|$, where A_w^* is the optimal solution to Equation (2.10) by setting $\alpha_v = 0$, then we have the following theorem on the relationship between Equation (2.17) and Equation (2.5).

Theorem 7 *if $k \geq k^*$, then Equation (2.17) is equivalent to Equation (2.5)* [6].

Proof Suppose that A_w^* is an optimal solution to Equation (2.5) with respect to tree T_w and B_w^* an optimal solution to Equation (2.17). (1) By the definition of optimal solution, we easily know that A_w^* is an optimal solution to Equation (2.17) since $k \geq k^*$. (2) Now we prove B_w^* is an optimal solution to Equation (2.5). We apply reduction to absurdity, and we have $G(T_w, A_w^*) < G(T_w, B_w^*)$. Since A_w^* is a feasible solution to

[6]Equivalent means that the optimal solution to Equation (2.17) is an optimal solution to Equation (2.5) and the optimal solution to Equation (2.5) is also an optimal solution to Equation (2.17).

Equation (2.17), we have $G(T_w, A_w^*) = G(T_w, A_w^*)$ and $G(T_w, B_w^*) = G(T_w, B_w^*)$, so we have $G(T_w, A_w^*) < G(T_w, B_w^*)$. This contradicts the fact that A_w^* is an optimal solution to Equation (2.5) with respect to tree T_w. Hence, the theorem is proven. □

From Theorem 7, we can see that the problem as described in Equation (2.5) can be viewed as a special case of the problem being discussed in this subsection by setting $k = n$.

Based on Algorithm 3 , we can present the following algorithm for placing at most k copies of an object.

Algorithm 4**:** $C - CERWOC$—Placing at Most k Copies of An Object

Step 1. Initialization
gain:=0, placement= ϕ.
Step 2. Recursive Procedure
for $i = 0$ to k do
    Call Algorithm 3 by setting $k = i$;
    gain(i) = $G(T_w, B_w^*)$;
    if gain(i) > gain then
        gain=gain(i)
        placement=B_w^*

We can see that Algorithm 4 converges to an optimal solution to Equation (2.17) very quickly and the time complexity of Algorithm 4 is $O(kn^2$, where n is the total number of nodes in the network.

2.1.3 Optimal Solution for Autonomous Systems

In this section, we focus on solving the problem for $CERWOC$ for autonomous systems, determining the locations for placing exactly k copies of an object among the en-route caches such that the overall cost gain is maximized.

Autonomous systems play an important role in routing objects in the internet [8, 73]. In this section, we assume that the network topology for each autonomous system is a tree. We denote the whole network by $T_{AS} = (V_{AS}, E_{AS})$, where V_{AS} is the set of the nodes and E_{AS} is the set of the links. We assume that there are m ASes in the network, each of which is represented by tree $T_i, i = 1, 2, \cdots, m$. Figure 2.5 shows a simple example of such an autonomous system in which the replicas have the same contents as the server. We denote the set of the replica servers and the content server by S_{AS}.

Based on Equation (2.4), the problem for $CERWOC$ for autonomous systems is defined as follows:

$$\begin{cases} \max\limits_{P_{AS}} G(T_{AS}, P_{AS}) = \max\limits_{P_{AS}} \left\{ \sum\limits_{v \in P_{AS}} \left[\Big(f(v) - f^{'}(v) \Big) m(v) - l(v) \right] \right\} \\ s.t.\ |P_{AS}| = k \end{cases} \tag{2.18}$$

where $P_{AS} \subseteq V_{AS} - S_{AS}$.

From Equation (2.18), we can see that this problem degenerates to the problem addressed in Equation (2.15) when $m = 1$, i.e., determining the optimal locations for placing k copies of an object in tree networks. In this section, we also use $\widetilde{G}(T_{AS}, k)$ to denote the overall maximum cost gain for placing k copies of an object in T_{AS} for convenience, i.e.,

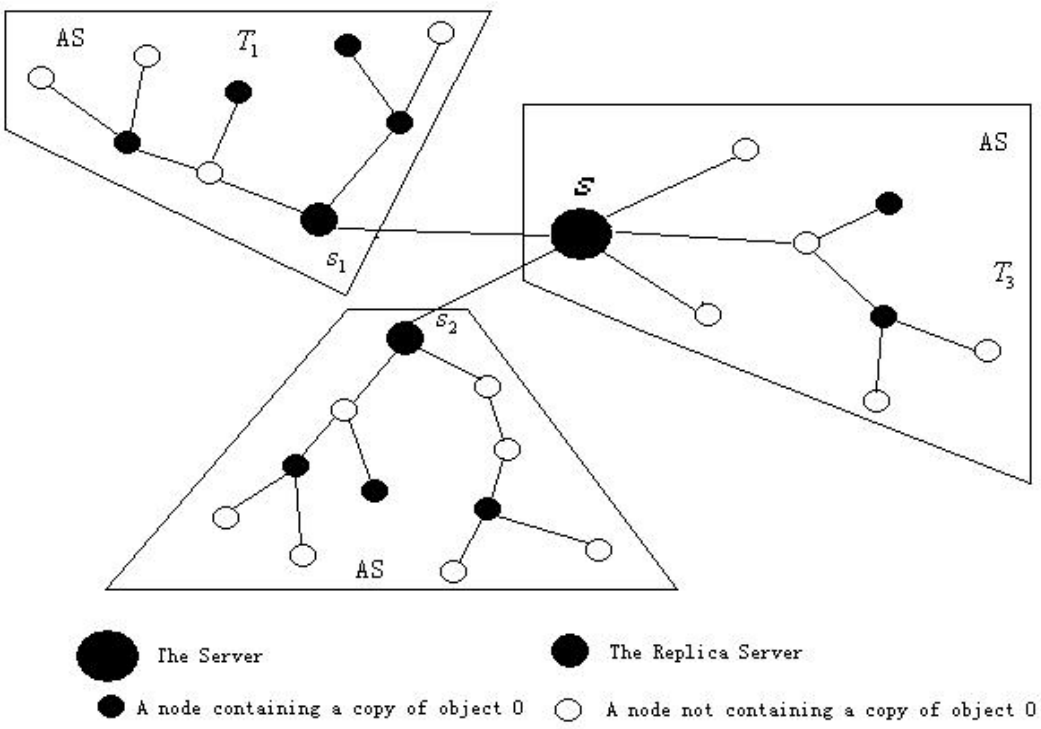

Figure 2.5: En-Route Web Caching for Autonomous Systems

$\widetilde{G}(T_{AS}, k) = G(T_{AS}, P^*_{AS})$, where $G(T_{AS}, P^*_{AS})$ is an optimal solution to Equation (2.18) and $k^* = |P^*_{AS}|$.

Now we apply the following idea to solve this problem, which is similar to that presented in [44]. We first divide T_{AS} into two parts: $\cup_{i=1}^{m-1} T_i$ and T_m; thus, we consider the problem of placing k_m copies of an object in the first part and $k - k_m$ copies of an object in the second part, where $0 \leq k_m \leq k$. Then, we divide $\cup_{i=1}^{m-1} T_i$ into two parts: $\cup_{i=1}^{m-2} T_i$ and T_{m-1}; thus, we consider the problem of placing k_{m-1} copies of an object in the first part and $k_m - k_{m-1}$ copies of an object in the second part, where $0 \leq k_{m-1} \leq k_m$. We repeat this process until there is only one tree left. Regarding the recursive process, we have the following theorem.

Theorem 8

$$\widetilde{G}(T_{+i}, k) = \begin{cases} \widetilde{G}(T_1, k) & \text{if } T_{+i} = T_1 \\ \max\limits_{0 \leq k' \leq k} \{\widetilde{G}(T_{+(i-1)}, k') + \widetilde{G}(T_i, k - k')\} & \text{if } T_{+i} \neq T_1 \end{cases} \tag{2.19}$$

where $T_{+i} = \cup_{j=1}^{i} T_j$.

Proof When $T_{+i} = T_1$, it becomes the $C - CERWOC$ problem of placing k copies of an object among the en-route caches for tree topology; therefore, it is obviously correct.

Now we consider $T_{+i} \neq T_1$. Let $\widetilde{G}'(T_{+i}, k) = \max\limits_{0 \leq k' \leq k} \{\widetilde{G}(T_{+(i-1)}, k') + \widetilde{G}(T_i, k - k')\}$. We first prove $\widetilde{G}'(T_{+i}, k) \geq \widetilde{G}(T_{+i}, k)$. Suppose that P^* is an optimal solution for placing

k copies of an object in T_{+i}, then we have $\widetilde{G}(T_{+i},k) = G(T_{+i},P^*)$. Suppose that $P^* \cap T_{+(i-1)} = l$, then we have $P^* \cap T_i = k-l$, therefore we have

$$G(T_{+i},P^*) = G(T_{+(i-1)}, P^* \cap T_{+(i-1)}) + G(T_i, P^* \cap T_i) \tag{2.20}$$

It is easy to see that $P^* \cap T_{+(i-1)}$ and $P^* \cap T_i$ are optimal placements for placing l copies and $k-l$ copies in $T_{+(i-1)}$ and T_i respectively. Otherwise, there should be a better placement than $P^* \cap T_{+(i-1)}$ or $P^* \cap T_i$, which would contradict that P^* is an optimal placement. Therefore, we have

$$\begin{aligned}
\widetilde{G}(T_{+i},k) &= G(T_{+i},P^*) \\
&= G(T_{+(i-1)}, P^* \cap T_{+(i-1)}) + G(T_i, P^* \cap T_i) \\
&= \widetilde{G}(T_{+(i-1)},l) + \widetilde{G}(T_i,k-l) \\
&\leq \max_{0\leq l^{'}\leq k} \{\widetilde{G}(T_{+(i-1)},l^{'}) + \widetilde{G}(T_i,k-l^{'})\} \\
&= \widetilde{G}^{'}(T_{+i},k).
\end{aligned}$$

Now we prove $\widetilde{G}^{'}(T_{+i},k) \leq \widetilde{G}(T_{+i},k)$. Suppose that $\widetilde{G}(T_{+(i-1)},l) + \widetilde{G}(T_i,k-l)\} = \max_{0\leq l^{'}\leq k} \{\widetilde{G}(T_{+(i-1)},l^{'}) + \widetilde{G}(T_i,k-l^{'})\}$. Let $P^*_{+(i-1)}$ be an optimal placement for placing l copies of an object in $T_{+(i-1)}$ and P^*_i an optimal placement for placing $k-l$ copies of an object in T_i, then for any l, we have

$$\begin{aligned}
\widetilde{G}(T_{+i},k) &\geq G(T_{+i}, P^*_{+(i-1)} \cup P^*_i) \\
&= G(T_{+(i-1)}, P^*_{+(i-1)}) + G(T_i, \cup P^*_i) \\
&= \widetilde{G}(T_{+(i-1)},l) + \widetilde{G}(T_i,k-l),
\end{aligned}$$

therefore, we have $\widetilde{G}(T_{+i},k) \geq \max_{0\leq l^{'}\leq k} \{\widetilde{G}(T_{+(i-1)},l^{'}) + \widetilde{G}(T_i,k-l^{'})\} = \widetilde{G}^{'}(T_{+i},k)$. Hence, the theorem is proven. □

The algorithm for $CERWOC$ for autonomous systems is described as follows:

Algorithm 5**:** $CERWOC$ for Autonomous Systems

Main Procedure
for $i = 1$ to m do (Initialization)
  for $j = 0$ to $k-m+i$ do
    $\widetilde{G}(T_{+i},j) = -1$; (The cost gain for placing j copies in T_{+i})
    $P(T_i,j) = \phi$ (The optimal solution for placing j copies in T_i)
Call $Placement(T_{+m},k)$; (Calling procedure $Placement$ recursively)
Procedure $Placement(T_{+i},l)$
if $\widetilde{G}(T_{+i},l) \geq 0$ then
  Return $\widetilde{G}(T_{+i},l)$; ($\widetilde{G}(T_{+i},l)$ computed)
if $i = 1$ then
  Return $\widetilde{G}(T_1,l)$; (Call Algorithm 3 since $\widetilde{G}(T_1,l) = G(T_1,A^*_1)$)
$TG = 0$;
for $l^{'} = (i-1)$ to $l-1$ do (Finding the optimal number of copies to be placed in T_i)
  TG=$Placement(T_{+(i-1)},l^{'}) + \widetilde{G}(T_i,l-l^{'})$; (According to Theorem 8)

$= Placement(T_{+(i-1)}, l') + G(T_i, A_i^*)$;
if $\widetilde{G}(T_{+i}, l) > TG$ then
$\widetilde{G}(T_{+i}, l) = TG$
$P(T_i, l - l') = A_i^*$

The time complexity of Algorithm 5 follows the following corollary.

Corollary 3 *The time complexity of Algorithm* 5 *is* $O(kn^2)$*, where* n *is the total number of nodes in the network.*

Proof According to Algorithm 3, the time for running $Placement(T_{+i}, l)$ is $O(n_i^2)$, where n_i is the number of nodes of tree T_i and $1 \leq i \leq m$. Since the Algorithm 5 calls $Placement(T_{+i}, l)$ to compute all the elements in $\widetilde{G}(T_{+i}, l)$, its time complexity is $O\left(\sum_{i=1}^{m}\sum_{l=1}^{k}\left(n_i^2\right)\right) = O\left(\sum_{i=1}^{m}\left(kn_i^2\right)\right) \leq O\left(\sum_{i=1}^{m}\left(n_i^2\right)\right) \leq O\left(kn^2\right)$. Hence, the corollary is proven. □

2.1.4 Parameter Estimation

In the actual implementation, the access frequency and the miss penalty of an object with respect to a node are not usually constant. We have to estimate them accurately so that the characteristics of data access can be well captured.

We follow the methods described in [90] for estimating parameters used in our model. The access frequency $f(v)$ is estimated by recent request data. We apply a "sliding window" technique to estimate the access frequency to make our model less sensitive to transient workload [88]. Specifically, $f(v)$ is calculated by $K/(t - t_K)$, where K is the number of accesses recorded, t is the current time, and t_K is the Kth most recently referenced time (the time of the oldest reference in the sliding window). It is shown by Shim *et al.* in [88] that K can be as small as 2 or 3 to achieve the best performance. In the simulation, k is set to 2. If knowledge of access frequency is imprecise, another method can be applied to estimate the average access frequency for object O observed at node v based on the size of this object. Specifically, $f(v)$ is calculated by p/s^b, where p and b are constants for object O at node v, and s is the size of object O [88]. This is shown by Cunha *et al.* in [28] based on the studies that web clients exhibit a strong preference for accessing small objects [28, 39]. The miss penalty is updated by the response messages. Specifically, a variable with an initial value of zero is attached to each object. At each intermediate node along the way, the variable is increased by the cost of the last link the object has just traversed. The value is then used to update the miss penalty of the object maintained by the associated cache. If the object is inserted into the cache, the node resets the value to zero before forwarding the object downstream. In this way, the updated cost loss is disseminated to all the caches on the way.

2.1.5 Simulation Model

We have performed extensive simulation experiments for comparing the results of our model with those of the existing models. Here, we assume that there is only one server. As far as we knew, it is difficult to find true trace data in the open literature to simulate

our model. Similar to the simulation model proposed in [90], we generated the simulation model from empirical results presented in [7, 11, 14].

The network topology is randomly generated by the Tier program [14]. We have conducted experiments for a lot of topologies with different parameters and found that the performance of our model was relatively insensitive to topology changes. Here, we list only the experimental results for one topology because of space limitations. Table 2.1 shows the parameters and their values used in our experiments, where $U(x, y)$ denotes the uniform distribution between x and y.

Table 2.1: Parameters of Our Experiments

Parameter	Value
Total Number of Nodes	300
Number of WAN Nodes	150
Number of MAN Nodes	150
Delay of WAN Links	0.45 second
Delay of MAN Links	0.06 second
Number of Objects	1000
Average Object Size	$30KB$
Average Request Rate Per Node	$U(1, 9)$ requests per second

The WAN (Wide Area Network) is viewed as the backbone network to which no servers or clients are attached. Each MAN (Metropolitan Area Network) node is assumed to connect to the content server. Each MAN and WAN node is associated with an en-route cache. Similar to the studies in [11, 15, 46, 88], we describe cache size as the total relative size of all objects available in the content server. We assume for our experiments that the object sizes follow the distribution as described in [7] and that the average object size is $30KB$. In our experiments, the client at each MAN node randomly generates the requests, and the average request rate of each node follows the distribution of $U(1, 9)$. The cost for each link is calculated by the access latency. For simplicity, the delay caused by sending the request and the relevant response for that request is proportional to the size of the requested object. Here, we consider the average object sizes for calculating all delays, including the propagation delay, the transmission delay, and the searching delay. The cost function is taken to be the delay of the link, which means that the cost in our model is interpreted as the access latency in our simulation.

2.1.6 Experimental Results

In our experiments, we compare the performance results of different models across a wide range of cache sizes, from 0.04 percent to 12 percent, and the performance metrics include the average access latency, the response ratio of a request [7], the object hit ratio [8], the

[7]The response ratio of a request is defined as the ratio of its access latency to the size of the target object.

[8]The object hit ratio is defined as the ratio of the number of requests satisfied by the caches as a whole to the total number of requests.

byte hit ratio [9], and the average server load [10].

In our experiments, we denote the results for the LRU model [102] by LRU, the results for $CERWOC$ for linear topology [90] by LT, and the results for $U-CERWOC$ for tree topology by TT.

Figure 2.6 shows the results of the average access latency and the average response ratio as a function of the relative cache size at each node. As we have known, the lower the average access latency or the average response ratio, the better the performance. We can easily see that the performance results for the three models improve as the relative cache size increases. We can also see that TT can improve both the average access latency and the average response ratio compared to LRU and LT, since our model determines the optimal locations in the whole tree, while LRU places the copies of an object at each en-route cache and LT optimally places the copies of an object on the path from the client to the server.

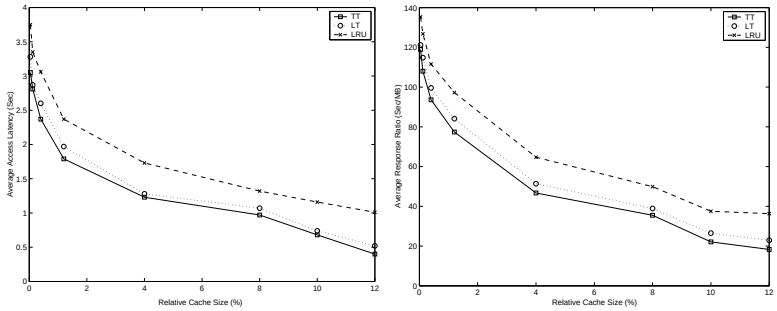

Figure 2.6: Experiment for Average Access Latency and Average Response Ratio

Figure 2.7 shows the results of the object hit ratio and the byte hit ratio as functions of the relative cache size for different models respectively. By computing the optimal locations for the tree topology, we can see that the results for our model can greatly outperform those of the other two models, especially for smaller cache sizes. The object hit ratio and the byte hit ratio steadily improve as the relative cache size increases, which conforms to the fact that more requests will be satisfied by the caches as the cache size becomes larger.

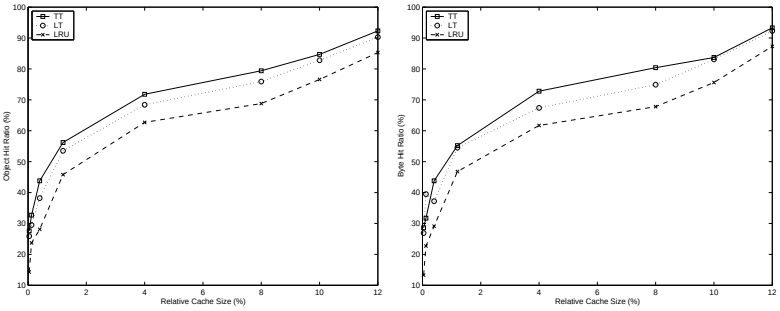

Figure 2.7: Experiment for Object Hit Ratio and Byte Hit Ratio

Figure 2.8 shows the results of the server load as a function of the relative cache size.

[9]The byte hit ratio is defined as the ratio of the bytes of requests satisfied by the caches as a whole to the total bytes of requests.

[10]The average server load is defined as the average number of bytes served by the server per second.

It can be seen that the average server load for our model is lower than that of the other models. We can also see that the average server load decreases as the relatice cache size increases.

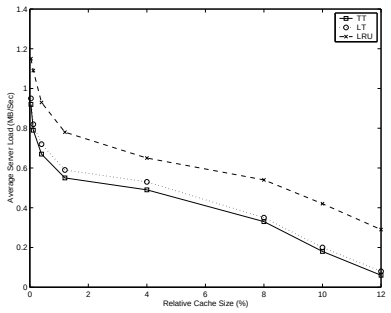

Figure 2.8: Experiment for Average Server Load

Figure 2.9 shows the results of the average response ratio and the object hit ratio as functions of the average number of copies of the objects placed among the en-route caches respectively. We can see that the average response ratio decreases as the number of copies of an object placed among the en-route caches increases. When the number arrives at about 175, the average response ratio begins to decrease slowly. This is true because the optimal number of copies of an object to be placed in such a network topology is approximately 175. We also can see that the object hit ratio decreases with the number of copies of an object placed among the en-route caches increasing. When the number reaches about 190, the object hit ratio starts to decrease. This is true because the optimal number of copies of an object to be placed in such a network topology is approximately 190.

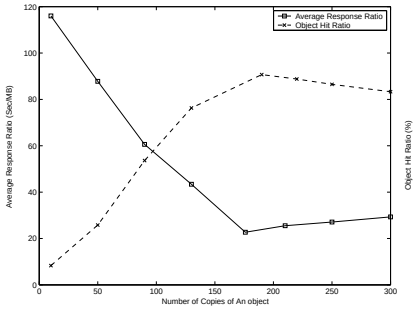

Figure 2.9: Experiment for Different Number of Copies

2.2 Proxy Placement

Besides object caching as discussed in Section 2.1, proxy placement is also an important factor that affects the performance of en-route web caching. In this section, we present optimal solutions for the problem of proxy placement for coordinated en-route web caching as a natural extension of the solution proposed for coordinated en-route web object caching.

2.2.1 Mathematical Model

The notations and definitions used in this section are similar to those introduced in Section 2.1. We still model the network as a connected graph $T = (V, E)$, where V is the set of nodes and E is the set of network links. Figure 2.10 shows an example of a tree topology.

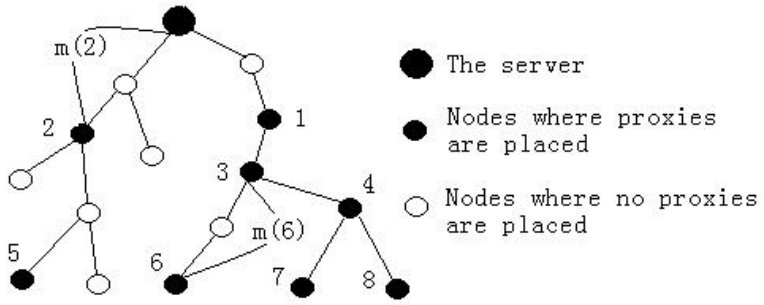

Figure 2.10: Proxy Placement for Coordinated En-Route Web Caching for Tree Networks

Let $P \subseteq V$ be a subset of nodes, at each of which a proxy server is placed. Let $f(v)$ denote the all the requests passing through node v during a certain period of time (including the requests from node v itself and from others), obviously, $f(v) \geq \sum_{w \in C(v)} f(w)$.

As we have known, placing a web proxy at a node makes some requests previously passing it now can be satisfied at it; hence, there will be some access cost gain by placing a proxy server at a node. Thus, we can define the mathematical model for proxy placement problem as an optimization problem by conducting a similar analysis to that in Section 2.1.1 as follows:

$$\max_{P \subseteq V; |P|=k} G(T, P) = \max_{P \subseteq V; |P|=k} \left\{ \sum_{v \in P} \Big(f(v) - f^{'}(v) \Big) m(v) \right\} \tag{2.21}$$

where $G(T, P)$ is the overall access cost gain for placing a proxy server at each node in P, $m(v) = \sum_{(u_1, u_2) \in E[v \rightarrow v']} c(u_1, u_2)$ is the miss penalty of one request with respect to node v, $v^{'}$ is the nearest higher level node of v at which a proxy server is placed (see Figure 2.10), and $f^{'}(v)$ is the total access frequency that can still be served by the original proxies on the downstream of node v if the proxy placed at node v is removed. For instance, in Figure 2.10, $f^{'}(1) = f(3)$, $f^{'}(2) = f(5)$, $f^{'}(3) = f(4) + f(6)$, $f^{'}(4) = f(7) + f(8)$, $f^{'}(5) = f^{'}(6) = f^{'}(7) = f^{'}(8) = 0$.

2.2.2 Dynamic Programming-Based Solution for Tree Networks

In this section, we use T_w to denote a tree, whose root is w. Without loss of generality, we assume that there is only one content server at the root of a tree, at which the objects requested by users are maintained. Based on Equation (2.21), the problem of proxy

placement for tree T_w is defined as follows:

$$\max_{A_w \subseteq D(w); |A_w|=k} G(T_w, A_w) = \max_{A_w \subseteq D(w); |A_w|=k} \left\{ \sum_{v \in A_w} \Big(f(v) - f^{'}(v) \Big) m(v) \right\} \tag{2.22}$$

where $f(v)$, $f^{'}(v)$, $m(v)$, and $l(v)$ are the same as defined in Section 2.2.1. Suppose that A_w^* is an optimal solution to Equation (2.22) with respect to tree T_w, then we should place a proxy server at each node in A_w^*, and $G(T_w, A_w^*)$ represents the relevant maximum overall access cost gain.

Before solving Equation (2.22), we consider the following equations:

$$\begin{cases} \max\limits_{B_w \subseteq D(w)} G(T_w, B_w) = \max\limits_{B_w \subseteq D(w)} \left\{ \sum\limits_{v \in B_w} \Big(f(v) - f^{'}(v) \Big) m(v) \right\} \\ s.t. \ \ \big(f(v) - f^{'}(v) \big) m(v) \geq \alpha \ \ (\forall \, v \in B_w) \end{cases} \tag{2.23}$$

$$\begin{cases} \max\limits_{B_w \subseteq D(w)} G(T_w, B_w) = \max\limits_{B_w \subseteq D(w)} \left\{ \sum\limits_{v \in B_w} \Big[\Big(f(v) - f^{'}(v) \Big) m(v) - \alpha \Big] \right\} + |B_w| \alpha \\ s.t. \ \ \big(f(v) - f^{'}(v) \big) m(v) - \alpha \geq 0 \ \ (\forall \, v \in B_w) \end{cases} \tag{2.24}$$

$$\begin{cases} \max\limits_{B_w \subseteq D(w)} G(T_w, B_w) = \max\limits_{B_w \subseteq D(w)} \left\{ \sum\limits_{v \in B_w} \Big[\Big(f(v) - f^{'}(v) \Big) m(v) - \alpha \Big] \right\} \\ s.t. \ \ \big(f(v) - f^{'}(v) \big) m(v) - \alpha \geq 0 \ \ (\forall \, v \in B_w) \end{cases} \tag{2.25}$$

It is obvious that equations (2.23), (2.24) and (2.25) are equivalent [11].

The following lemma depicts an important property of the solution to the following equation:

$$\max_{A_w \subseteq D(w)} G(T_w, A_w) = \max_{A_w \subseteq D(w)} \left\{ \sum_{v \in A_w} \Big[\Big(f(v) - f^{'}(v) \Big) m(v) - \alpha \Big] \right\} \tag{2.26}$$

Lemma 4 *If A_w^* is an optimal solution to Equation (2.26), then we have*

$$\Big(f(v) - f^{'}(v) \Big) m(v) - \alpha \geq 0, \ \ \forall \, v \in A_w^*.$$

The proof of this lemma is similar to that of Lemma 3.

The following theorem describes the relationship between optimal solutions to Equation (2.25) and Equation (2.26).

Theorem 9 *If A_w^* is an optimal solution to Equation (2.26), then A_w^* is also an optimal solution to Equation (2.25).*

[11] Equivalent means that an optimal solution to one equation is also an optimal solution to the other equations. For Example, an optimal solution to Equation (2.23) is an optimal solution to Equation (2.25) and an optimal solution to Equation (2.25) is also an optimal solution to Equation (2.23).

Proof Since A_w^* is an optimal solution to Equation (2.26), by Lemma 4, we have $\left(f(v) - f^{'}(v)\right) m(v) - \alpha \geq 0, \forall\, v \in A_w^*$; therefore, A_w^* is a feasible solution to Equation (2.25), so we have $G(T_w, A_w^*) \leq G(T_w, B_w^*)$. Now we suppose $G(T_w, A_w^*) < G(T_w, B_w^*)$. It is obvious that B_w^* is a feasible solution to Equation (2.26); therefore, we have $G(T_w, A_w^*) = G(T_w, A_w^*)$ and $G(T_w, B_w^*) = G(T_w, B_w^*)$, so we have $G(T_w, A_w^*) < G(T_w, B_w^*)$. This contradicts the fact that A_w^* is an optimal solution to Equation (2.26). Hence, the theorem is proven. □

By now, we have proved that we can obtain an optimal solution to Equation (2.23) by solving Equation (2.26). Now we start to solve Equation (2.26).

Let $T_{w,x}$ be a subtree of T_w, whose node set is $V[w \to x] \cup D(x)$, where $x \in D(w)$. We define another equation as follows:

$$\max_{A_{w,x} \subseteq D(x) \cup \{x\}; |A_{w,x}| = k} G(T_{w,x}, A_{w,x}) = \max_{A_{w,x} \subseteq D(x) \cup \{x\}; |A_{w,x}| = k} \left\{ \sum_{v \in A_{w,x}} \left[\left(f(v) - f^{'}(v) \right) m(v) - \alpha \right] \right\} \tag{2.27}$$

If not specially declared in this section, A_w^* and $A_{w,x}^*$ are used to denote an optimal solution to Equation (2.26) and to Equation (2.27) with respect to tree T_w and tree $T_{w,x}$ respectively.

Based on theorems 1 and 2 proposed in Section 2.1, we can present the following dynamic programming-based algorithm for solving Equation (2.26).

Algorithm 6**:** Preliminary Algorithm

Step 1. Initialization:
$A_w^* = \phi$ and $G(T_w, A_w^*) = 0$;
Step 2. End condition
if $D(w) = \phi$ then return;
Step 3. Recursive procedure
for $v \in C(w)$ do
  if $D(v) = \phi$ then
    if $f(v)c(w,v) - \alpha > 0$ then
      $A_{w,v}^* = \{v\}$
    else
      $A_{w,v}^* = \phi$
  else
    for $x \in C(v)$ do
      if $\sum_{x \in C(v)} G(T_{w,x}, A_{w,x}^*) \geq G(T_v, A_v^*) + (f(v) - f^{'}(v))c(w,v) - \alpha$ then
        $A_{w,v}^* = \cup_{x \in C(v)} A_{w,x}^*$
      else
        $A_{w,v}^* = A_v^* \cup \{v\}$ (According to Theorem 2)
  $A_w^* = \cup_{v \in C(w)} A_{w,v}^*$ (According to Theorem 1)

Similarly, the time complexity of Algorithm 6 is given in the following corollary.

Corollary 4 *The time complexity of Algorithm* 6 *is* $O(n^2)$*, where* n *is the total number of nodes in the network.*

Now we begin to solve the problem of finding the optimal locations of placing k proxy servers in tree networks as described in Equation (2.22).

First, we give the following definition. The local access cost gain [12] at node v, denoted by $h(v)$, is defined as $h(v) = f(v)c(v, w)$.

Let $\alpha_{max} = \max_{v \in V}\{h(v)\}$. If $\alpha > \alpha_{max}$, then we can easily see that there is no feasible solutions to Equation (2.23); thus, the parameter α in Equation (2.23) should satisfy: $0 \leq \alpha \leq \alpha_{max}$. The crucial observation is that the optimal number of proxies to be placed is a monotonically decreasing function of α, that is, as α increases, this number will decrease monotonically. Therefore, we can determine the optimal locations for placing exactly k proxy servers among the nodes by tuning the parameter α. The relationship between the optimal number of proxies to be placed, denoted by k^*, and the parameter α can be seen in Figure 2.11.

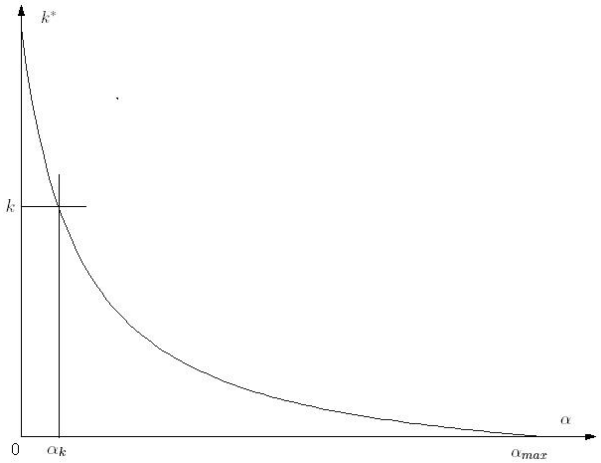

Figure 2.11: Relationship between k^* and α

Therefore, based on Algorithm 6, we can present an algorithm for placing k proxy servers among the nodes for tree networks as follows:

Algorithm 7: Proxy Placement for Tree Networks

Step 1. Initialization
$\alpha_{min} = 0$; $\alpha_{max} = \max_{v \in V}\{h(v)\}$, $k^* = |A_w^*|$.
Step 2. Recursive Procedure
while $k^* \neq k$ do
    $\alpha = (\alpha_{min} + \alpha_{max})/2$;
    Call Algorithm 6;
    $k^* = |A_w^*|$;
    if $k^* > k$ then
        $\alpha_{min} = \alpha$
    else
        $\alpha_{max} = \alpha$

We can see that Algorithm 7 converges to the optimal solution to Equation (2.22) quickly. The time complexity of Algorithm 7 is given in the following corollary.

[12] The local access cost gain is the access cost gain for placing only one proxy server among the nodes.

Corollary 5 *The time complexity of Algorithm 7 is $O(n^2)$, where n is the total number of nodes in the network.*

From Corollary 5, we can see that the time complexity of our algorithm is much better than the $O(n^3k^2)$ time complexity of a different dynamic programming algorithm for this problem proposed in [53]. Compared with the $O(nhk)$ time complexity of the algorithm proposed in [49], we can see that the two algorithms have their own strength, i.e., if $hk \geq n$, then our algorithm is a little better; otherwise, the algorithm in [49] is a little better.

2.2.3 Dynamic Programming-Based Solution for Autonomous Systems

In this section, we assume that the network topology for each AS is a tree. We denote the whole network by $T_{AS} = (V_{AS}, E_{AS})$, where V_{AS} is the set of the nodes and E_{AS} is the set of the links. We assume that there are m autonomous systems in the network, each of which is represented by tree $T_i, i = 1, 2, \cdots, m$. We denote the set of the replica servers and the content server as S_{AS}.

Based on Equation (2.21), the proxy placement problem for autonomous systems is defined as follows:

$$\max_{A_{AS} \subseteq V_{AS} - S_{AS}; |A_{AS}| = k} G(T_{AS}, A_{AS}) = \max_{A_{AS} \subseteq V_{AS} - S_{AS}; |A_{AS}| = k} \left\{ \sum_{v \in A_{AS}} \Big(f(v) - f^{'}(v) \Big) m(v) \right\} \tag{2.28}$$

From Equation (2.28), we can see that this problem degenerates to the problem addressed in Section 2.2.2 when $m = 1$. In this section, we also use $\widetilde{G}(T_{AS}, k)$ to denote the overall maximum access cost gain for placing k proxy servers in T_{AS} for convenience, i.e. $\widetilde{G}(T_{AS}, k) = G(T_{AS}, P^*_{AS})$, where P^*_{AS} is an optimal solution to Equation (2.28) and $k = |P^*_{AS}|$.

Now we apply the following idea to solve this problem, which is similar to that presented in [44]. We first divide T_{AS} into two parts: $\cup_{i=1}^{m-1} T_i$ and T_m, and consider the problem of placing k_m proxy servers in the first part and $k - k_m$ proxy servers in the second part, where $0 \leq k_m \leq k$. Then, we divide $\cup_{i=1}^{m-1} T_i$ into two parts: $\cup_{i=1}^{m-2} T_i$ and T_{m-1}, and consider the problem of placing k_{m-1} proxy servers in the first part and $k_m - k_{m-1}$ proxy servers in the second part, where $0 \leq k_{m-1} \leq k_m$. We repeat this process until there is only one tree left. The recursive process is given in the following theorem.

Theorem 10

$$\widetilde{G}(T_{+i}, k) = \begin{cases} \widetilde{G}(T_1, k) & \text{if } T_{+i} = T_1 \\ \max\limits_{0 \leq k^{'} \leq k} \{\widetilde{G}(T_{+(i-1)}, k^{'}) + \widetilde{G}(T_i, k - k^{'})\} & \text{if } T_{+i} \neq T_1 \end{cases}$$

where $T_{+i} = \cup_{j=1}^{i} T_j$.

The proof of Theorem 10 can be found in Section 2.1. Now we can present the algorithm for proxy placement for autonomous systems as follows:

Algorithm 8: Proxy Placement for Autonomous Systems

Main Procedure
for $i = 1$ to m do (Initialization)
  for $j = 0$ to $k - m + i$ do
    $\widetilde{G}(T_{+i}, j) = -1$; (The cost gain for placing j proxy servers in T_{+i})
    $P(T_i, j) = \phi$ (The optimal solution for placing j proxy servers in T_i)
Call $Placement(T_{+m}, k)$; (Calling procedure *Placement* recursively)

Procedure $Placement(T_{+i}, l)$
if $\widetilde{G}(T_{+i}, l) \geq 0$ then
  Return $\widetilde{G}(T_{+i}, l)$; ($\widetilde{G}(T_{+i}, l)$ computed)
if $i = 1$ then
  Return $\widetilde{G}(T_1, l)$; (Call Algorithm 7)
$TG = 0$;
for $l' = (i-1)$ to $l - 1$ do (Finding the optimal placement in T_i)
  TG=$Placement(T_{+(i-1)}, l') + \widetilde{G}(T_i, l - l')$; (According to Theorem 8)
    =$Placement(T_{+(i-1)}, l') + G(T_i, A_i^*)$;
  if $\widetilde{G}(T_{+i}, l) > TG$ then
    $\widetilde{G}(T_{+i}, l) = TG$
    $P(T_i, l - l') = A_i^*$

The time complexity of Algorithm 8 is given in the following corollary.

Corollary 6 *The time complexity of Algorithm* 8 *is* $O(n^2k)$*, where* n *is the total number of nodes in the network and* k *is the number of proxy servers to be placed in the network.*

From Corollary 6, we can see that the time complexity of our algorithm is much better than the $O(n^3k^3)$ time complexity of a different dynamic programming algorithm for this problem proposed in [44].

2.2.4 Simulation Model

We have performed extensive simulation experiments for comparing the results of our model with those of the random placement model. As it is difficult to find true trace data in the open literature to simulate our model, we generated the simulation model from empirical results presented in [7, 11, 14].

The network topology is randomly generated by the Tier program [14] and consists of numerous WAN (Wide Area Network) nodes and MAN (Metropolitan Area Network) nodes. We have conducted experiments for a lot of topologies with different parameters and found that the performance of our model was insensitive to the topology changes. Here, we just list only the experimental results for one topology due to space limitations. Table 2.2 shows the parameters and their values used in our experiments, where $U(x, y)$ denotes the uniform distribution between x and y.

The WAN is viewed as the backbone network to which no content servers or clients are attached. Each MAN node is assumed to connect to the content server. Each MAN and WAN node is associated with a proxy server. In our experiments, the client randomly generates the requests and the average request rate of each node follows the distribution of $U(1, 9)$. The cost for each link is calculated by the access latency. For simplicity,

Table 2.2: Parameters of Our Experiments

Parameter	Value
Total Number of Nodes	1000
Ratio of Number of WAN nodes to that of MAN Nodes	1 : 1
Number of Objects	1000
The Average Object Size	$30K$
Delay of WAN Links	0.45 second
Delay of MAN Links	0.06 second
Average Request Rate Per Node	$U(1, 9)$ requests per node

the delay caused by sending the request and the relevant response for that request is proportional to the size of the requested object. Here, we consider the average object sizes for calculating all delays, including the propagation delay, the transmission delay, and the searching delay. The cost function is taken to be the delay of the link, which means that the cost in our model is interpreted as the access latency in our simulation.

2.2.5 Experimental Results

In our experiments, we denote the results for random proxy placement model by $RPPM$, and the results for the heuristic model ($KMPC$) proposed in [63] by $KMPC$, and the results for dynamic programming-based solution proposed in this section by $DPBS$.

Figure 2.12 shows the results of the average access latency and the average response ratio as functions on numbers of proxy servers. As we have known, the lower the average access latency and the average response ratio, the better the performance. We can easily see that the performance results for both models improve as the number of proxy servers increases. We can also see that our model for tree topology can improve the average access latency and the average response ratio compared to the other models, since our model determines the optimal locations in the whole tree.

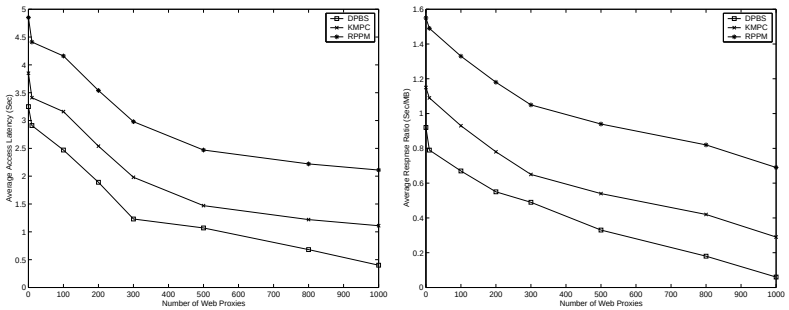

Figure 2.12: Experiment for Average Access Latency and Average Response Ratio

Figure 2.13 plots the results of the object hit ratio and the byte hit ratio as functions on numbers of proxy servers for different models. By computing the optimal locations for the tree topology, we can see that the results for our model outperforms those of the other models greatly. The object hit ratio and the byte hit ratio for each model steadily improve as the number of proxy servers increases, which conforms to the fact that more

requests can be satisfied by closer proxies as the number of proxy servers becomes greater.

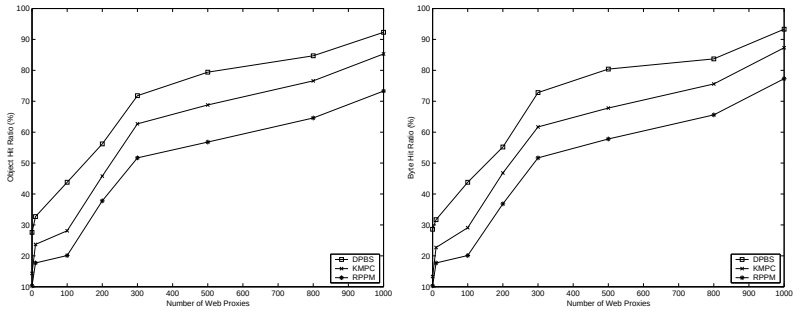

Figure 2.13: Experiment for Object Hit Ratio and Byte Hit Ratio

Figure 2.14 shows the results of the server load as a function on the number of proxy servers. It can be seen that the average server load for our model is lower than that of the other model. We can also see that the average server load for each model decreases as the number of proxy servers increases.

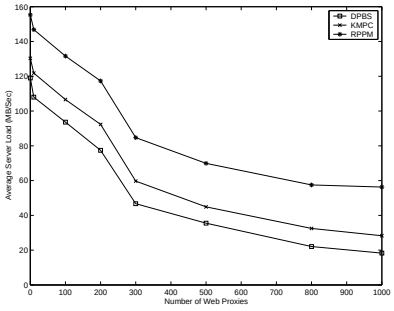

Figure 2.14: Experiment for Average Server Load

2.3 Chapter Summarization

In this chapter, we first addressed the problem of coordinated en-route web object caching in Section 2.1, and then studied the problem of proxy placement for coordinated en-route web caching in Section 2.2. We proposed several solutions for solving the above problems for several cases and conducted extensive simulation experiments to evaluate our solutions. The contents included in this chapter can be found in [55–57].

Chapter 3

Coordinated En-Route Transcoding Proxy Caching

This chapter studies transcoding proxy caching (i.e., web caching in transcoding proxies), which typically connects two heterogeneous networks. A WAP proxy, or gateway, is such a transcoding proxy that connects the wireless network and the Internet. Figure 3.1 shows a simple example consisting of content servers, transcoding proxies, and various types of client devices. The transcoding proxy is connected to the content servers via the IP network, while the clients are connected to the transcoding proxy via different kinds of networks, such as LANs (Local Area Network), WANs (Wide Area Network), wireless networks, etc.

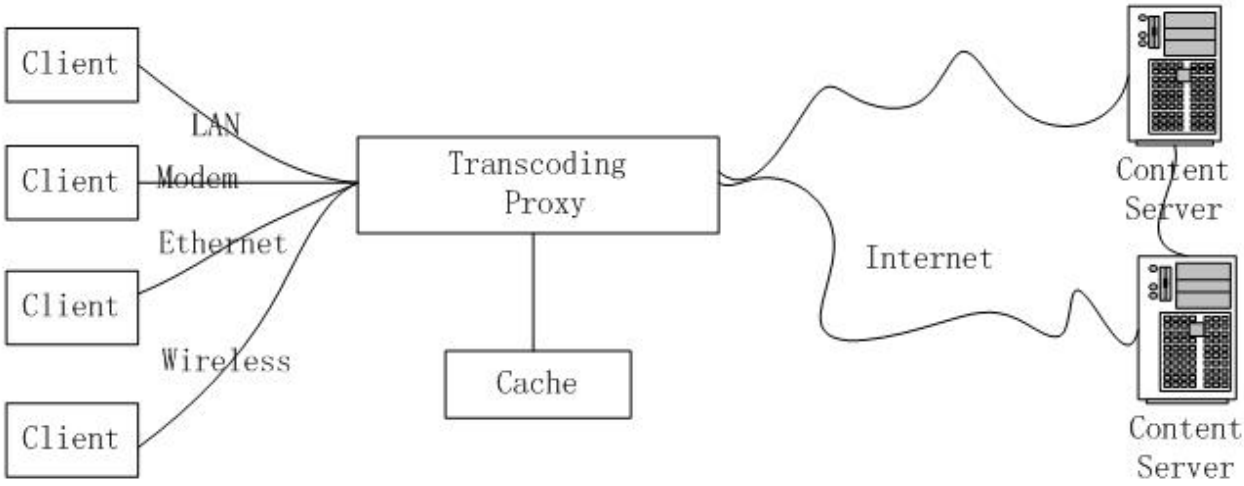

Figure 3.1: System Architecture

All the media objects which are maintained by the content servers, such as audio and video, are referred to as *full object versions*. The transcoding proxy is located closer to its clients. A full object version has numerous transcoded versions such that different clients' capabilities can be accommodated.

The request handling procedure works as follows. Clients' requests for media objects are directed to the transcoding proxy. The requests consist of the name of the media objects and the capability of the client device. When a user's request arrives at the

transcoding proxy, the proxy searches its cache for the appropriate media object version. One of the following situations could occur:

- *Version Hit*: The requests are satisfied by the exact versions of the media objects cached. No transcoding is necessary.
- *Content Hit*: The requests are satisfied by the more detailed versions of the media objects cached. Additional transcoding is necessary.
- *Cache Miss*: The requests can not be satisfied by the exact versions of the objects or the more detailed versions of the objects cached. The full object version needs to be retrieved from the server and transcoding is also necessary.

3.1 Coordinated En-Route Multimedia Object Caching

3.1.1 Problem Formulation

The network we use in this section is modelled as a tree $T = (V, E)$, where $V = (v_1, v_2, \cdots, v_n)$ is the set of nodes, and E is the set of network links. Without loss of generality, we assume that there is only one content server at the root by which all the objects are maintained. In our analysis, we also assume that each node is associated with an en-route cache. A client's request for a multimedia object goes along the path from the client to the server until it is satisfied by the first node on the path whose cache stores a more detailed version of the requested object. After transcoding if necessary, the proper version will be sent back to the client along the same path. We also assume in this section that the routing path is symmetric. For the asymmetric case, we can consider a subset of V by excluding those noses which are not on both upstream and downstream paths. Such a simplification is validated in [90]. Figure 3.2 shows an example of such a tree topology. Here, v_0 is the node associated with the content server, and the other nodes are associated with en-route caches.

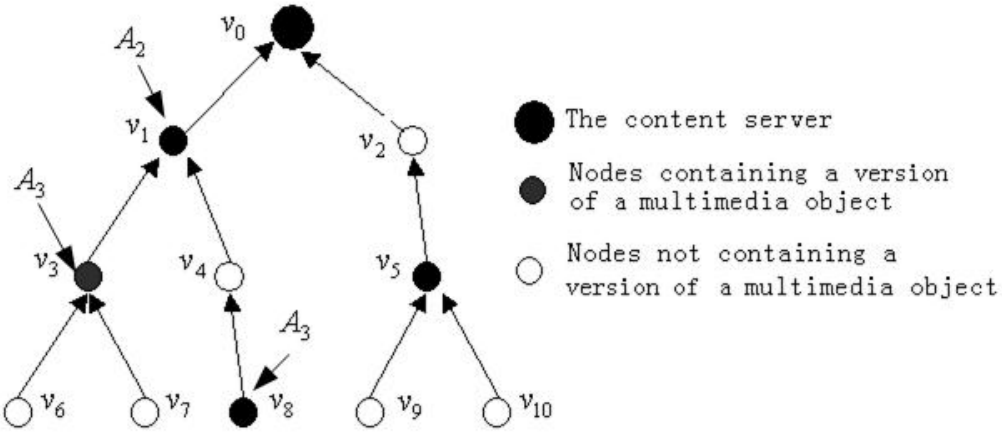

Figure 3.2: Coordinated En-Route Multimedia Object Caching in Transcoding Proxies

We denote the set of all nodes that are the children of node v as $C(v)$, and the set of all nodes that are the descendants of node v as $D(v)$. For instance, in Figure 3.2,

$C(v_1) = \{v_3, v_4\}$, and $D(v_1) = \{v_3, v_4, v_6, v_7, v_8\}$. Let $A = (A_1, A_2, \cdots, A_m)$ denote the set of all versions of a multimedia object [1], and e_{A_j} denote the size of A_j. We assume that the access frequencies for A_j from v_i, denoted by f_{A_j,v_i}, are independent. The cost of transmitting a multimedia object between v_i and v_j is denoted by c_{v_i,v_j}. In this section, we assume that c_{v_i,v_j} is the same for different versions on the same network link. We shall consider the general case in which c_{v_i,v_j} is different for different versions in our future work. The cost in our analysis is calculated from a general point of view. It can be different performance measures such as delay, bandwidth requirement, and access latency, or a combination of these measures. If a request goes through multiple network links, the cost is the sum of the cost on all these links.

The relationship among different versions of a multimedia objects can be expressed by a weighted transcoding graph [23]. An example of such a weighted transcoding graph is shown in Figure 3.3, where the original version A_1 can be transcoded to each of the less detailed versions A_2, A_3, A_4, and A_5. It should be noted that not every A_i can be transcoded to A_j since it is possible that A_i does not contain enough content information for the transcoding from A_i to A_j. In our example, transcoding can not be executed between A_4 and A_5 due to insufficient content information. The transcoding cost of a multimedia object from A_i to A_j is denoted by $w(A_i, A_j)$. The number beside each edge in Figure 3.3 is the transcoding cost from one version to another. For example, $w(A_1, A_2) = 12$, and $w(A_3, A_4) = 10$. $\Phi(A_i)$ is the set of all the versions that can be transcoded from A_i, including A_i. For example, $\Phi(A_1) = \{A_1, A_2, A_3, A_4, A_5\}$, $\Phi(A_2) = \{A_2, A_4, A_5\}$, and $\Phi(A_4) = \{A_4\}$. In this section, we assume that $w(A_i, A_j)$ is independent on other factors, such as the CPU and the workload of the computer with an en-route cache. We shall further consider this case in our future work.

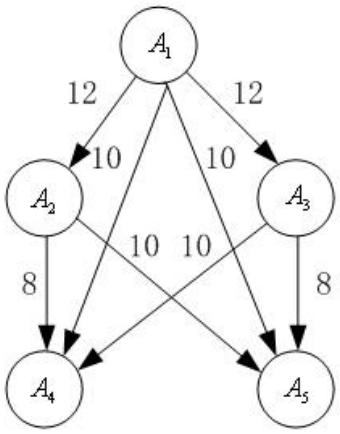

Figure 3.3: An Example of A Weighted Transcoding Graph

We use $v^+(A_i)$ to denote the nearest higher level node of v that stores a more detailed version than A_i (including A_i), $B_v^-(A_i)$ to denote the set of the nearest lower level node of v that stores a less detailed version than A_i (including A_i) in different branches from node v, and H_v to denote the version cached or to be cached at node v. For example, if a multimedia object has five versions as shown in Figure 3.3, and A_2 is cache at v_1, A_3

[1] $A_1, A_2, \cdots, A_m$ are all stored at the server, i.e., v_0 .

is cached at v_3 and v_8 (see Figure 3.2), then we have $v_6^+(A_3) = v_6^+(A_4) = v_6^+(A_5) = v_3$, $v_6^+(A_2) = v_1$, $v_6^+(A_1) = v_0$; $B_{v_1}^-(A_3) = B_{v_1}^-(A_4) = B_{v_1}^-(A_5) = \{v_3, v_8\}$ since A_3, A_4 and A_5 can all be transcoded from A_3, $B_{v_1}^-(A_1) = B_{v_1}^-(A_2) = \phi$ in that A_1 and A_2 cannot be transcoded from A_3. A list of symbols used in this section is given in Table 3.1.

Table 3.1: A List of the Notations

Notation	Decription
$C(v)$	set of all nodes that are the children of node v
$D(v)$	set of all nodes that are the descendants of node v
$A = (A_1, A_2, \cdots, A_m)$	set of versions of a multimedia object
e_{A_j}	the size of A_j
f_{A_j,v_i}	mean access frequency of A_j from v_i
c_{v_i,v_j}	cost of transmitting a multimedia object between v_i and v_j
$w(A_i, A_j)$	transcoding cost from A_i to A_j for a multimedia object
$\Phi(A_i)$	set of all versions that can be transcoded from A_i
$v^+(A_i)$	nearest higher level node of v that stores a more detailed version than A_i
$B_v^-(A_i)$	set of the nearest lower level node of v that stores a less detailed version than A_i in different branches
H_v	version cached or to be cached at node v

The problem of multimedia object caching is similar to that of web object caching as discussed in Chapter 2. The problem of coordinated en-route multimedia object caching in transcoding proxies is further complicated due to the relationship among different versions of the same multimedia object. We begin with computing the cost saving and the cost loss of caching a multimedia object at a single node. We first define the miss penalty, which will be used to define the cost saving and the cost loss later. Let $m(A_i, v_j)$ be the miss penalty of version A_i with respect to node v_j, which is defined as the additional cost of accessing A_i if the version cached at node v_j, i.e., H_{v_j}, is removed. In our model, $m(A_i, v_j)$ is given by the following definition.

Definition 1 *$m(A_i, v_j)$ is a function for calculating the miss penalty of A_i if H_{v_j} is removed from node v_j, which is defined as*

$$m(A_i, v_j) = c_{v_j, v_j^+(A_i)} + w(H_{v_j^+(A_i), A_i}) - w(H_{v_j}, A_i)$$

It is easy to see that $c_{v_j, v_j^+(A_i)}$ is the additional access cost, $w(H_{v_j^+(A_i)}, A_i)$ is the new transcoding cost, and $w(H_{v_j}, A_i)$ is the original transcoding cost. For example, if a multimedia object has five versions as shown in Figure 3.3, and A_2 is cache at v_1, A_3 is cached at v_3 and v_8 (see Figure 3.2), then we have $m(A_5, v_3) = c(v_3, v_1) + w(A_2, A_5) - w(A_3, A_5)$, $m(A_4, v_3) = c(v_3, v_1) + w(A_2, A_4) - w(A_3, A_4)$, and $m(A_3, v_3) = c(v_3, v_0)$.

Obviously, the cost saving of caching H_v at v is defined as follows.

Definition 2 *$s(H_v)$, a function for calculating the cost saving of caching H_v at v, is defined as*

$$s(H_v) = \sum_{A_x \in \Phi(H_v)} f_{A_x,v} \cdot m(A_x, v)$$

Second, we consider the cost loss of caching a multimedia object at a node, which is calculated in the same way as introduced in Chapter 2.

Taking into account the cache dependencies, the cost saving of caching H_v at v should be calculated as follows.

$$s(H_v) = \sum_{A_x \in \Phi(H_v)} \left(f_{A_x,v} - \sum_{z \in B_v^-(A_x)} f_{A_x,z} \right) m(A_x, v)$$

Now we give a simple example to explain how to calculate the cost saving by considering the cache dependencies. For example, if a multimedia object has five versions as shown in Figure 3.3, and A_2 is cache at v_1, A_3 is cached at v_3 and v_8 (see Figure 3.2), then the cost saving of caching A_2 at v_1 is calculated as $f_{A_2,v_1} m_{A_2,v_1} + \sum_{i=4}^{5} (f_{A_i,v_1} - (f_{A_i,v_3} + f_{A_i,v_8})) m_{A_i,v_1}$, since $\Phi(A_2) = \{A_2, A_4, A_5\}$.

The cost gain, defined as the difference between the cost saving and the cost loss, is the reduction of access cost in the network. Based on the above analysis, the total cost gain of caching multiple versions of a multimedia object is defined as follows.

Definition 3 *$G(T, P)$, the total cost gain of caching multiple versions of a multimedia object at nodes in $P \subseteq V$ is defined as*

$$G(T, P) = \sum_{v \in P} s(H_v) - l(H_v) = \sum_{v \in P} \sum_{A_x \in \Phi(H_v)} \left(f_{A_x,v} - \sum_{z \in B_v^-(A_x)} f_{A_x,z} \right) m(A_x, v) - l(H_v)$$

Obviously, our objective is to find $P^* = \{P_1, P_2, \cdots, P_{k^*}\} \subseteq V$ such that $G(T, P^*) = \max_P \{G(T, P)\}$, where $H^*_{P_i} \in A$ $(1 \leq i \leq k^*)$ is the proper version that should be cached at node P_i, and the caching decisions are made from the optimal solution, i.e., $H^*_{P_i}$.

3.1.2 Dynamic Programming-Based Solution for Linear Networks

Now we begin to present an optimal solution for coordinated en-route multimedia object caching for linear networks. Consider the snapshot when a request for a media object is being served (see Figure 3.4). Let Node 0 be the content server or the higher level node satisfying the object request, node n be the client issuing the request, and node $1, 2, \cdots, n-1$ the nodes on the path from 0 to n.

Based on Definition 3, this problem can be simplified by the follwoing definition.

Definition 4 *Let $v_1, v_2, \cdots, v_k$ be a set of k nodes such that $1 \leq v_1 \leq v_2 \leq \cdots \leq v_k \leq n$. $F(n : v_1, v_2, \cdots, v_k)$, which is the aggregate profit of caching multiple versions of a media*

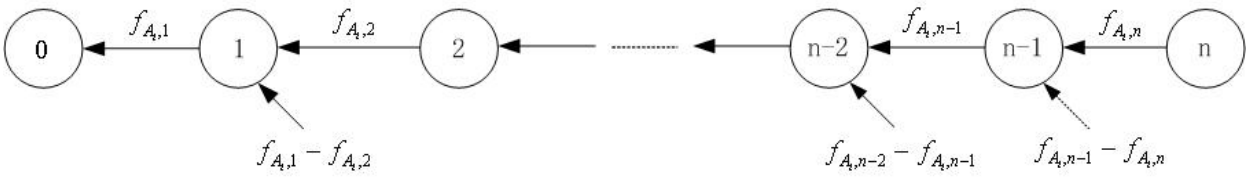

Figure 3.4: Coordinated En-Route Multimedia Object Caching for Linear Networks

object at $v_1, v_2, \cdots, v_k$*, is defined as*

$$F(n : v_1, v_2, \cdots, v_k) = \sum_{j=1}^{k} (S(H_{v_j}) - L(H_{v_j})) \tag{3.1}$$

If $k = 0$*, then we define* $F(n : \phi) = 0$*. Finding* k *and* $v_1, v_2, \cdots, v_k$ *that maximizes* $F(n : v_1, v_2, \cdots, v_k)$ *is referred to as the* n*-optimization problem [90].*

Our objective is to compute the locations for placing multiple versions of a media object in a subset of nodes $\{v_1, v_2, \cdots, v_k\}$ that maximizes the aggregate profit as defined in Equation (3.1).

In this following, we develop a dynamic programming-based algorithm which is inspired by [53] to solve the problem formulated in Section 3.1.1. The following theorem shows that an optimal solution to Equation (3.1) must contain optimal solutions to some subproblems.

Theorem 11 *Suppose that* $\{v_1, v_2, \cdots, v_I\}$ *is an optimal solution to the* n*-optimization problem as defined in Equation (3.1) and* $\{u_1, u_2, \cdots, u_l\}$ *is an optimal solution to the* $(v_I - 1)$*-optimization problem, then* $\{u_1, u_2, \cdots, u_l, v_I\}$ *is also an optimal solution to the* n*-optimization problem.*

Proof By definition, it is obvious that the following inequality is correct.

$$F(v_I - 1 : u_1, u_2, \cdots, u_l) \geq F(v_I - 1 : v_1, v_2, \cdots, v_{I-1})$$

Therefore, we have

$$\begin{aligned}
&F(n : u_1, u_2, \cdots, u_l, v_I) \\
&\quad = (S(H_{u_1}) - L(H_{u_1})) + \cdots + (S(H_{u_l}) - L(H_{u_l})) + (S(H_{v_I}) - L(H_{v_I})) \\
&\quad = F(v_I - 1 : u_1, u_2, \cdots, u_l) + (S(H_{v_I}) - L(H_{v_I})) \\
&\quad \geq F(v_I - 1 : v_1, v_2, \cdots, v_{I-1}) + (S(H_{v_I}) - L(H_{v_I})) \\
&\quad = (S(H_{v_1}) - L(H_{v_1})) + \cdots + (S(H_{v_{I-1}}) - L(H_{v_{I-1}})) + (S(H_{v_I}) - L(H_{v_I})) \\
&\quad = F(n : v_1, v_2, \cdots, v_{I-1}, v_I)
\end{aligned}$$

On the other hand, since $\{v_1, v_2, \cdots, v_I\}$ is an optimal solution to the n-optimization problem, we have

$$F(n : u_1, u_2, \cdots, u_l, v_I) \leq F(n : v_1, v_2, \cdots, v_{I-1}, v_I).$$

So we have

$$F(n : u_1, u_2, \cdots, u_l, v_I) = F(n : v_1, v_2, \cdots, v_{I-1}, v_I).$$

Hence, the theorem is proven. □

Before presenting the dynamic programming-based algorithm, we give the following definition.

Definition 5 *Define F_n^* to be the maximum aggregate profit of $F(n : v_1, v_2, \cdots, v_k)$ obtained by solving the n-optimization problem and I_n the maximum index in the optimal solution. If the optimal solution is an empty set, define $I_n = -1$.*

Obviously, we have $I_0 = -1$ and $F_0^* = 0$. From Theorem 11, we know that if $I_r > 0$,

$$F_{I_r} = F_{I_r - 1} + (S(H_{v_{I_r}}) - L(H_{v_{I_r}}))$$

Therefore, we can check all possible locations of I_r $(0 \leq r \leq n)$ and select the one that maximizes $F(r : v_1, v_2, \cdots, v_k)$. So we have

$$\begin{cases} F_0^* = 0 \\ F_r^* = \max\limits_{1 \leq v_i \leq r} \left\{0, F_{v_i - 1}^* + (S(H_{v_i}) - L(H_{v_i}))\right\} \end{cases}$$

and

$$\begin{cases} I_0 = -1 \\ I_r = \begin{cases} -1 \;\; if \;\; F_r^* = 0 \\ v \quad if \;\; F_r^* = F_{v-1}^* + (S(H_v) - L(H_v)) \end{cases} \end{cases}$$

The original problem can be solved using a dynamic programming-based algorithm with the recurrences above. Theorem 11 ensures the correctness.

In this following, we present an analysis of the algorithm above. The following theorem describes an important property of the algorithm.

Theorem 12 *Suppose that $\{v_1, v_2, \cdots, v_I\}$ is an optimal solution to the n-optimization problem, then we have*

$$\sum_{A_x \in D(B_{v_i})} f_{A_x, v_i} \cdot m(A_x, v_i) - l(A_x, v_i) \geq 0 \quad \forall \;\; 1 \leq i \leq k$$

Proof Suppose that there exists r such that

$$\sum_{A_x \in D(B_{v_r})} f_{A_x, v_r} \cdot m(A_x, v_r) - l(A_x, v_r) < 0$$

then we have

$$\begin{aligned} & F(n : v_1, v_2, \cdots, v_I) \\ & \quad = F(v_r - 1 : v_1, v_2, \cdots, v_{r-1}) \\ & \qquad + \sum_{A_x \in D(B_{v_r})} \left(f_{A_x, v_r} - f_{A_x, v_r^-(A_x)}\right) \cdot m(A_x, v_r) - l(A_x, v_r) \\ & \qquad + F(n - v_r : v_{r+1}, v_{r+2}, \cdots, v_I) \\ & \quad \leq F(v_r - 1 : v_1, v_2, \cdots, v_{r-1}) \\ & \qquad + \sum_{A_x \in D(B_{v_r})} f_{A_x, v_r} \cdot \cdot m(A_x, v_r) - l(A_x, v_r) \\ & \qquad + F(n - v_r : v_{r+1}, v_{r+2}, \cdots, v_I) \\ & \quad < F(v_r - 1 : v_1, v_2, \cdots, v_{r-1}) + F(n - v_r : v_{r+1}, v_{r+2}, \cdots, v_I) \\ & \quad \leq F(n : v_1, \cdots, v_{r-1}, v_{r+1}, \cdots, v_I) \end{aligned}$$

which contradicts the fact that $\{v_1, v_2, \cdots, v_I\}$ is an optimal solution to the n-optimization problem. Hence, the theorem is proven. □

Theorem 12 shows that we should only consider the nodes where the local profit is beneficial [2]. It can be easily shown that the time complexity of the algorithm proposed in Section 3.1.2 is $O(n^2m)$, where n is the number of nodes and m is the number of all the versions of a media object. From Theorem 12, we can also easily see that the time complexity of the algorithm is greatly less than $O(n^2m)$ since there must be many nodes whose local profit are not beneficial; therefore, the cost of computing the optimal solution is low.

3.1.3 Dynamic Programming-Based Solution for Tree Networks

Now we begin to solve the problem of coordinated en-route multimedia object caching for tree networks. To formally define our problem, we use $T_r = (V, E)$ to identify a tree whose root is r. Based on Definition 3, the problem of coordinated en-route multimedia object caching for tree T_r is defined as an optimization problem as follows:

$$\max_{P_r} G(T_r, P_r) = \max_{P_r} \sum_{v \in P_r} (s(H_v) - l(H_v)) \tag{3.2}$$

where $P_r \subseteq D(r)$.

Before presenting a solution to Problem (3.2), we define the following optimization problem for tree $T_{r,w}$.

$$\max_{P_{r,w}} G(T_{r,w}, P_{r,w}) = \max_{P_{r,w}} \sum_{v \in P_{r,w}} (s(H_v) - l(H_v)) \tag{3.3}$$

where $P_{r,w} \subseteq D(w) \cup \{w\}$. Here, tree $T_{r,w}$ is a subtree of T_r whose node set is $D_w \cup \{w\}$, where $w \in D(r)$.

Now we start to present a solution to Problem (3.2). First, we give a theorem that indicates an important property between the optimal solutions for tree T_r and T_{r,r_i}, where $r_i \in C(r)$.

Theorem 13 *For tree T_r, if $C(r) = \{r_1, r_2, \cdots, r_s\}$, then we have*

$$P_r^* = \cup_{i=1}^{s} P_{r,r_i}^*$$

where P_r^* is an optimal solution to Problem (3.2) with respect to tree T_r, and P_{r,r_i}^* is an optimal solution to Problem (3.3) with respect to tree T_{r,r_i}, $i = 1, 2, \cdots, s$.

Theorem 13 is the same as Theorem 1 in Chapter 1. For easy understanding, we redescribe it there. Note that Theorem 13 shows that computing an optimal solution to Problem (3.2) can be decomposed into computing the optimal solutions for subtrees T_{r,r_i}, where $C(r) = \{r_1, r_2, \cdots, r_s\}$. Therefore, we can solve Problem (3.2) if and only if we can solve Problem (3.3).

In the following, we present another theorem that describes an important property of the solution to Problem (3.3), by which an optimal solution to Problem (3.3) can be obtained.

[2]Here, local profit refers to the profit of caching only one version of a media among the caches.

Theorem 14 *For tree $T_{r,w}$, if $C(w) = \{w_1, w_2, \cdots, w_t\}$, then we have*

$$P^*_{r,w} = \begin{cases} \cup_{i=1}^{t} P^*_{r,w_i} & G(T_{r,w}, \cup_{i=1}^{t} P^*_{r,w_i}) \geq G(T_{r,w}, P^*_w \cup \{w\}) \\ P^*_w \cup \{w\} & G(T_{r,w}, \cup_{i=1}^{t} P^*_{r,w_i}) < G(T_{r,w}, P^*_w \cup \{w\}) \end{cases}$$

where $P^*_{r,w}$ is an optimal solution to Problem (3.3) with respect to tree $T_{r,w}$, P^*_w is an optimal solution to Problem (3.2) with respect to tree T_w, and P^*_{r,w_i} is an optimal solution to Problem (3.3) with respect to tree T_{r,w_i}, $i = 1, 2, \cdots, t$.

The proof of Theorem 14 is similar to that of Theorem 2 in Chapter 1. Theorem 14 shows that an optimal solution to Problem (3.3) is decided according to the cost gain of storing a version at node w or not.

Based on theorems 13 and 14, the original problem, i.e. Problem (3.2), can be solved using dynamic programming with the following recurrences.

- Suppose that $v \in V$ and $C(v) = \{v_1, v_2, \cdots, v_{t_1}\}$. If $t_1 = 0$, then $P^*_v = \phi$; otherwise, $P^*_v = \cup_{i=1}^{t_1} P^*_{v,v_i}$.
- Suppose that $u \in D(v)$ and $C(u) = \{u_1, u_2, \cdots, u_{t_2}\}$. If $t_2 = 0$, then

$$P^*_{v,u} = \begin{cases} \{u\} & \sum_{A_x \in \Phi(H_u)} f_{A_x,u}[c(v,u) + w(H_u, A_x)] - l(H_u) \geq 0 \\ \{\phi\} & \text{Otherwise} \end{cases}$$

  Otherwise,

$$P^*_{v,u} = \begin{cases} \cup_{i=1}^{t_2} P^*_{v,u_i} & G(T_{v,u}, \cup_{i=1}^{t_2} P^*_{v,u_i}) - G(T_{v,u}, P^*_u \cup \{u\}) \geq 0 \\ P^*_u \cup \{u\} & \text{Otherwise} \end{cases}$$

With the same analysis presented in Chapter 1, we can obtain the following two theorems for describing some important properties of the solution proposed above.

Theorem 15 *If P^*_r is an optimal solution to Problem (3.2) with respect to tree T_r, then we have $s(H_v) - l(H_v) \geq 0, \ \forall\, v \in P^*_r$.*

Proof Suppose that there exists $u \in P^*_r$ that satisfies $s(H_u) - l(H_u) < 0$, then we have

$$\begin{aligned} & G(T_r, P^*_r - \{u\}) \\ &= \sum_{v \in P^*_r - (D(u) \cup \{u\})} (s(H_v) - l(H_v)) + \sum_{v \in P^*_r \cap D(u)} (s(H_v) - l(H_v)) \\ &> \sum_{v \in P^*_r - (D(u) \cup \{u\})} (s(H_v) - l(H_v)) + (s(H_u) - l(H_u)) + \sum_{v \in P^*_r \cap D(u)} (s(H_v) - l(H_v)) \\ &\geq \sum_{v \in P^*_r} \left[\sum_{A_x \in \Phi(H_v)} \left(f_{A_x,v} - \sum_{z \in B^-_v(A_x)} f_{A_x,z} \right) m(A_x, v) - l(H_v) \right] \text{[3]} \\ &= G(T_r, P^*_r) \end{aligned}$$

which contradicts the fact that P^*_r is an optimal solution to Problem (3.2) with respect to tree T_r. Hence, the theorem is proven. □

[3]This is because $f_{A_x,z}$ becomes larger for the nodes of upstream of node u and $m(A_x, v)$ smaller for the nodes of downstream of node u when a version is placed at node u.

Theorem 15 shows an important property of the optimal solution to Problem (3.2). All the nodes in the optimal solution must satisfy the condition above; otherwise, it can be removed from the optimal solution.

Theorem 16 *If P_r^* is an optimal solution to Problem (3.2) with respect to tree T_r, then we have*

$$\sum_{A_x \in \Phi(H_v)} f_{A_x,v} \cdot (c_{v,r} - w(H_v, A_x)) - l(H_v) \geq 0 \quad \forall \ v \in P_r^*$$

The proof of this theorem can be similarly obtained from that of Theorem 3 given in Chapter 1. Theorem 16 shows that we should only consider the nodes where the local cost gain is beneficial, i.e. the cost saving outweighs the cost loss with respect to that single node. It can be shown that the complexity of this dynamic programming-based solution is $O(n^3m^2)$, where n is the total number of nodes in the network and m is the number of the versions that a multimedia object owns. In practical network, there must be a number of nodes whose local cost gain is not beneficial; therefore, the time complexity should be less than $O(n^3m^2)$.

3.1.4 Coordinated Caching Scheme

Based on the previous analysis, we present the following coordinated web caching scheme. In our scheme, every cache maintains some information about the objects in the form of object descriptors. An object descriptor contains information that includes the object size, the access frequencies for all versions of the media object, and the cost losses of the media object with respect to the associated node. When a request is issued from node n for a version of a media object, each node v on the path between node 0 and n piggybacks the corresponding information f_{A_i,v_j}, $c_{v_i,v_j}, l(A_j, v_i)(i = 1, 2, \cdots, m; j = 1, 2, \cdots, n)$, as it passes the node. When the request arrives at the content server, the optimal locations for placing multiple versions of a media object are calculated using the dynamic programming-based algorithm according to the information accumulated, then the content server sends the decisions back with copies of that media object. If a version should be cached at node v (transcoding will be executed if necessary), then node v executes the greedy heuristic algorithm (e.g. *LRU*) to decide which objects should be removed and update its cache information accordingly.

Since the cache contents change over time, the access frequency and the cost loss of an object with respect to a node must be refreshed from time to time. The access frequency can be estimated based on recent request history, which is locally available (e.g. by using a "sliding window" technique [88]). The cost loss is updated by the response messages. Specifically, a variable with an initial value of zero is attached to each object. At each intermediate node along the way, the variable is increased by the cost of the last link the object has just traversed. The value is then used to update the cost loss of the object maintained by the associated cache. If the object is inserted into the cache, the node resets the value to zero before forwarding the object downstream. In this way, the updated cost loss is disseminated to all the caches on the way.

3.1.5 Simulation Model

In this section, the simulation model used for performance evaluation is described. We have performed extensive simulation experiments to compare the results of our model (proposed in Section 3.1.3) with those of existing caching models. The network in our simulation consists of numerous nodes and content servers. The system configuration is outlined in section 3.1.5, and four existing caching models used for the purpose of comparison are introduced in Section 3.1.5.

System Configuration

To the best of our knowledge, it is difficult to find true trace data in the open literature to simulate our model. Therefore, we generated the simulation model from the empirical results presented in [1, 7, 11, 14, 23, 45, 53, 90].

The network topology was randomly generated by the Tier program [14]. Experiments for many topologies with different parameters have been conducted and the relative performance of our model was found to be insensitive to topology changes. Here, only the experimental results for one topology was listed due to space limitations. The characteristics of this topology and the workload model are shown in Table 3.2, which are chosen from the open literature and are considered to be reasonable.

Table 3.2: Parameters Used in Simulation

Parameter	Value
Number of WAN Nodes	200
Number of MAN Nodes	200
Delay of WAN Links	Exponential Distribution $p(x) = \theta^{-1}e^{-x/\theta}$ $(\theta = 0.45\ Sec)$
Delay of MAN Links	Exponential Distribution $p(x) = \theta^{-1}e^{-x/\theta}$ $(\theta = 0.06\ Sec)$
Number of Servers	100
Number of Web Objects	1000 objects per srever
Web Object Size Distribution	Pareto Distribution $p(x) = \frac{ab^a}{a-1}$ $(a = 1.1, b = 8596)$
Web Object Access Frequency	Zipf-Like Distribution $\frac{1}{i^\alpha}$ $(i = 0.7)$
Relative Cache Size Per Node	4%
Average Request Rate Per Node	$U(1, 9)$ requests per second
Transcoding Rate	$20KB/Sec$

The WAN (Wide Area Network) is viewed as the backbone network to which no servers or clients are attached. Each MAN (Metropolitan Area Network) node is assumed to connect to a content server. Each MAN and WAN node is associated with an en-route cache. The number of objects generated is N and these N objects are divided

into two types: *text* and *multimedia*. Similar to the studies in [11, 15, 46, 88, 90], cache size is described as the total relative size of all objects available in the content server. In our experiments, the object sizes are assumed to follow a Pareto distribution and the average object size is $26KB$. We also assume that each multimedia object has five versions and that the transcoding graph is as shown in Figure 3.5. The sizes of each version are assumed to be 100 percent, 80 percent, 60 percent, 40 percent, and 20 percent of the original object size. The transcoding delay is determined as the quotient of the object size to the transcoding rate. In our experiments, the client at each MAN node randomly generates the requests, and the average request rate of each node follows the distribution of $U(1, 9)$, where $U(x, y)$ represents a uniform distribution between x and y. The access frequencies of both the content servers and the objects maintained by a given server follow a Zipf-like distribution [11, 69]. Specifically, the probability of a request for object O in server S is proportional to $1/(i^{\alpha} \cdot j^{\alpha})$, where S is the ith most popular server and O is the jth popular object in S. The delay of both MAN links and WAN links follows an exponential distribution, where the average delay for WAN links is 0.46 seconds and the average delay for WAN links is 0.06 seconds.

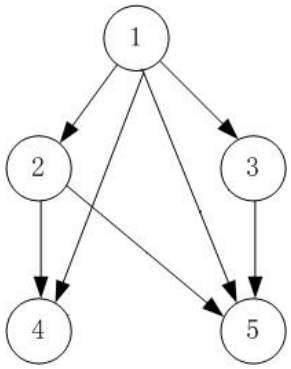

Figure 3.5: Transcoding Graph for Simulation

The cost for each link is calculated by the access delay. For simplicity, the delay caused by sending the request and the relevant response for that request is proportional to the size of the requested object. Here, we consider the average object sizes for calculating all delays, including the transmission delay, and transcoding delay. The cost function is taken to be the delay of the link, which means that the cost in our model is interpreted as the access latency in our simulation.

We apply a "sliding window" technique to estimate the access frequency to make our model less sensitive to transient workload [88]. Specifically, for each object O, $f(O, v)$ is calculated by $K/(t - t_K)$, where K is the number of accesses recorded, t is the current time, and t_K is the Kth most recently referenced time (the time of the oldest reference in the sliding window). K is set to 2 in the simulation. To reduce overhead, the access frequency is only updated when the object is referenced and at reasonably large intervals, e.g., several minutes, to reflect aging, which is also applied in [90].

Existing Caching Models

We also consider the following caching models in our simulation experiments.

- *LRU*: Least Recently Used (LRU) evicts the web object which is requested the least recently. The requested object is stored at each node through which the object passes. The cache purges one or more least recently requested objects to accommodate the new object if there is not enough room for it.

- *$LNC-R$ [86]*: Least Normalized Cost Replacement ($LNC-R$) is a model that approximates the optimal cache replacement model. It selects the least profitable documents for replacement. Similar to LRU, the requested object is cached by all nodes along the routing path.

- *AE [23]*: Aggregate Effect (AE) is a model that explores the aggregate effect of caching multiple versions of the same multimedia object in the cache. It formulates a generalized profit function to evaluate the aggregate profit from caching multiple versions of the same multimedia object. When the requested object passes through each node, the cache will determine which version of that object should be stored at that node according to the generalized profit.

- *$TCLT$ proposed in Section 3.1.2*: Multimedia object Caching for linear networks ($TCLT$).

- *$TCTN$ proposed in Section 3.1.3*: Multimedia object caching for tree networks ($TCTN$).

3.1.6 Performance Evaluation

In this section, we compare the performance results of our solution (proposed in Section 3.1.3) with those models introduced in Section 3.1.5 in terms of several performance metrics. We have The performance metrics we used in our simulation include delay-saving ratio (DSR), which is defined as the fraction of communication and server delays which is saved by satisfying the references from the cache instead of the server, average access latency (ASL), request response ratio (RRR), which is defined as the ratio of the access latency of the target object to its size, object hit ratio (OHR), which is defined as the ratio of the number of requests satisfied by the caches as a whole to the total number of requests, and highest server load (HSL), which is defined as the largest number of bytes served by the server per second. In the following figures, LRU, $LNC-R$, AE, and $TCLT$ denote the results for the four models introduced in Section 3.1.5, and $TCTN$ shows the results for the model of coordinated en-route multimedia object caching in transcoding proxies for tree networks proposed in Section 3.1.2. Table 3.3 lists the abbreviations used in this section.

Impact of Cache Size

In this experiment set, we compare the performance results of different models across a wide range of cache sizes, from 0.04 percent to 15.0 percent.

The first experiment investigates DSR as a function of the relative cache size at each node and Figure 3.6 shows the simulation results. As presented in Figure 3.6, we can see that our model outperforms the other models since our model considers coordinated en-route multimedia object caching in transcoding proxies by optimally determining the locations to place multiple versions of a multimedia object in a coordinated way, whereas

Table 3.3: Abbreviations Used in Performance Analysis

Meaning	Abbreviation	Decription
Performance Metric	DSR	Delay-Saving Ratio (%)
	ASL	Average Access Latency (Sec)
	RRR	Request Response Ratio (Sec/MB)
	OHR	Object Hit Ratio (%)
	HSL	Highest Server Load (MB/Sec)
Caching Model	$TCTN$	Multimedia Object Caching for Tree Networks
	$TCLT$	Multimedia Object Caching for Linear Networks
	AE	Standing for Aggregate Effect
	$LNC-R$	Least Normalized Cost Replacement
	LRU	Least Recently Used

existing models, including LRU, $LNC-R$, and AE, consider web caching in transcoding proxies either on a path or only at a single node. Specifically, the mean improvements of DSR over $TCLT$, AE, $LNC-R$, LRU are 4.3 percent, 21.3 percent, 29.7 percent, and 31.7 percent, respectively.

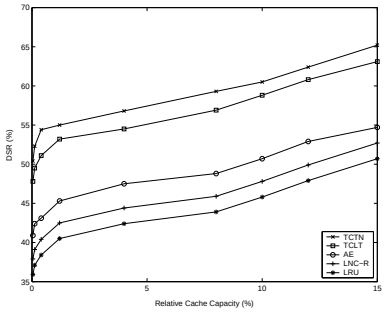

Figure 3.6: Experiment on DSR

Figure 3.7 shows the simulation results of ASL and RRR as a function of the relative cache size at each node. Clearly, the lower the ASL or the RRR, the better the performance. As we can see, all models provide steady performance improvement as the cache size increases. We can also see that $TCTN$ significantly improves both ASL and RRR compared to $TCLT$, AE, $LNC-R$ and LRU, since our model determines the locations for tree networks in an optimal and coordinated way, while the others place multiple versions of a multimedia object for linear topology or at each en-route cache. For ASL to achieve the same performance as $TCTN$, the other models need 2 to 12 times as much cache size.

Figure 3.8 shows the results of OHR as a function of the relative cache size for different models. By computing the optimal locations, we can see that the results for our model can greatly outperform those of the other models, especially for smaller cache sizes. We can also see that OHR steadily improves as the relative cache size increases, which conforms to the fact that more requests will be satisfied by the caches as the cache size becomes larger. Figure 3.8 also plots the results of HSL as a function of the relative cache size. It can be seen that HSL for our model is lower than that of the other models. We can

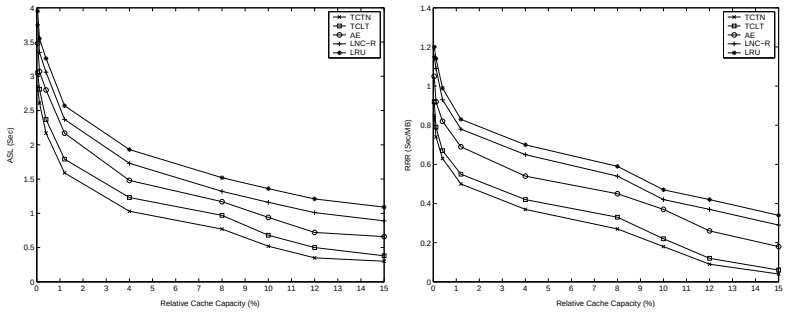

Figure 3.7: Experiment on ASL and RRR

also see that HSL decreases as the relative cache size increases.

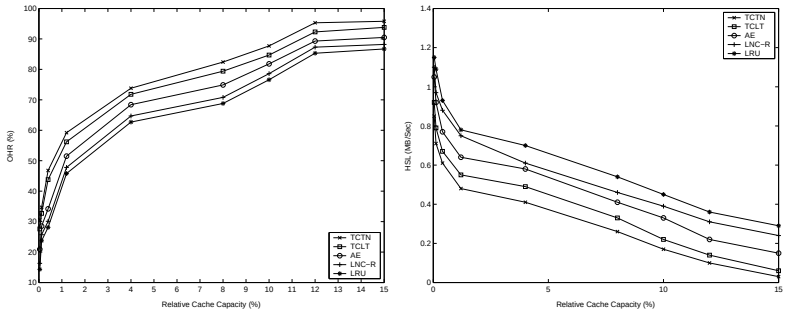

Figure 3.8: Experiment for OHR and HSL

Impact of Object Access Frequency

This experiment set examines the impact of object access frequency distribution on the performance results of different models. Figure 3.9 shows the performance results of DSR, RRR, and OHR for the values of Zipf parameter α from 0.2 to 1.0.

We can see that $TCTN$ consistently provides the best performance over a wide range of object access frequency distributions. Specially, $TCTN$ reduces or improves DSR by 30.4 percent, 24.4 percent, 21.3 percent, and 8.5 percent compared to LRU, $LNC-R$, AE, and $TCLT$, respectively; the default cache size used here (4 percent) is fairly large in the context of en-route caching due to the large network under consideration.

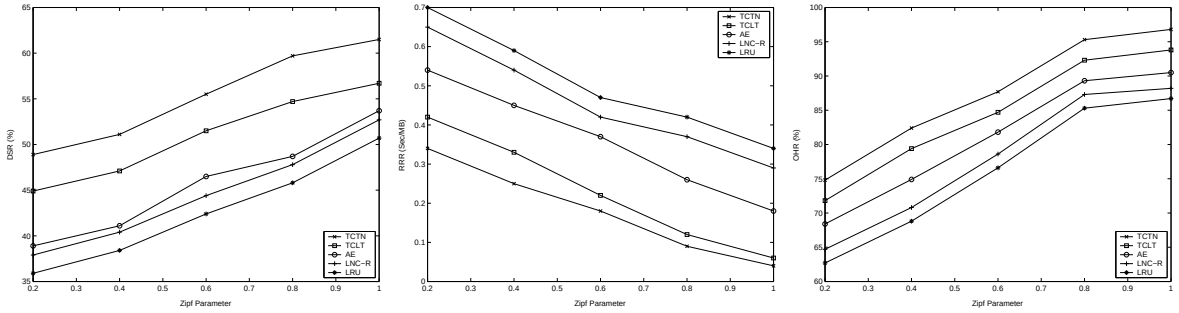

Figure 3.9: Experiment for DSR, RRR, and OHR

3.2 Transcoding Proxy Placement

3.2.1 Problem Formulation

Transcoding proxy caching is especially used to improve the performance of mobile appliances. Needless to say, the placement decision of transcoding proxies is crucial to the success of this idea. Therefore, it is necessary to find some methods to obtain the optimal placement decisions. To realize this, we formulate the placement problem of transcoding proxies in this section, i.e., finding the optimal locations to place a fixed number of transcoding proxies in a network such that the specified objective is achieved.

In the previous section, only a multimedia object is considered. In this section, we extend the solutions proposed in the previous section to solve the problem of proxy placement for coordinated en-route transcoding proxy caching by considering all the objects. Assume that the set of all the media objects is denoted by $O = (O_1, O_2, \cdots, O_l)$. In our analysis, we assume that each media object O_h has m_h versions, denoted by $O_h = (A_{h,1}, A_{h,2}, \cdots, A_{h,m_h})$. We also assume that the access frequencies for $A_{h,i}$ from v_j, denoted by $f_{A_{h,i},v_j}$, are independent.

3.2.2 Dynamic Programming-Based Solution for Linear Networks

Based on Definition 3, we define the *aggregate cost gain* of placing K transcoding proxies on the path from node 0 to n as follows.

Definition 6 *Given K, $f_{A_{h,i},v_j}$, c_{v_i,v_j} and $w(A_{h,i}, A_{h,j})$ where $i = 1, 2, \cdots, m$, $j = 1, 2, \cdots, n$, and $h = 1, 2, \cdots, l$. Let $v_1, v_2, \cdots, v_K$ be a set of K nodes such that $1 \leq v_1 \leq v_2 \leq \cdots \leq v_K \leq n$. The aggregate cost gain of placing K transcoding proxies at $v_1, v_2, \cdots, v_K$ which is denoted by $G(n : v_1, v_2, \cdots, v_K)$ is defined as*

$$G(n : v_1, v_2, \cdots, v_K) = \sum_{j=1}^{K} \sum_{h=1}^{l} \sum_{A_{h,k} \in D(B_{h,v_j})} \left(f_{A_{h,k},v_j} - f_{A_{h,k},v_{j+1}}\right) \cdot m(A_{h,k}, v_j) \quad (3.4)$$

where $v_j^+(A_{h,k})$ is the nearest higher level node of v_j at which the requests for $A_{h,k}$ are satisfied. If $K = 0$, then we define $G(n : \phi) = 0$.

Obviously, our objective is to compute the locations for placing K transcoding proxies in a subset of nodes $\{v_1, v_2, \cdots, v_K\}$ that maximizes the aggregate cost gain as defined in Equation (3.4), which is generalized as an optimization problem as follows.

$$\begin{aligned} &\max_{(v_1,v_2,\cdots,v_K)} G(n : v_1, v_2, \cdots, v_K) \\ &\qquad = \sum_{j=1}^{K} \sum_{h=1}^{l} \sum_{A_{h,k} \in D(B_{h,v_j})} \left(f_{A_{h,k},v_j} - f_{A_{h,k},v_{j+1}}\right) \cdot m(A_{h,k}, v_j) \end{aligned} \quad (3.5)$$

Before solving Problem (3.5), we define another optimization problem [4] by introducing introducing a parameter α in Problem (3.5).

[4] In this section, we can this problem an n-optimization problem.

$$\max_{(v_1,v_2,\cdots,v_K)} H(n,\alpha : v_1, v_2, \cdots, v_k) = \sum_{j=1}^{k}\sum_{h=1}^{l}\sum_{A_{h,k}\in D(B_{h,v_j})} \left[\left(f_{A_{h,k},v_j} - f_{A_{h,k},v_{j+1}}\right)\cdot m(A_{h,k},v_j) - \alpha\right] \tag{3.6}$$

Here, we call α in this section *control parameter* because it plays an important role on the solution to Problem (3.5). Before presenting an algorithm to solve Problem (3.5), we discuss the relationship between the solutions to Problem (3.5) and Problem (3.6). From Problem (3.6), we can easily get that the number of transcoding proxies to be placed in the network is relevant to the control parameter α greatly. The crucial observation is that the optimal number of transcoding proxies to be placed is a monotonically decreasing function of α, that is, as α increases, the optimal number decreases monotonically. Hence, the proper selection of α determines the optimal number of transcoding proxies to be placed among the en-route nodes in a network. Therefore, we can determine the optimal locations for placing K transcoding proxies among the en-route nodes by tuning the control parameter α. The relationship between the optimal number of transcoding proxies to be placed, denoted by k^*, and the parameter α can be visualized in Figure 3.10. Therefore, we can solving Problem (3.5) by tuning the parameter α in (3.6) until we find the exact number K.

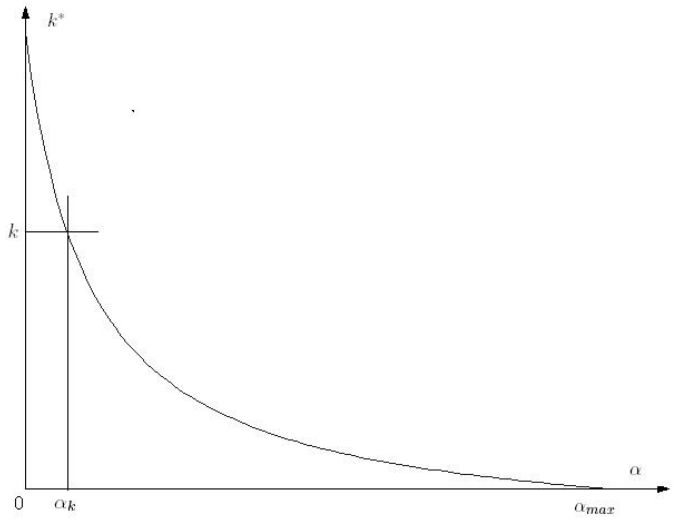

Figure 3.10: Relationship between k^* and α

Now we start to present a solution to Problem (3.6). The following corollary shows that an optimal solution to Problem (3.6) must contain optimal solutions to some subproblems. The proof of this corollary can be easily deduced from Theorem 11.

Corollary 7 *Suppose that $\{v_1, v_2, \cdots, v_I\}$ is an optimal solution to the n-optimization problem (Problem (3.6)) and $\{u_1, u_2, \cdots, u_p\}$ is an optimal solution to the $(v_I - 1)$-optimization problem, then $\{u_1, u_2, \cdots, u_l, v_I\}$ is also an optimal solution to the n-optimization problem.*

Before presenting a dynamic programming-based solution for Problem (3.6), we give the following definition.

Definition 7 *Define H_n^* to be the maximum aggregate cost gain of $H(n, \alpha : v_1, v_2, \cdots, v_k)$ obtained by solving the n-optimization problem and I_n the maximum index in the optimal solution. If the optimal solution is an empty set, define $I_n = -1$.*

Obviously, we have $I_0 = -1$ and $H_0^* = 0$. From Theorem 7, we know that if $I_r > 0$,

$$H_{I_r} = H_{I_r-1} + \sum_{h=1}^{l} \sum_{A_{h,k} \in D(B_{h,v_{I_r}})} \left[\left(f_{A_{h,k},v_{I_r}} - f_{A_{h,k},v_{I_r+1}} \right) \cdot m(A_{h,k}, v_{I+r}) - \alpha \right] \tag{3.7}$$

Therefore, we can check all possible locations of I_r $(0 \leq r \leq n)$ and select the one that maximizes $H(r : v_1, v_2, \cdots, v_k)$. So we have

$$\begin{cases} H_0^* = 0 \\ H_r^* = \max\limits_{1 \leq v_i \leq r} \{0, H_{v_i-1}^* + \sum\limits_{h=1}^{l} \sum\limits_{A_{h,k} \in D(B_{h,v_i})} \left[\left(f_{A_{h,k},v_i} - f_{A_{h,k},v_{i+1}} \right) \cdot m(A_{h,k}, v_i) - \alpha \right] \} \end{cases} \tag{3.8}$$

and

$$\begin{cases} I_0 = -1 \\ I_r = \begin{cases} -1 \;\; if \;\; H_r^* = 0 \\ v \quad if \;\; H_r^* = H_{v-1}^* + \sum\limits_{A_{h,k} \in D(B_{h,v})} \left[\left(f_{A_{h,k},v} - f_{A_{h,k},v+1} \right) \cdot m(A_{h,k}, v) - \alpha \right] \end{cases} \end{cases} \tag{3.9}$$

Based on Theorem 7 and the recurrences above, Problem (3.6) can be solved using dynamic programming. After computing H_n^* and I_n, we can start from $v_r = I_n$ and obtain all the locations iteratively. Therefore, the original problem, i.e., Problem (3.5), can be solved by tuning the control parameter α in Problem 3.6 recurrently.

It is easy to see that the time complexity of the dynamic programming solution is $O(n^2lm)$, where n is the number of nodes, l is the number of the media objects, and m the maximum number of versions for all the media objects.

3.2.3 Dynamic Programming-Based Algorithm for Tree Networks

In this section, we consider two cases for the problem of transcoding proxy placement for tree networks.

- **Without Constraint on the Number of Proxies**

Based on Definition 3, the problem of proxy placement for coordinated en-route transcoding proxy caching for tree T_r, without constraint on the number of transcoding proxies, is defined as the optimization problem below.

$$\max_{P_r} G(T_r, P_r) = \max_{P_r} \sum_{v \in P_r} \sum_{i=1}^{l} \sum_{A_{i,x} \in \theta(H_{i,v})} \left(f_{A_{i,x},v} - \sum_{z \in B_v^-(A_{i,x})} f_{A_{i,x},z} \right) m(A_{i,x}, v) - l(H_{i,v}) \tag{3.10}$$

where $P_r \subseteq V$.

Obviously, our objective is to compute the locations for placing transcoding proxies in a subset of nodes P_r that maximizes $G(T_r, P_r)$. Similarly, we can formulate the proxy placement problem of placing k transcoding proxies by adding a constraint that $|P_r| = k$.

Before presenting a solution to Problem (3.10), we define the following optimization problem for tree $T_{r,w}$ as below.

$$\max_{P_{r,w}} G(T_r, P_{r,w}) = \max_{P_{r,w}} \sum_{v \in P_{r,w}} \sum_{i=1}^{l} \sum_{A_{i,x} \in \theta(H_{i,v})} \left(f_{A_{i,x},v} - \sum_{z \in B_v^-(A_{i,x})} f_{A_{i,x},z} \right) m(A_{i,x}, v) - l(H_{i,v}) \quad (3.11)$$

where $P_{r,w} \subseteq D(w) \cup \{w\}$. Here, tree $T_{r,w}$ is a subtree of T_r whose node set is $D_w \cup \{w\}$, where $w \in D(r)$.

Based on theorems 13 and 14, the original problem, i.e., Problem (3.10), can be solved using dynamic programming with the following recurrences.

- Suppose that $v \in V$ and $C(v) = \{v_1, v_2, \cdots, v_{t_1}\}$. If $t_1 = 0$, then $P_v^* = \phi$; otherwise, $P_v^* = \cup_{i=1}^{t_1} P_{v,v_i}^*$.
- Suppose that $u \in D(v)$ and $C(u) = \{u_1, u_2, \cdots, u_{t_2}\}$. If $t_2 = 0$, then

$$P_{v,u}^* = \begin{cases} \{u\} & \text{if } \pi(u,v) \geq 0 \\ \{\phi\} & \text{Otherwise} \end{cases}$$

where $\pi(u,v) = \sum_{i=1}^{l} \sum_{A_{i,x} \in \theta(H_{i,u})} f_{A_{i,x},u}(c(v,u) + w(H_{i,u}, A_{i,x}))$.

Otherwise,

$$P_{v,u}^* = \begin{cases} \cup_{i=1}^{t_2} P_{v,u_i}^* & \text{if } \rho(u,v) \geq 0 \\ P_u^* \cup \{u\} & \text{Otherwise} \end{cases}$$

where $\rho(u,v) = G(T_{v,u}, \cup_{i=1}^{t_2} P_{v,u_i}^*) - G(T_{v,u}, P_u^* \cup \{u\})$.

It is easy to see that the time complexity of the dynamic programming-based algorithm is $O(n^2 lm)$, where n is the number of nodes, l is the number of the multimedia objects, and m the maximum number of versions for all the multimedia objects.

- **With Constraint on the Number of Proxies**

In the following, we will solve the problem of finding the optimal locations for placing k transcoding proxies. This problem can be defined as follows.

$$\max_{|P_r|=k} G(T_r, P_r) = \max_{|P_r|=k} \sum_{v \in P_r} \sum_{i=1}^{l} \sum_{A_{i,x} \in \theta(H_{i,v})} \left(f_{A_{i,x},v} - \sum_{z \in B_v^-(A_{i,x})} f_{A_{i,x},z} \right) m(A_{i,x}, v) - l(H_{i,v}) \quad (3.12)$$

where $P_r \subseteq V$.

Before presenting an algorithm to solve Problem (3.12), we define another problem.

$$\max_{P_r} G(T_r, P_r, \alpha) = \max_{P_r} \sum_{v \in P_r} \sum_{i=1}^{l} [\sum_{A_{i,x} \in \theta(H_{i,v})} \left(f_{A_{i,x},v} - \sum_{z \in B_v^-(A_{i,x})} f_{A_{i,x},z} \right) m(A_{i,x}, v) - l(H_{i,v}) - \alpha] \quad (3.13)$$

where $P_r \subseteq V$.

Now we discuss the relationship between the solutions to Problems (3.12) and (3.13). From Problem (3.13), we can easily get that the number of transcoding proxies to be

placed in the network is highly relevant to the parameter α. Hence, the proper selection of α determines the optimal number of transcoding proxies to be located among the en-route nodes in a network. The crucial observation is that the optimal number of transcoding proxies to be located is a monotonically decreasing function of α, that is, as α increases, the optimal number decreases monotonically. Therefore, we can determine the optimal locations for placing k transcoding proxies among the en-route nodes by tuning the parameter α. The relationship between k^* and α can be visualized in Figure 3.11, where k^* is the optimal number of transcoding proxies to be placed obtained from Problem (3.12). Therefore, we can solve Problem (3.13) by tuning the parameter α in Problem (3.12) until we find the exact number k since a solution to Problem (3.12) has been discussed.

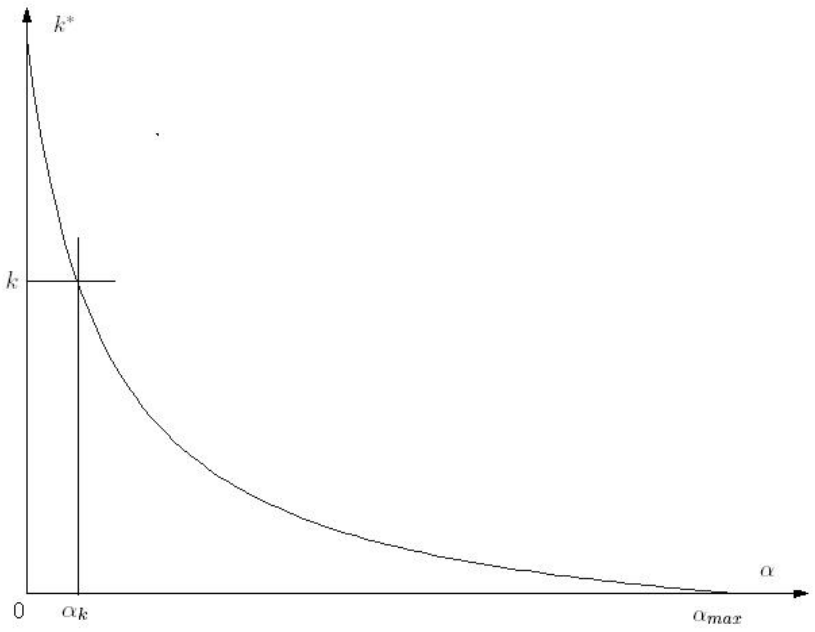

Figure 3.11: Relationship between k^* and α

3.2.4 Simulation Model

In this section, we describe the simulation model used for performance analysis. We have performed extensive simulation experiments for comparing the results of our model with existing models.

To the best of our knowledge, it is difficult to find true trace data in the open literature to simulate our model. Therefore, we generated the simulation model from empirical results presented in [7, 11, 14].

Table 3.4 lists the parameters and their values used in our simulation.

The network topology is randomly generated by the Tier program [14]. We have conducted experiments for many topologies with different parameters and found that the relative performance of our model was insensitive to topology changes. Here, we list only the experimental results for one topology due to space limitations. Table 3.4 shows the characteristics of this topology and the workload model, which were chosen from the open literature and are considered to be reasonable.

Table 3.4: Parameters Used in Our Simulation

Parameter	Value
Number of WAN Nodes	200
Number of MAN Nodes	200
Delay of WAN Links	Exponential Distribution $p(x) = \theta^{-1}e^{-x/\theta}$ $(\theta = 0.46\ Sec)$
Delay of MAN Links	Exponential Distribution $p(x) = \theta^{-1}e^{-x/\theta}$ $(\theta = 0.07\ Sec)$
Number of Servers	100
Number of Web Objects	1000 objects per srever
Average Web Object Size	$28KB$
Object Access Frequency	Zipf-Like Distribution $\frac{1}{i^{\alpha}}$ $(i = 0.8)$
Request Rate Per Node	$U(1, 9)$ requests per second
Transcoding Rate	$18KB/Sec$

The WAN (Wide Area Network) is viewed as the backbone network to which no servers or clients are attached. Each MAN (Metropolitan Area Network) node is assumed to connect to a content server. The number of objects generated is N and these N objects are divided into two types: *text* and *media*. We assume that each media object has five versions and the transcoding graph is as shown in Figure 3.12. The sizes of each version are assumed to be 100 percent, 80 percent, 60 percent, 40 percent, and 20 percent of the original object size. The transcoding delay is determined as the quotient of the object size to the transcoding rate. In our experiments, the client at each MAN node randomly generates the requests, and the average request rate of each node follows the distribution of $U(1, 9)$, where $U(x, y)$ represents a uniform distribution between x and y. The access frequencies of both the content servers and the objects maintained by a given server follow a Zipf-like distribution [11, 69]. Specifically, the probability of a request for an object O in server S is proportional to $1/(i^{\alpha} \cdot j^{\alpha})$, where S is the ith most popular server and O is the jth popular object in S. Both the delay of MAN links and WAN links follow an exponential distribution, where the average delay for WAN links is 0.46 seconds and the average delay for WAN links is 0.07 seconds.

The total cost, including the cost for transferring request and the cost for transcoding, is calculated by the access delay. For simplicity, the delay caused by sending the request and the relevant response for that request is proportional to the size of the requested object. Here, we consider the average object sizes for calculating all delays, including the propagation delay, the transmission delay, transcoding delay, and the searching delay. The cost function is taken to be the delay of the link, which means that the cost in our model is interpreted as the access latency in our simulation.

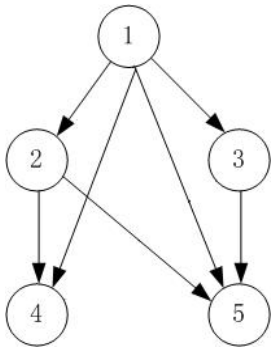

Figure 3.12: Transcoding Graph for Simulation

3.2.5 Performance Results

In this section, we compare the performance of our model with existing models proposed in the literature in terms of several performance metrics.

In addition to the model presented in Section 3.2.1, we also consider the following models for comparison purposes.

- *Random*: The random placement model places a fixed number of transcoding proxies in a network randomly. This model is obviously not optimal.
- *Linear proposed in Section 3.2.2*: This solution is used to decide the optimal locations for placing a fixed number of transcoding proxies for linear-array network topology. From the point view of network topology, this model can be viewed as a special case of our model.

We still employ the performance metrics defined in previous section in our simulation, including delay-saving ratio (DSR , average access latency (ASL), request response ratio (RRR), version hit ratio (VHR)[5], and content hit ratio (CHR)[6]. In the following figures, $Tree$ denotes the results for our proposed model, $Linear$ the results for the model for linear topology, and $Random$ the results for the random placement model.

The first experiment was to investigate DSR as a function of the number of transcoding proxies and Figure 3.13 shows the simulation results. As presented in Figure 3.13, we can see that our *Tree* model outperforms the other two models. Since the *Linear* model can be viewed as a special case of our model, its performance can not be better than our model. The *Random* model is obviously not optimal and its performance must be worst as well. Specifically, the mean improvements of DSR over *Linear* and *Random* are about 22.9 percent and 7.3 percent, respectively.

Figure 3.14 plots the simulation results of ASL and RRR as a function of the number of transcoding proxies. As we know, the lower the ASL or the RRR, the better the performance. We can see that all models provide steady performance improvement as the number of transcoding proxies increases. We can also see that our model significantly improves both ASL and RRR compared to the other models considered. This is because our model determines the optimal locations for coordinated en-route transcoding proxy

[5] VHR is defined as the ratio of the number of requests satisfied by the exact versions in the nearest higher level transcoding proxies as a whole to the total number of requests.

[6] CHR is defined as the ratio of the number of requests satisfied by the exact versions or the more detailed versions in the nearest higher level transcoding proxies as a whole to the total number of requests.

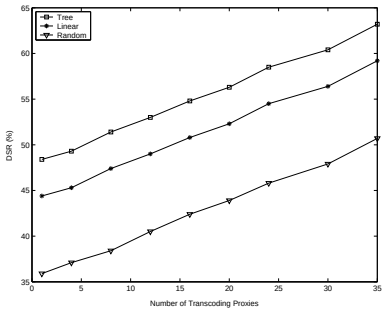

Figure 3.13: Experiment on DSR

caching, while the other models consider transcoding proxy placement for a linear topology or randomly. The average improvements of ASL over *Linear* and *Random* are about 8.2 percent and 20.9 percent, respectively.

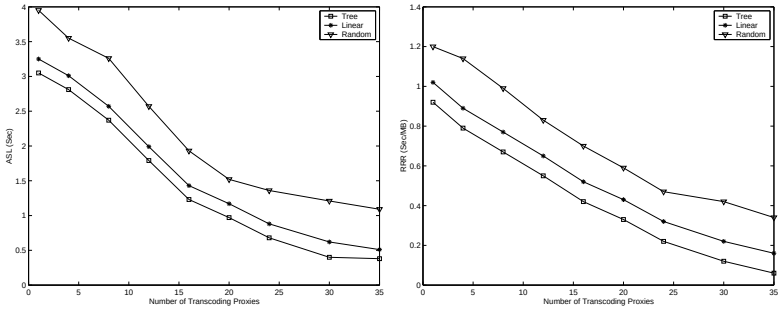

Figure 3.14: Experiment on ASL and RRR

Figure 3.15 shows the simulation results of VHR and CHR as functions of the number of transcoding proxies. As we know, the higher the VHR or the CHR, the better the performance. We can see that all models provide steady performance improvement as the number of transcoding proxies increases. We can also see that our model greatly improves both VHR and CHR compared to the other two models. This is because of the fact that our model obtains the optimal locations for tree networks, while the other models consider transcoding proxy placement for linear topology or in a random way. Specifically, the mean improvements of VHR over *Linear* and *Random* are about 6.1 percent and 20.5 percent, respectively.

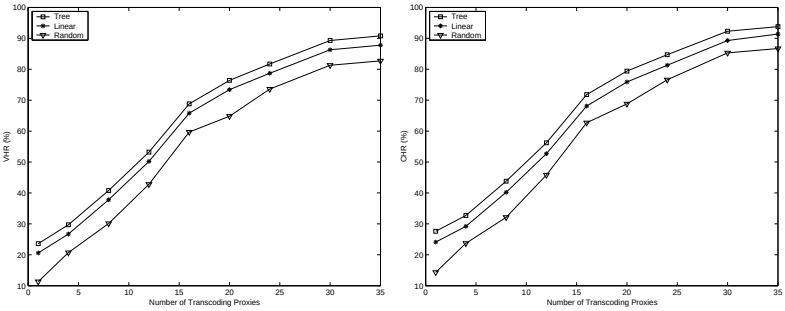

Figure 3.15: Experiment on VHR and CHR

3.3 Multimedia Object Placement for Transparent Data Replication

3.3.1 Problem Formulation

The network topology in this section is modelled as a graph $G = (V, E)$, where $V = \{v_0, v_1, \cdots, v_n\}$ is the set of nodes or vertices, and E is the set of edges or links. For a multimedia object O, we assume that it has m versions: $A_1, A_2, \cdots, A_m$. For each version of object O, we associate each link $(u, v) \in E$ a nonnegative cost $L_k(u, v)$, which is defined as the cost of sending a request for version A_k and the relevant response over the link (u, v). In particular, $L_k(u, u) = 0$. If a request goes through multiple network links, the cost is the sum of the cost on all these links. The cost in our analysis is calculated from a general point of view. It can be different performance measures such as delay, bandwidth requirement, and access latency, or a combination of these measures. Let $f_{i,j}$ be access frequency of version A_j from node v_i.

Now we start to formulate the problem of multimedia object placement for data transparent replication (*MOP problem*). Consider the snapshot when a request for a specified version of a multimedia object is being served (see Figure 3.16). Here v_0 denotes the content server which contains all versions of object O. v_n is the client and $v_1, v_2, \cdots, v_{n-1}$ are the nodes on the path from the client to the server. We can see that a request for a version of a multimedia object from a node can be satisfied either by this node or by upstream nodes (transcoding if necessary) until it arrives at the server at which no transcoding is necessary. Therefore, the total access cost can be decomposed into two parts: transcoding cost and transmission cost. Our objective is to find the exact version of a multimedia object to be placed at each node on the path from v_1 to v_n so that the total access cost is minimized. Note that all requests at node v_0 can be satisfied at zero cost. If we denote

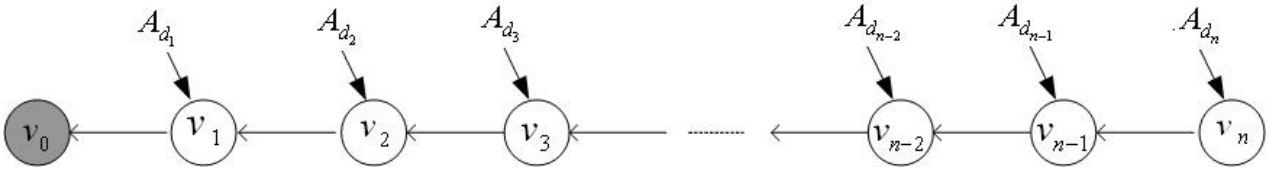

Figure 3.16: System Model for Multimedia Object Caching

A_{d_i} ($d_i \in \{1, 2, \cdots, m\}$) as the version cached at node v_i, then the total access cost of caching $A_{d_1}, A_{d_2}, \cdots, A_{d_n}$, denoted by $C(X)$, is defined as follows:

$$C(X) = \sum_{i=1}^{n} \sum_{j=1}^{m} f_{i,j} \min_{0 \le k \le i} \{L_j(v_i, v_k) + T(A_{d_k}, A_j)\} \tag{3.14}$$

where $X = (A_{d_1}, A_{d_2}, \cdots, A_{d_n})$ and $T(A_{d_k}, A_j) = \begin{cases} 0 & \text{if } k = 0 \\ t(A_{d_k}, A_j) & \text{if } k \neq 0 \end{cases}$.

Obviously, our objective is to obtain $X^* = (A_{d_1^*}, A_{d_2^*}, \cdots, A_{d_n^*})$ such that $C(X^*) = \min_X \{C(X)\}$.

Before we solve the MOP problem based on the cost function as given in Equation (3.14), we can make the following assumptions.

- *Assumption* 1. $L_j(v_i, v_k) = (i-k)L$ for all $1 \leq j \leq m$ as there are $i-k$ links on the path between nodes v_i and v_k, and the cost on each link for each version is L.
- *Assumption* 2. The transcoding graph is a linear array and the transcoding cost between any two adjacent versions is constant, i.e., $t(A_i, A_j) = \sum_{k=i}^{j-1} t(A_k, A_{k+1}) = (j-i)^+ T$, where $x^+ = x$ if $x \geq 0$ else $x^+ = \infty$.
- *Assumption* 3. $(\delta - 1)T \leq L$, and $\delta T > L$ for some positive integer δ.

If there does not exist δ such that *Assumption* 3 can be satisfied, i.e., $L \gg T$ or $T \gg L$. Obviously, these are two special cases. If $L \gg T$, then version A_{d_i} should be cached so that no transmission cost is necessary to incur, where $d_i = \min\{j | f_{i,j} > 0, 1 \leq j \leq m\}$. If $T \gg L$, this case is not trivial and is equivalent to the en-route caching problem of caching m objects on a linear network of n nodes, where transcoding cost is prohibited.

With the above assumptions, the MOP problem can be simplified as follows:

$$C(X) = \sum_{i=1}^{n} \sum_{j=1}^{m} f_{i,j} \min \left\{ \min_{1 \leq k \leq i} \left\{ (i-k)L + (j-d_k)^+ T \right\}, iL \right\} \tag{3.15}$$

3.3.2 Dynamic Programming-Based Solutions

In this section, we first consider the case of $n = 1$, i.e., there is only one node, and then discuss the case of $n > 1$.

The Case of $n = 1$

Before presenting the optimal solutions, we give a brief explanation of the significance for solving the MOP problem for the case of $n = 1$. For cache replacement problem, a crucial thing is to determine the objects to be removed so that the cost loss is minimized and the free space is enough to accommodate the new object. The following solutions are of great importance in deciding the objects to be removed for transcoding-enabled cache replacement problem.

First, we begin by computing the access cost of caching only one version A_k at node v_1 with $1 \leq k \leq m$. Intuitively, all the requests for version A_i with $i < k$ will be handled by server v_0, while some of the requests for A_i with $i \geq k$, depending on the transcoding cost and the transmission cost, will be taken care of by transcoding from version A_k. Based on the cost function defined by Equation (3.15), the total access cost of caching only version A_k at node v_1 is computed as follows:

$$C(A_k) = \sum_{i=1}^{k-1} f_{1,i} L + \sum_{i=k}^{m} f_{1,i} \min\{(i-k)T, L\} \tag{3.16}$$

Since version A_k is cached at node v_1, we can see (with Assumption 3) that δ is such a parameter that the request for version A_i will be served by the local node if $0 < i-k < \delta$, and the request for version A_i will be served by the server if $i - k \geq \delta$.

Based on Equation (3.16), $C(A_k)$ can be further defined as follows:

$$C(A_k) = \begin{cases} \sum_{i=1}^{k-1} f_{1,i}L + \sum_{i=k}^{k+\delta-1} f_{1,i}(i-k)T + \sum_{i=k+\delta}^{m} f_{1,i}L & \text{if } k+\delta \leq m \\ \sum_{i=1}^{k-1} f_{1,i}L + \sum_{i=k}^{m} f_{1,i}(i-k)T & \text{if } k+\delta > m \end{cases} \tag{3.17}$$

It is easy to see that $C(A_1)$ can be calculated in $O(m)$ time. As

$$\begin{aligned} C(A_{k+1}) &= \begin{cases} C(A_k) + f_{1,k}L - \sum_{i=k+1}^{k+\delta-1} f_{1,i}T + f_{1,k+\delta}((\delta-1)T - L) & \text{if } k+\delta \leq m \\ C(A_k) + f_{1,k}L - \sum_{i=k+1}^{m} f_{1,i}T & \text{if } k+\delta > m \end{cases} \\ &= C(A_k) + E(k) \end{aligned}$$

where

$$E(k) = \begin{cases} f_{1,k}L - \sum_{i=k+1}^{k+\delta-1} f_{1,i}T + f_{1,k+\delta}((\delta-1)T - L) & \text{if } k+\delta \leq m \\ f_{1,k}L - \sum_{i=k+1}^{m} f_{1,i}T & \text{if } k+\delta > m \end{cases}$$

and

$$E(k+1) = \begin{cases} E(k) - f_{1,k}L + f_{1,k+1}(L+T) \\ \quad -f_{1,k+\delta}(\delta T - L) + f_{1,k+\delta+1}(\delta-1)T - f_{1,k+\delta+1}L & \text{if } k+\delta < m \\ E(k) - f_{1,k}L + f_{1,k+1}(L+T) - f_{1,k+\delta}(\delta T - L) & \text{if } k+\delta = m \\ E(k) - f_{1,k}L + f_{1,k+1}(L+T) & \text{if } k+\delta > m \end{cases}$$

Thus, each $C(A_2), C(A_3), \cdots, C(A_m)$ can be done in constant time; Therefore, the MOP problem can be solved in $O(m)$ time. Regarding to the time complexity of solving the MOP problem, we have the following theorem.

Theorem 17 *Based on the cost function as given in Equation (3.16), the MOP problem for $\{A_1, A_2, \cdots, A_m\}$ by caching only one version (i.e., $n = 1$) can be solved in $O(m)$ time.*

Proof Since the cost function as given in Equation (3.17) is equivalent to the cost function as given in Equation (3.16) and the MOP problem based on the cost function as given in Equation (3.17) can be solved in $O(m)$ time, the MOP problem based on the cost function as given in Equation (3.16) can also be solved in $O(m)$ time. Hence, the theorem is proven. □

The second step is to extend the above solution to compute the optimal solution for caching two versions, A_{k_1} and A_{k_2}, at the same time at node v_1.

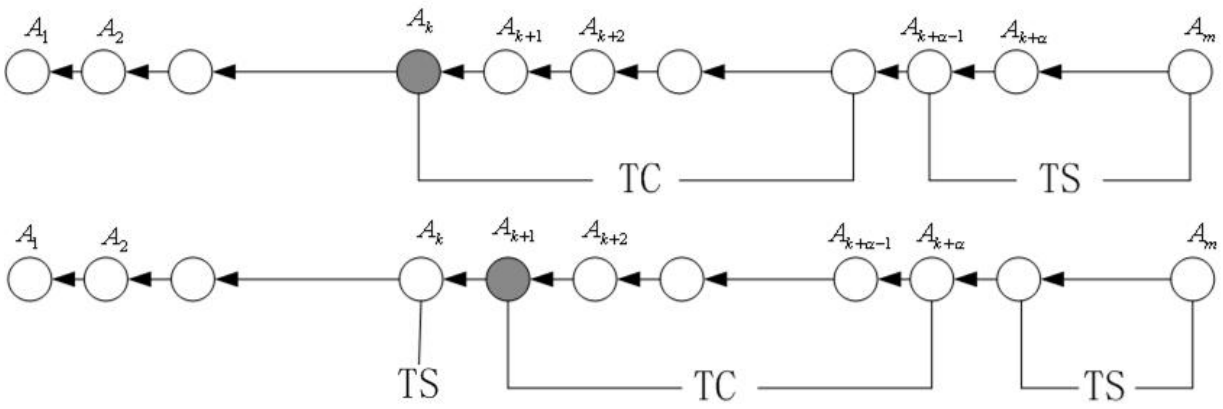

Figure 3.17: Relationship between $C(A_k)$ and $C(A_{k+1})$

Suppose that A_{k_1} and A_{k_2} are the two optimal versions to be cached. The key observation here is that A_{k_1} is also an optimal solution for the problem with $\{A_1, A_2, \cdots, A_{k_2-1}\}$ if $k_1 < k_2$, because the requests for $\{A_{k_2}, A_{k_2+1}, \cdots, A_m\}$ can only be served by A_{k_2}. Regarding to this observation, we have the following lemma.

Lemma 5 *Assume that A_{b_p} and A_{b_q} are the optimal solutions for the problem of caching only one version from the set of $\{A_1, A_2, \cdots, A_{p-1}\}$ and $\{A_1, A_2, \cdots, A_{q-1}\}$ respectively, then we have $b_p \leq b_q$ if $p < q$.*

Proof Without loss of generality, it is sufficient for us to prove that $b_p \leq b_{p+1}$ where $1 \leq b_p \leq p-1$ and $1 \leq b_{p+1} \leq p$. The proof is by contradiction. Assume that we have $b_p > b_{p+1}$. As A_{b_p} is the optimal version to be cached, we have $C_{1,p}(A_{b_p}) < C_{1,p}(A_{b_{p+1}})$. Let $C_{1,p}(A_i)$ denote the access cost of caching A_i for the MOP problem with $\{A_1, A_2, \cdots, A_{p-1}\}$. From the definition of the access cost function $C_{1,p}$ as given in Equation (3.16), adding A_p to the set $\{A_1, A_2, \cdots, A_{p-1}\}$ will increase both $C_{1,p}(A_{b_p})$ and $C_{1,p}(A_{b_{p+1}})$ by $f_{1,p} \min\{(p - b_p)T, L\}$ and $f_{1,p} \min\{(p - b_{p+1})T, L\}$ respectively. The increase to $C_{1,p}(A_{b_{p+1}})$ is no less than that to $C_{1,p}(A_{b_p})$ because $b_p > b_{p+1}$. So we have $C_{1,p+1}(A_{b_p}) < C_{1,p+1}(A_{b_{p+1}})$, which contradicts the fact that $C_{1,p+1}(A_{b_{p+1}})$ is the minimum access cost of caching $A_{b_{p+1}}$ for the problem with $\{A_1, A_2, \cdots, A_{p-1}, A_p\}$. Hence the lemma is proven. □

Based on Lemma 5, we can see that the feasible range of the optimal solution for the problem with $\{A_1, A_2, \cdots, A_q\}$ can be reduced if the optimal version for the problem with $\{A_1, A_2, \cdots, A_p\}$ has been obtained. So is the other case when the optimal solution for the problem with $\{A_1, A_2, \cdots, A_q\}$ is known, the feasible range of the optimal solution for the problem with $\{A_1, A_2, \cdots, A_p\}$ is also reduced. Therefore, we can find A_{b_p} and compute $C_{1,p}(A_p)$ by divide and conquer.

Let $D_{p,q}^{(k)}$ denote the minimum access cost of caching k versions for the MOP problem with $q-p$ versions, i.e., $A_p, A_{p+1}, \cdots, A_{q-1}$, where $1 \leq p < q \leq m$. Thus, $D_{1,p}^{(1)} = C_{1,p}(A_{b_p})$ and $D_{1,m+1}^{(1)} = \min_{1 \leq k \leq m} \{C_{1,m+1}(A_k)\}$. Based on Lemma 5, we have the following theorem on the time complexity of computing $D_{1,p}^{(1)}$ for $1 < p \leq m$.

Theorem 18 *All the p MOP problems for $\{A_1, A_2, \cdots, A_p\}$ where $1 \leq p \leq m$, i.e., $D_{1,p}^{(1)}$ for $1 < p \leq m$, can be computed in $O(m \log m)$ time.*

Proof Assume that there exists an integer θ such that $m = 2^\theta$. Based on Theorem 17, we can compute $D^{(1)}_{1,\frac{1}{2}m}$ in $O(m)$ time. Assume that $A_{b_{\frac{m}{2}}}$ is the optimal solution for the problem of caching only one version with $\{A_1, A_2, \cdots, A_{\frac{m}{2}-1}\}$, then we can find the optimal solution for the problem of caching only one version for $\{A_1, A_2, \cdots, A_{\frac{m}{4}}\}$ in $O(m)$ time. Similarly, $D^{(1)}_{1,\frac{3m}{4}}$ can also be computed by solving the problem of caching only one version with $\{A_1, A_2, \cdots, A_{\frac{3m}{4}-1}\}$. As we have already computed $C_{1,\frac{m}{2}}(A_y)$ where $y = \min(b_{\frac{m}{2}}, \frac{m}{2}-1)$, we can base on this result to compute $C_{1,\frac{3m}{4}}(A_y)$ for $\{A_1, A_2, \cdots, A_{\frac{3m}{4}-1}\}$ (by adding at most $\frac{m}{4}$ terms to $C_{1,\frac{m}{2}}(A_{\frac{m}{2}-1})$. We then compute $C_{1,\frac{3m}{4}}(A_y)$, $C_{1,\frac{3m}{4}}(A_{y+1})$, $\cdots$, $C_{1,\frac{3m}{4}}(A_{\frac{3m}{4}-1})$ in at most $O(\frac{3m}{4}-y)$ time. So it takes at most $O(m)$ time to compute $D^{(1)}_{1,\frac{m}{4}}$ and $D^{(1)}_{1,\frac{3m}{4}}$. According to the similar decomposition, $D^{(1)}_{1,\frac{m}{8}}$, $D^{(1)}_{1,\frac{3m}{8}}$, $D^{(1)}_{1,\frac{5m}{8}}$, and $D^{(1)}_{1,\frac{7m}{8}}$ can all be solved in $O(m)$ time. To be precise, let $A_{z_1}, A_{z_2}, A_{z_3}$ be the optimal versions for $\{A_1, A_2, \cdots, A_{\frac{m}{4}-1}\}$, $\{A_1, A_2, \cdots, A_{\frac{m}{2}-1}\}$, and $\{A_1, A_2, \cdots, A_{\frac{3m}{4}-1}\}$ respectively. The first step is to compute $C_{1,\frac{m}{8}}(A_1)$, and then $C_{1,\frac{3m}{8}}(A_{z_1})$, $C_{1,\frac{5m}{8}}(A_{z_2})$, and $C_{1,\frac{7m}{8}}(A_{z_3})$ from $C_{1,\frac{m}{4}}(A_{z_1})$, $C_{1,\frac{m}{2}}(A_{z_2})$, and $C_{1,\frac{3m}{4}}(A_{z_3})$ respectively. As the computation of each item takes $O(\frac{m}{8})$ time, this step takes $O(m)$ time in total. Then we can search the optimal solutions for $\{A_1, A_2, \cdots, A_{\frac{m}{8}-1}\}$, $\{A_1, A_2, \cdots, A_{\frac{3m}{8}-1}\}$, $\{A_1, A_2, \cdots, A_{\frac{5m}{8}-1}\}$, $\{A_1, A_2, \cdots, A_{\frac{7m}{8}-1}\}$ in the ranges $(1, \min\{z_1, \frac{m}{8}-1\})$, $(z_1, \min\{z_2, \frac{3m}{8}-1\})$, $(z_2, \min\{z_3, \frac{5m}{8}-1\})$, and $(z_3, \frac{7m}{8}-1)$ respectively. Since each step takes constant time, all these searches take no more than $O(m)$ time in total. After repeating this process $\log m$ times, we can finish computing $D^{(1)}_{1,p}$ for $1 < p \leq m$. This process can be viewed from Figure 3.18. Hence, the theorem is proven. $\square$

Now we can accomplish the problem of caching two versions in the following three steps.

- *Step* 1: Evaluate $D^{(1)}_{1,p}$ for $1 < p \leq m$, where $D^{(1)}_{1,p}$ denotes the minimum access cost of caching only one version for the MOP problem with $p-1$ versions, i.e., $A_1, A_2, \cdots, A_{p-1}$. In particular, $D^{(1)}_{1,m+1} = \min\limits_{1 \leq k \leq m}\{C_{1,m+1}(A_k)\}$.

- *Step* 2: Evaluate $\overline{D}_p$ for $2 \leq p \leq m$, where $\overline{D}_p$ is the access cost for versions $A_p, A_{p+1}, \cdots, A_m$ if A_p is cached at node v_1. $\overline{D}_p$ is defined as follows:

$$\overline{D}_p = \begin{cases} \sum\limits_{i=p}^{p+\delta-1} f_{1,i}(i-p)T + \sum\limits_{i=p+\delta}^{m} f_{1,i}L & \text{if } p+\delta \leq m \\ \sum\limits_{i=p}^{m} f_{1,i}(i-p)T & \text{if } p+\delta > m \end{cases}$$

- *Step* 3: Compute $D^{(2)}_{1,m}$, where $D^{(2)}_{1,m}$ is the minimum access cost of caching two versions for the problem with $\{A_1, A_2, \cdots, A_m\}$. $D^{(2)}_{1,m}$ is calculated as follows:

$$D^{(2)}_{1,m} = \min_{2 \leq p \leq m}\{D^{(1)}_{1,p} + \overline{D}_p)\}$$

The following theorem shows that $D^{(2)}_{1,m}$ is the minimum access cost of caching two versions the MOP problem.

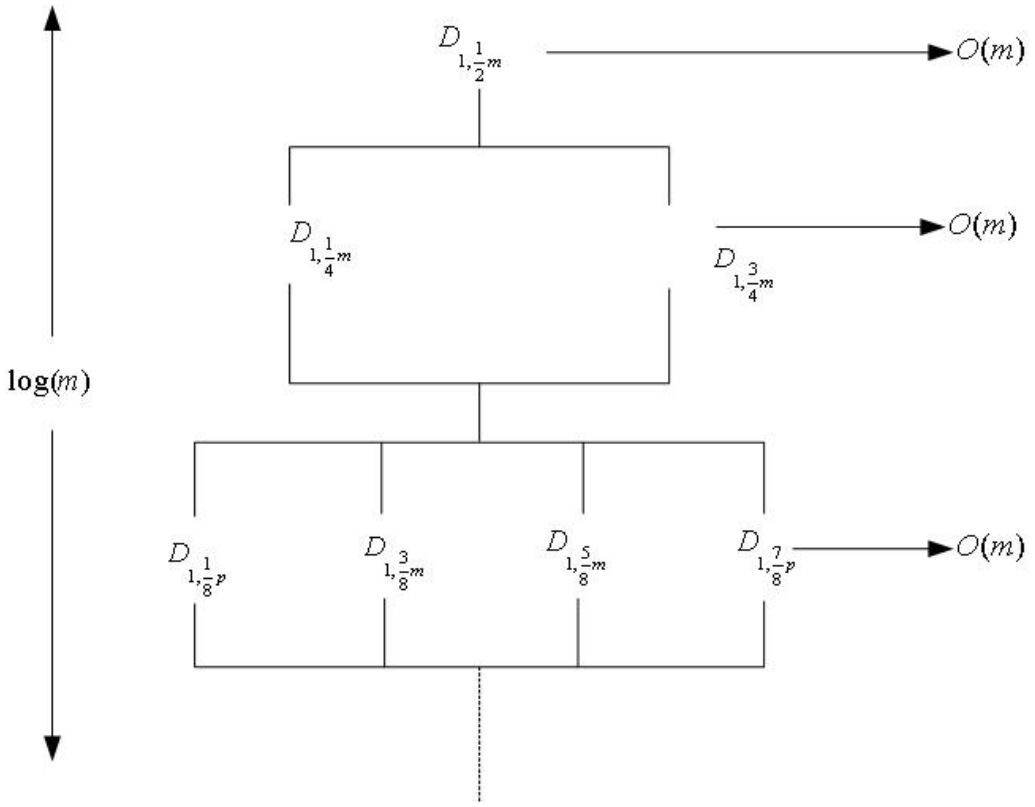

Figure 3.18: Decomposition Process of Theorem

Theorem 19 *$D^{(2)}_{1,m}$ is the minimum access cost of caching two versions for the MOP problem.*

Proof Assume that $D^{(2)}_{1,m} = D^{(1)}_{1,p^*} + \overline{D}_{p^*} = \min_{2 \le p \le m}\{D^{(1)}_{1,p} + \overline{D}_p)\}$. It is obvious from the computation of $D^{(2)}_{1,m}$ that b_{p^*} and A_{p^*} are the two versions which achieve the minimum access cost of caching two versions, where $D^{(1)}_{1,p^*} = C_{1,p^*}(b_{p^*})$. Hence, the theorem is proven. □

The following theorem shows the time complexity of computing $D^{(2)}_{1,m}$.

Theorem 20 *$D^{(2)}_{1,m}$ can be computed in $O(m \log m)$ time.*

Proof Since *Step* 1 can be computed in $O(m \log m)$ time (Theorem 18) and *Steps* 2 *and 3* both take $O(m)$ time, the total time for computing $D^{(2)}_{1,m}$ is $O(m \log m)$. Hence, the theorem is proven. □

After we have calculated $D^{(1)}_{1,p}$ for $1 \le p \le m$ in *Step* 1, we can obtain $D^{(2)}_{1,p}$ for all $2 \le p \le m$ in another $O(m \log m)$ time by divide and conquer, where $D^{(2)}_{1,p}$ is the minimum access cost of caching only two versions for the problem with $p-1$ versions, i.e., $A_1, A_2, \cdots, A_{p-1}$. The main idea is similar to Lemma 5 in the finding of $D^{(1)}_{1,p}$. Assume that $A_{b_{p_1}}$ and $A_{b_{p_2}}$ with $1 \le b_{p_1} < b_{p_2} < p$ are the two optimal versions cached in node v_1 for $A_1, A_2, \cdots, A_{p-1}$ to achieve the optimal access cost $D^{(2)}_{1,p}$. Similarly, $A_{b_{q_1}}$ and $A_{b_{q_2}}$ with

$1 \leq b_{q_1} < b_{q_2} < q$ are the two optimal versions cached in node v_1 for $A_1, A_2, \cdots, A_{q-1}$ to achieve the optimal access cost $D^{(2)}_{1,q}$. We can show with a similar argument with Lemma 5 that $b_{p_2} \leq b_{q_2}$ if $p < q$ and this property limits the range of searching for the optimal solutions. As in Theorem 18, the two optimal solutions in $D^{(2)}_{1,\frac{m}{2}}$ can be found in $O(m)$ time after knowing the optimal versions of $D^{(1)}_{1,p}$ for $1 < p \leq m$; then $D^{(2)}_{1,\frac{m}{4}}$ and $D^{(2)}_{1,\frac{3m}{4}}$ in another $O(m)$ time; then $D^{(2)}_{2,\frac{m}{8}}$, $D^{(2)}_{1,\frac{3m}{8}}$,$D^{(2)}_{1,\frac{5m}{8}}$, and $D^{(2)}_{1,\frac{7m}{8}}$ in another $O(m)$ time until $D^{(2)}_{1,p}$ for $2 < p \leq m$ are found after $\log m$ times. Therefore, the minimum access cost of caching three versions, denoted by $D^{(3)}_{1,m}$, can be computed similarly, i.e., $D^{(3)}_{1,m} = \min_{3 \leq p \leq m} \{D^{(2)}_{1,p} + \overline{D}_p)\}$, with at most $O(m \log m)$ time (similar to Theorem 21). Using the same idea, we can solve the problem of caching K versions in $O(Km \log m)$ time.

Let $D^{(K)}_{1,m}$ denote the minimum access cost of caching K versions from m versions, i.e., $A_1, A_2, \cdots, A_m$, then we have the following theorem on the time complexity of computing $D^{(K)}_{1,m}$.

Theorem 21 *$D^{(K)}_{1,m}$ can be computed in $O(Km \log m)$ time.*

Proof Based on the above analysis, we have $D^{(K)}_{1,m} = \min_{K \leq p \leq m} \{D^{(K-1)}_{1,p} + \overline{D}_p)\}$. Since $\overline{D}_p$ can all be computed in $O(m)$ time and we have showed that $D^{(1)}_{1,p}$ can be computed in $O(m \log m)$ time, we can easily prove that $D^{(K)}_{1,m}$ can be computed in $O(Km \log m)$ time by induction. Note that in the induction step, $D^{(K-1)}_{1,p}$ for $K - 1 < p \leq m$ is computed in $O((K-1)m \log m)$ time. Hence, the theorem is proven. □

The Case of $n > 1$

When $n > 1$, the problem of multimedia object placement can be visualized as given in Figure 3.19. We can see that the requests can be served in one of the following ways.

- 1. A request is served by the exact versions at its local cache or one of the upstream caches.
- 2. A request is served by more detailed versions according to transcoding at its local cache or one of the upstream caches.
- 3. A request is served by the original server (no transcoding is executed since all versions are stored at the server).

In Figure 3.19, a square symbol at (d_i, i) indicates that version A_{d_i} is cached at node v_i and a dot indicates the request for a specified version from a node. Each node has exactly one such square symbol. A request for version A_j at node v_i might be either served at node v_i by version A_{d_i} if $j \geq d_i$ with transcoding cost $(j - d_i)T$, or at node v_{i-1} with additional transmission cost L. In the latter case, a new request for A_j is created at node v_{i-1}. This process can be generalized as follows:

$$w(i,j) = \begin{cases} \min\{(j-d_i)^+ T, w(i-1,j) + L\} & \text{if } i \geq 1 \\ 0 \quad \text{if } i = 0 \end{cases}$$

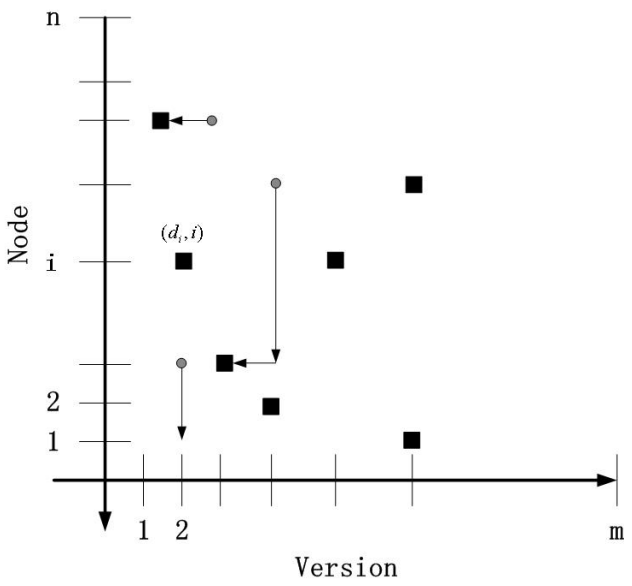

Figure 3.19: Request Flow for Multimedia Object Caching

where $w(i, j)$ is the access cost for request A_j at node v_i. In particular, if $L > mT$, then $w(i, j) = (j - d_i)T$ if $j \geq d_i$, i.e., transcoding is always performed if possible. Therefore, we can solve this problem using dynamic programming.

Assume that version A_j is cached at node v_i, and v_k is the smallest vertex, $k > i$, with a cached version, say A_z, more detailed than A_j, i.e, $z \leq j$ (see Figure 3.20). Let $B_{i,j,k} = \{(\alpha, \beta) | i \leq \alpha \leq k - 1, j \leq \beta \leq m\}$.

Assume that A_y is the most detailed version in Block $B_{i,j,k}$, which is cached at node v_x. Let $W(i, j, k)$ denote the minimum total access cost for serving all the requests in Block $B_{i,j,k}$. It is obvious that all the requests in Block C are served by version A_j at node v_i because the versions of all the requests in this block is more detailed than A_y, i.e. there does not exist a version in Block $B_{i,j,k}$ other than A_j that can provide the requested versions in this block since A_y is the most detailed version in Block $B_{i,j,k}$ besides A_j. Similarly, it is easy to see that the minimum total access cost for serving all the requests in Block A, i.e., $B_{x,y,k}$ and Block B, i.e., $B_{i,j,x+1}$ (see Figure 3.20), is $W(x, y, k) + W(i, j, y - 1)$. With a similar method for partitioning Block $B_{i,j,k}$, Blocks A and B can be divided recursively until the minimum total access cost for serving all the requests in each block, i.e., $W(x, y, k)$ and $W(i, j, y - 1)$, can be finally determined.

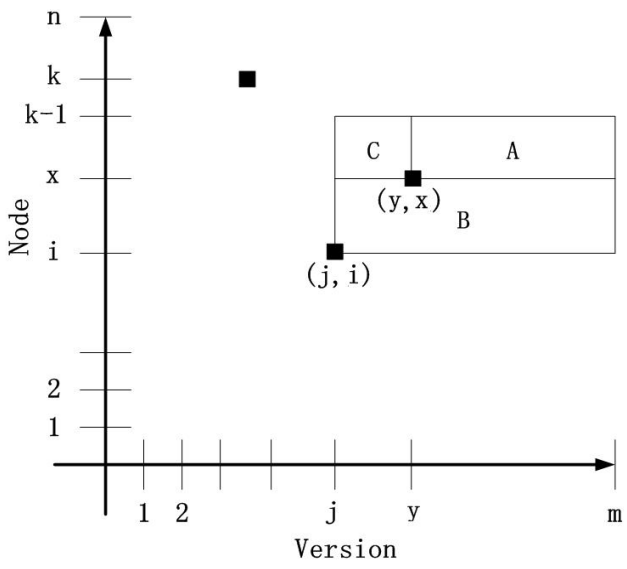

Figure 3.20: Block Definition for Multimedia Object Placement

Based on the above observation, $W(i,j,k)$ is defined as follows:

$$W(i,j,k) = \begin{cases} \min\limits_{i \le x < k; j \le y \le m} \{W(i,j,y-1) + W(x,y,k)\} \\ \qquad + \sum\limits_{x \le \alpha < k; j \le \beta \le y} f_{\alpha,\beta}((\alpha - i)L + (\beta - j)T) \\ \qquad (\text{for } 0 < i < k \le n; 1 \le j \le m) \\ \min\limits_{0 < x \le k; 1 \le y \le m} \{W(0,1,y-1) + W(x,y,k)\} + \sum\limits_{x \le \alpha < k; 1 \le \beta \le y} \beta L f_{\alpha,\beta} \\ \qquad (\text{for } i = 0, j = 1) \\ 0 \qquad (\text{for } i = k) \end{cases} \tag{3.18}$$

Now let us refer to the first equation in the recurrence formula above. The first term $W(i,j,y-1)$ is the total access cost for the requests in Block B and D, the second term is the total access cost for the requests in Block A, and the last term is the total access cost for the requests in Block C. The second equation is for the special case of $i = 0$ which denotes the original server, where transcoding is not necessary since all versions are stored there. To obtain the optimal solution, all possible values of i, j, and k must be checked. The following theorem shows the correctness of the above recurrence formula for $W(i,j,k)$.

Theorem 22 *Formula (3.18) is the correct recurrence formula for $W(i,j,k)$.*

Proof Without loss of generality, we only need to prove the correctness of the first equation in Formula (3.18) since the second equation can be easily derived in a similar way and the third equation is trivial.

Let $W'(i,j,k)$ denote the value of the right side of the first equation, i.e., $W'(i,j,k) = \min\limits_{i \le x < k; j \le y \le m} \{W(i,j,y-1) + W(x,y,k) + \sum\limits_{x \le \alpha < k; j \le \beta \le y} f_{\alpha,\beta}((\alpha - i)L + (\beta - j)T)$.

We now prove that $W^{'}(i,j,k)$ is the optimal access cost, i.e., $W^{'}(i,j,k) = W(i,j,k)$. Suppose $A_{d_i^*}, A_{d_{i+1}^*}, \cdots, A_{d_k^*}$ is the optimal placement in Block $B_{i,j,k}$, which makes $W(i,j,k)$ $= C(A_{d_i^*}, A_{d_{i+1}^*}, \cdots, A_{d_k^*})$. Thus, we can always divide Block $B_{i,j,k}$ into four parts according to y^* (see Figure 3.20), where $y^* = \max_{j<y\leq m}\{d_y^*\}$ and version A_{y^*} is cached at node v_{x^*}. Therefore, we have

$$\begin{aligned} & W(i,j,k) = C(A_{d_i^*}, A_{d_{i+1}^*}, \cdots, A_{d_{k-1}^*}) \\ = \; & W(i,j,y^*-1) + W(x^*,y^*,k) + \sum_{x^*\leq\alpha<k;j\leq\beta\leq y^*} f_{\alpha,\beta}((\alpha-i)L + (\beta-j)T) \\ \geq \; & \min_{i\leq x<k;j\leq y\leq m}\{W(i,j,y-1) + W(x,y,k) + \sum_{x\leq\alpha<k;j\leq\beta\leq y} f_{\alpha,\beta}((\alpha-i)L + (\beta-j)T) \end{aligned}$$

Now we want to prove $W^{'}(i,j,k) \geq W(i,j,k)$. Suppose there exists $(x^{'}, y^{'})$ such that $W^{'}(i,j,k) = \min_{i\leq x<k;j\leq y\leq m}\{W(i,j,y-1) + W(x,y,k) + \sum_{x\leq\alpha<k;j\leq\beta\leq y} f_{\alpha,\beta}((\alpha-i)L +$ $(\beta-j)T) = \min_{i\leq x^{'}<k;j\leq y^{'}\leq m}\{W(i,j,y^{'}-1) + W(x^{'},y^{'},k) + \sum_{x^{'}\leq\alpha<k;j\leq\beta\leq y^{'}} f_{\alpha,\beta}((\alpha-i)L +$ $(\beta-j)T)$. Thus, Block $B_{\alpha,\beta}$ can be divided into four parts according to $x^{'}$ and $y^{'}$. According to the definition of $W(i,j,k)$, we have $W(i,j,k) \leq \min_{i\leq x^{'}<k;j\leq y^{'}\leq m}\{W(i,j,y^{'}-$ $1) + W(x^{'},y^{'},k) + \sum_{x^{'}\leq\alpha<k;j\leq\beta\leq y^{'}} f_{\alpha,\beta}((\alpha-i)L + (\beta-j)T) = \min_{i\leq x<k;j\leq y\leq m}\{W(i,j,y-1) +$ $W(x,y,k) + \sum_{x\leq\alpha<k;j\leq\beta\leq y} f_{\alpha,\beta}((\alpha-i)L + (\beta-j)T) = W^{'}(i,j,k)$. Therefore, we have proved that $W^{'}(i,j,k) = W(i,j,k)$. Hence, the theorem is proven. □

The original multimedia object placement problem, i.e., with the cost function based on Equation (3.15), can be solved using dynamic programming with these recurrences. We can also see that the minimum access cost is $W(0,1,n)$. The detailed algorithm is given as follows.

Regarding to the time complexity of Algorithm 1, we have the following theorem.

Theorem 23 *Algorithm* 1 *can terminate in* $O(n^3m^2)$ *time, where* n *is the number of nodes and* m *is the number of versions.*

Proof The work of Procedure $Block(i,j,k)$ is to compute $W(i,j,k)$. It is easy to see that $W(i,j,k)$ has n^2m different entries and each entry is computed only once (it simply returns the value if it was computed before). Consider the time complexity of computing an entry in $Block(i,j,k)$. It takes two layers of loops to compute an element in $Block(i,j,k)$. The outside *for-loop* on x iterates at most n times, and the inner *for-loop* on y iterates at most m times. Thus, it takes at most $O(n^2m)$ time of comparisons to compute an entry. Therefore, it takes $O(n^2m \cdot nm) = O(n^3m^2)$ time to compute all entries in $Block(i,j,k)$. Hence, the theorem is proven. □

3.3.3 Simulation Model

In this section, the simulation model used for performance evaluation is described. We have performed extensive simulation experiments to compare our solution with existing

Table 3.5: Algorithm 1

Main()
begin
 Call $Block(0, 1, n)$
end

Procedure $Block(i, j, k)$
begin
 if $j = k$ then
  Exit;
 for x=i to k do
  for y=j to m do
   $Block(i, j, k) = Block(i, j, y - 1) + Block(x, y, k) + BlockC(j)$
  Return $Block(i, j, k)$
end

Function $BlockC(j)$
begin
 if $j = 0$ then
  $$BlockC(j) = \sum_{x \le \alpha < k; j \le \beta \le y} \beta L f_{\alpha,\beta}$$
 if $j > 0$ then
  $$BlockC(j) = \sum_{x \le \alpha < k; j \le \beta < y} f_{\alpha,\beta}[(\alpha - i)T + (\beta - j)L]$$
  Return $BlockC(j)$
end

solutions. The network in our simulation consists of numerous nodes and content servers. The system configuration is outlined in Section 3.3.3, and existing solutions used for the purpose of comparison are introduced in Section 3.3.3.

System Configuration

To the best of our knowledge, it is difficult to find true trace data in the open literature to execute such simulations. Therefore, we generated the simulation model from the empirical results presented in [1, 7, 11, 14, 23].

The network topology was randomly generated by the Tier program [14]. Experiments for many topologies with various parameters were conducted and the performance of our solution was found to be insensitive to topology changes. Here, only the experimental results for one topology are presented due to space limitations. The characteristics of this topology and the workload model are shown in Table 3.6, which were chosen from the open literature and are considered to be reasonable.

The WAN (Wide Area Network) is viewed as a backbone network to which no servers or clients are attached. Each MAN (Metropolitan Area Network) node is assumed to connect to a content server. Each MAN and WAN node is associated with an en-route cache. The objects generated are divided into two types: *text* and *multimedia*. Similar

Table 3.6: Parameters Used in Simulation

Parameter	Value
Number of WAN Nodes	200
Number of MAN Nodes	200
Delay of WAN Links	Exponential Distribution ($\theta = 1.5Sec$)
Delay of MAN Links	Exponential Distribution ($\theta = 0.7Sec$)
Number of Servers	100
Number of Web Objects	1000 objects per srever
Web Object Size Distribution	Pareto Distribution ($\mu = 6KB$)
Web Object Access Frequency	Zipf-Like Distribution ($\alpha = 0.7$)
Relative Cache Size Per Node	4%
Average Request Rate Per Node	$U(1, 9)$ requests per second
Transcoding Cost	$50KB/Sec$

to the studies in [11, 88], cache size is described as the total relative size of all objects available in the content server. In our experiments, the object sizes are assumed to follow a Pareto distribution and the average object size is $6KB$. We also assume that each multimedia object has five versions and that the transcoding graph is as shown in Figure 3.21. The sizes of each version are assumed to be 100 percent, 80 percent, 60 percent, 40 percent, and 20 percent of the original object size. The transcoding delay is determined as the quotient of the object size to the transcoding rate. In our experiments, the client at each MAN node randomly generates the requests, and the average request rate of each node follows the distribution of $U(1, 9)$, where $U(x, y)$ represents a uniform distribution between x and y. The access frequencies of both the content servers and the objects maintained by a given server follow a Zipf-like distribution [11, 69]. Specifically, the probability of a request for object O in server S is proportional to $1/(i^{\alpha} \cdot j^{\alpha})$, where S is the ith most popular server and O is the jth popular object in S. The delay of both MAN links and WAN links follows an exponential distribution; the average delay for WAN links is 1.5 seconds and the average delay for WAN links is 0.7 seconds. The cost

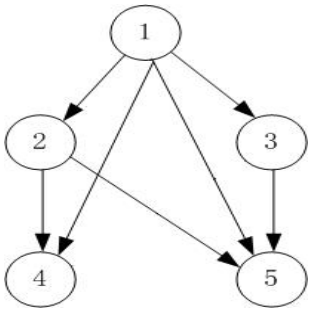

Figure 3.21: Transcoding Graph for Simulation

for each link is calculated by the access delay. For simplicity, the delay caused by sending the request and the relevant response for that request is proportional to the size of the requested object. Here, we consider the average object sizes for calculating all delays,

including the transmission delay and the transcoding delay. The cost function is taken to be the delay of the link, which means that the cost in our solution is interpreted as the access latency in our simulation.

We apply a "sliding window" technique, for estimating access frequency, to make our model less sensitive to transient workload [88]. Specifically, the access frequency is estimated by $N/(t - t_N)$, where N is the number of accesses recorded, t is the current time, and t_N is the Nth most recently referenced time (the time of the oldest reference in the sliding window). N is set to 2 in the simulation.

Existing Models

In addition to the solution proposed in Section 3.3.2, we also consider the following placement solutions for comparison purposes.

- SV: SV stores the same version of a multimedia object at each node when the request is sent back to the client from the server.
- MV: MV stores the most referred version of a multimedia object at each node as the request is returned back to the client from the server. Specifically, if $i^* = \max_{1 \le j \le m} \{f_{i,j}\}$, then version A_{i^*} is cached at node v_i.
- RV : RV randomly stores a version at each node.

3.3.4 Performance Evaluation

In this section, we compare the performance results of our solution with those solutions introduced in Section 3.3.3, in terms of several performance metrics. The performance metrics we used in our simulation include delay-saving ratio (DSR), average access latency (AST); request response ratio (RRR), object hit ratio (OHR), and average server load (ASL). In the following figures, SV, MV, and RV denote the results for the three solutions introduced in Section 3.3.3, and OV denotes the optimal solution proposed in Section 3.3.2.

Impact of Cache Size

In this experiment set, we compare the performance results of different solutions across a wide range of cache sizes, from 0.04 percent to 15.0 percent.

The first experiment investigates DSR as a function of the relative cache size at each node and Figure 3.22 shows the simulation results. As presented in Figure 3.22, we can see that our solution outperforms the others since it considers multimedia object placement by determining the optimal versions to be placed at each node, whereas existing solutions, including SV, MV, and RV, consider multimedia object placement heuristically or randomly. Specifically, the mean improvements of DSR over SV, MV, and RV are 4.3 percent, 17.9 percent, 19.8 percent, and 24.5 percent, respectively. Figure 3.23 describes the simulation results of ASL and RRR as a function of the relative cache size at each node. Clearly, the lower the ASL or the RRR, the better the performance. As we can see, all solutions provide steady performance improvement as the cache size increases. We can also see that OV significantly improves both ASL and RRR compared to SV, MV,

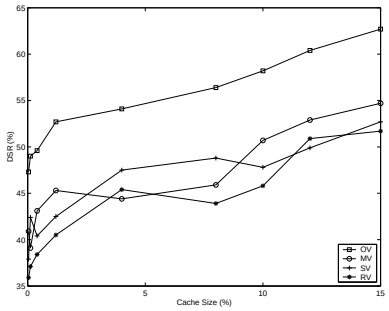

Figure 3.22: Experiment on DSR

and RV, since our solution determines the optimal versions to be cached on the path from the client to the server, while the others place multiple versions of a multimedia object in a heuristic or random way. For ASL to achieve the same performance as OV, the other solutions require 2 to 8 times as much cache size.

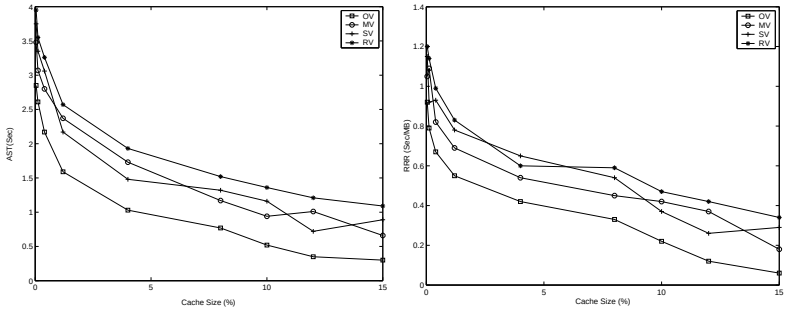

Figure 3.23: Experiment on AST and RRR

Figure 3.24 shows the results of OHR and ASL as a function of the relative cache size for different solutions. By computing the optimal versions to be cached, we can see that our solution produces better results than the others, especially for smaller cache sizes. We can also see that OHR steadily improves as the relative cache size increases, which conforms to the fact that more requests will be satisfied by the caches as the cache size becomes larger. It can also be seen that the ASL for our solution is lower than that for the other solutions. We can also see that ASL decreases as the relative cache size increases.

Impact of Object Access Frequency

This experiment set examines the impact of object access frequency distribution on the performance results of the various solutions. Figure 3.25 shows the performance results of DSR, RRR, and OHR for the values of Zipf parameter α from 0.2 to 1.0.

We can see that OV consistently provides the best performance over a wide range of object access frequency distributions. Specially, CV reduces or improves DSR by 30.4 percent, 24.4 percent, 21.3 percent, and 8.5 percent compared to SV, MV, and RV, respectively; the default cache size used here (4 percent) is fairly large in the context of web caching, due to the large network under consideration.

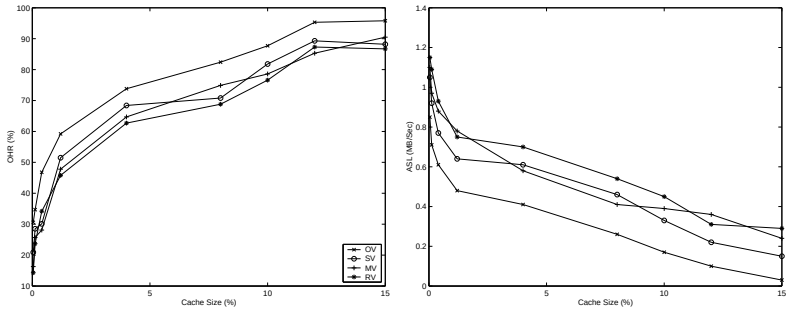

Figure 3.24: Experiment for OHR

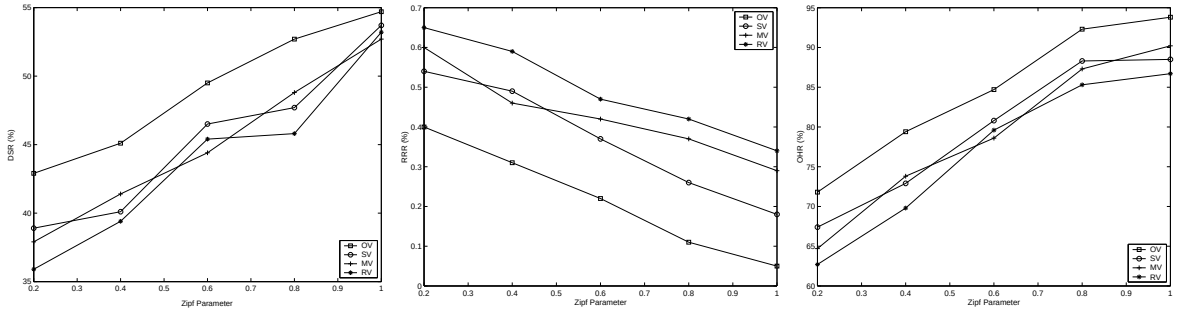

Figure 3.25: Experiment for DSR, RRR, and OHR

3.4 Chapter Summarization

In this chapter, we studied the problem for coordinated en-route transcoding proxy caching (i.e. multimedia object caching in Section 3.1 and transcoding proxy placement in Section 3.2). We also considered the problem of multimedia object placement for transparent data replication in Section 3.3. We conducted a lot of simulation experiments to compare the performance of our models with those proposed in the literature. The contents in this chapter can be found in [54, 58, 59, 61].

Chapter 4

Cache Replacement in Transcoding Proxies

4.1 Preliminary Knowledge

In this section some preliminary knowledge is introduced for later use.

4.1.1 Weighted Transcoding Graph for Simulation

In the simulation, the clients are divided into five classes and the transcoding relationship of the five versions is shown in Figure 4.1.

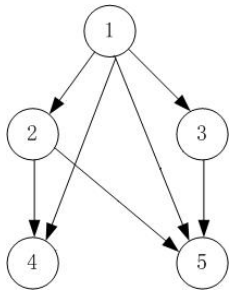

Figure 4.1: Weighted Transcoding Graph for Simulation

4.1.2 Evaluated Algorithms

We include the following algorithms for evaluating our proposed algorithms in the later sections.

- *LRU*: Least Recently Used (LRU) evicts the web object which was requested the least recently. The cache purges one or more least recently requested objects to accommodate the new object if there is not enough room for it.
- *LNC−R [86]*: Least Normalized Cost Replacement ($LNC-R$) is an algorithm that approximates the optimal cache replacement algorithm. It selects for replacement the least profitable documents. The profit function is defined as $profit(O_i) =$

$(c_i \cdot f_i)/s_i$, where c_i is the average delay to fetch document O_i to the cache, f_i is the total number of references to O_i, and s_i is the size of document O_i.

- *AE [23]*: Aggregate Effect (AE) is an algorithm that formulates a generalized profit function to evaluate the aggregate profit from caching multiple versions of an object. The difference between AE and the solution proposed in this section lies in that AE removes the objects from the cache one by one, and our solution removes the objects at the same time by considering the aggregate effect of caching multiple versions of the same object.

In the following figures, LRU, $LNC-R$, and AE denote the results for the algorithms introduced above respectively.

4.2 Cache Replacement for Transcoding Proxy Caching

In this section we present an effective cache replacement algorithm for transcoding proxy caching. In Section 4.2.1, a generalized aggregate cost saving function is defined to determine the rule for evicting the cached objects to make room for a new object if necessary. In Section 4.2.2, we present a cache replacement algorithm and its analysis. The method of estimating the parameter appearing in the algorithm in introduced in Section 4.2.3. Finally, we describe the simulation model and the performance evaluation in Sections 4.2.4 and 4.2.5.

4.2.1 Generalized Cost Saving Function

Let $o_{i,j}$ denote version j of object i and m_i denote the number of different versions of object i. $d_{i,j}$ is the cost of reading or writing $o_{i,j}$ from the server and $\omega_i(j_1, j_2)$ is the transcoding cost from version j_1 to version j_2 of object o_i. $\phi_i(j)$ is the set of all the versions of o_i that can be transcoded from $o_{i,j}$, including $o_{i,j}$ itself. For example, suppose an object o_1 has the transcoding graph shown in Figure 4.2. Then, $\phi_i(1) = \{1, 2, 3, 4, 5\}$,

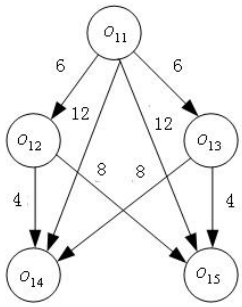

Figure 4.2: The Weighted Transcoding Graph for o_1

$\phi_i(2) = \{2, 4, 5\}$, and $\phi_i(4) = \{4\}$. In this section, we use G_i to denote the weighted transcoding graph for o_i. $\lambda_{i,j}$ is the read rate of $o_{i,j}$ from the server and μ_i is the update rate of $o_{i,1}$.

First we calculate the cost saving of caching only one version of an object (no other versions are cached). From the standpoint of clients, an optimal cache replacement algorithm

should maximize the cost saving from caching multiple copies of objects by considering both the read cost and the update cost. Thus, the individual cost saving of caching only $o_{i,j}$ is defined as follows.

Definition 8 *$CS(o_{i,j})$ is a function for calculating the individual cost saving of caching $o_{i,j}$, while no other versions of object i are cached.*

$$CS(o_{i,j}) = \sum_{x \in \phi_i(j)} \lambda_{i,x}(\omega_i(1,x) + d_{i,x} - \omega_i(j,x)) - \mu_i d_{i,j} \tag{4.1}$$

In Equation (4.1), $\omega_i(1,x)$ and $d_{i,x}$ is the cost for transcoding the original version to version x and sending it to the client, which are saved by caching $o_{i,j}$, and $\omega_i(j,x)$ is the additional cost of transcoding version j to version x at the client, which is needed when caching $o_{i,j}$. $d_{i,j}$ is the additional cost of sending $o_{i,j}$ from the server to the cache upon updates of o_i so that the content of the cached version is consistent with that of the server. Now we give an example of the calculation of the individual cost saving.

Example Consider the scenario shown in Figure 4.2, where only $o_{1,2}$ is cached. Suppose that $\lambda_{1,j} = 1$, $\mu_1 = 1$, and $d_{1,j} = 20$ for all versions of o_1. Based on Definition 8, it is easy to obtain that $CS(o_{1,2}) = 1*(20+6-0)+1*(20+6-4)+1*(20+6-8)-1*20 = 46$ since $\phi_1(2) = \{2,4,5\}$. □

As a matter of fact, there may be many versions of an object that can be cached at the same time if this is beneficial. In the following we discuss the aggregate cost saving of caching multiple versions of an object. We define the aggregate cost saving of caching multiple versions of an object at the same time as below.

Definition 9 *$CS(o_{i,j_1}, o_{i,j_2}, \cdots, o_{i,j_k})$ is a function for calculating the aggregate cost saving of caching o_{i,j_1}, o_{i,j_2}, $\cdots$, o_{i,j_k}.*

$$CS(o_{i,j_1}, o_{i,j_2}, \cdots, o_{i,j_k}) = \sum_{y \in \{j_1, j_2, \cdots, j_k\}} \left[\sum_{x \in \Phi_i(y)} \lambda_{i,x}(\omega(1,x) + d_{i,x} - \omega(y,x)) - \mu_i d_{i,y} \right] \tag{4.2}$$

where $\Phi_i(y)$ is the set of the versions that are transcoded from $o_{i,y}$.

Considering the same conditions as assumed in the example above, we can calculate the aggregate cost saving of caching $o_{i,1}$ and $o_{i,2}$ as $CS(o_{i,1}, o_{i,2}) = 1*(20-0-0)+1*(20+6-6)+1*(20+6-0)+1*(20+6-4)+1*(20+6-8-1)-1*20 = 86$.

If we use $s_{i,j}$ to denote the size of $o_{i,j}$, then we formulate the generalized aggregate cost saving function as follows:

$$CS^G(o_{i,j_1}, o_{i,j_2}, \cdots, o_{i,j_k}) = CS(o_{i,j_1}, o_{i,j_2}, \cdots, o_{i,j_k}) / \sum_{\alpha=1}^{k} s_{i,j_\alpha} \tag{4.3}$$

It is easy to see that the generalized aggregate cost saving function is further normalized by the total size of $o_{i,j_1}, o_{i,j_2}, \cdots, o_{i,j_k}$ to reflect the object size factor. The rationale behind this normalization is to order the objects by the ratio of aggregate cost saving to their total object size. The generalized aggregate cost saving function defined in Equation (4.3) explicitly takes into consideration the new emerging factors in the environment of transcoding proxies. Importantly, it also takes cache consistency into account.

4.2.2 A Cache Replacement Algorithm

In this section we propose an effective cache replacement algorithm for transcoding proxy caching based on the generalized aggregate cost saving function defined in Section 4.2.1. Suppose that there are l different multimedia objects cached and the size of a new object to be cached is s, then we should find a subset of objects $O^* \subseteq O$ that satisfies the following conditions.

(1) $\sum_{o_{i,j} \in O^*} s_{i,j} \geq s$,

(2) $\forall O' \subseteq O$ s.t. $\sum_{o_{i,j} \in O'} s_{i,j} \geq s$: $CS^G(O^*) \leq CS^G(O')$,

where $O^* = \{o_{1,\alpha_1^1}, \cdots, o_{1,\alpha_1^{r_1}}, \cdots, o_{l,\alpha_l^1}, \cdots, o_{l,\alpha_l^{r_l}}\}$ is the set of objects to be removed, $O = \{o_{1,\beta_1^1}, \cdots, o_{1,\beta_1^{c_1}}, \cdots, o_{l,\beta_l^1}, \cdots, o_{l,\beta_l^{c_l}}\}$ is the set of objects cached, and $CS^G(O^*) = \sum_{i=1}^{l} CS^G(o_{i,\alpha_i^1}, \cdots, o_{i,\alpha_i^{r_i}})$. $CS^G(O')$ can be similarly defined. Obviously, (1) is to make enough room for the new object, and (2) is to evict those objects whose generalized aggregate cost saving is minimal.

The naive approach to find such O^* will be in NP hard, same as the packing problem. In the following, we present an algorithm that computes an approximate answer of the problem efficiently by decomposing the set of the candidate objects to be removed into smaller sets and each such set can be decided in polynomial time.

Before we present the algorithm, we introduce some notations. In the following, let $R^*(i,k)$ denote the minimal generalized aggregate cost saving of caching k versions of object i and $R^*(k)$ the minimal generalized aggregate cost saving of the k objects to be removed. We can see that the k objects to be removed can be k versions of a multimedia object or different versions of different multimedia objects. Thus, k can be decomposed as $k = k_1 + k_2 + \cdots + k_a$, where a is the number of different objects to be removed and $0 \leq k_i \leq k$ is the number of versions of an object that are in the set of the k objects to be removed. For example, $1 \rightarrow \{1\}$, $2 \rightarrow \{2, 1+1\}$, $3 \rightarrow \{3, 2+1, 1+1+1\}$, $4 \rightarrow \{4, 3+1, 2+2, 2+1+1, 1+1+1+1\}$, $5 \rightarrow \{5, 4+1, 3+1+1, 3+2, 2+1+1+1, 2+2+1, 1+1+1+1+1\}$, $\cdots$. For the instance of $k = 4$, k can be the combination of $1+1+1+1$, $1+1+2$, $2+2$, $1+3$, and 4, where $1+1+1+1$ means that the objects to be removed should be the first four objects with minimal generalized aggregate cost savings of caching one version, $1+1+2$ means that the objects to be removed should be the three objects, i.e., the first two objects with minimal generalized aggregate cost savings of caching one version and the last object with minimal generalized aggregate cost saving of caching two versions, etc. It can be easily proved that there are at most k^2 different such combinations in all. Therefore, we have

$$R^*(k) = \min\{R^*(1,k), R^*(2,k), \cdots, R^*(l,k), \min_{k=k_1+k_2+\cdots+k_a} \{R^*(k_1) + R^*(k_2) + \cdots + R^*(k_a)\}\}$$

We denote the set of all the objects that achieves $R^*(k)$ by $O^*(k)$ and their total size is $S^*(k)$. Now we give an example to show how to calculate $R^*(k)$. For the case of $k = 3$, we

have the combination of 3, $1+2$, and $1+1+1$, each of which can be computed using the previous calculation results. For $3+0$, we just choose three versions from one object with minimal generalized aggregate cost savings of caching three versions. For $1+2$, we choose the version from an object with minimal generalized aggregate cost savings of caching one version and two versions from another object with minimal generalized aggregate cost savings of caching two versions. When we calculated $R^*(1)$ and $R^*(2)$, they may be using a same version. In this case, we select another version with minimal generalized cost saving that is not included. We denote the set of versions calculated by $R^*(1)$ and $R^*(2)$ as $O^*(1)$ and $O^*(2)$, respectively. In this case we will recalculate the set of versions with minimal number of elements by another set of versions of the same object with the same number of elements with more generalized aggregate cost saving. For example, if $o_{1,1} \in O^*(1)$ and $o_{1,1} \in O^*(2)$, then we will recalculate $O^*(1)$, i.e., finding $o_{1,j}$ with the minimal generalized aggregate cost saving except $o_{1,1}$ to represent $o_{1,1}$. Although this will be very costly in theory, the fact that the number of objects we hope to remove in practice is very small makes it feasible. We shall further study this issue in our future work. Based on the above calculation, we finally find how the k objects should be selected such that the generalized aggregate cost saving is minimized. In fact, there may exist a replacement decision by removing more than k objects and the generalized aggregate cost saving is less. Thus, the minimization here is conditional, i.e., under the condition that the minimal number of different objects is to be removed.

With the above analysis, we can devise the pseudocode of our algorithm as follows. In the algorithm, C is used to hold the cached objects, S_c is the cache capacity, S_u is the cache capacity used, o is the object to be cached, and its size is s.

Algorithm MOR (S_c, S_u, s)
Input: S_c, S_u, s
Output: $O^*(n)$
1. INSERT o INTO C
2. $n = 0$
3. $S^*(n) = 0$
4. WHILE $S_c - S_u - S^*(n) < s$ DO
5. $n = n + 1$
6. FOR $i = 1$ TO l DO
7. CALCULATE $R^*(i, n)$
8. CALCULATE $R^*(n)$
9. CHECK $O^*(n)$ (make all the n objects different)

Regarding to the time complexity of this algorithm, we have the following theorem.

Theorem 24 *The time complexity of Algorithm MOR is $O(p(k)(l+p(k))\log(l+p(k)))$, where l is the total number of different objects cached, $p(k)$ is the number of partitions of k, and k is the number of versions to be removed.*

Proof Suppose k objects are removed to make room for the new object. The running time of Algorithm MOR mainly depends on Steps 4, 6, 8, and 9. The running time of Step 6 is determined by computing $R^*(i, n)$ for $1 \leq i \leq l$. For object i, calculating $R^*(i, n)$ is to find the minimal generalized aggregate cost saving of caching n versions

of object i. Note that we should compute the aggregate profit of caching n versions of object i, and then order them according to the calculated profit. Thus, the running time for calculating $R^*(i,n)$ is $O(C(m_i;n)\log C(m_i;n))$. Therefore, The running time of Step 6 is $O(\sum_{i=1}^{l} C(m_i;n)\log C(m_i;n))$ since there are l objects cached and $C(m_i;n) = m_i!/(n!(m_i-n)!)$. The running time for Step 8 is $O((l+n)\log(l+n))$ because we should order all $l+n$ items to find the minimal one among them. Thus, the total running time for Algorithm MOR (Step 4) is $O(\sum_{n=1}^{p(k)}[(l+n^2)\log(l+n^2)+\sum_{i=1}^{l} C(m_i;n)\log C(m_i;n)]) = O(p(k)(l+p(k))\log(l+p(k)))$ since in general $m_i \ll l$. Since the running time for Step 9 is $O(\log l)$, the total running time for Algorithm MOR is $O(p(k)(l+p(k))\log(l+p(k)))$. Hence, the theorem is proven. □

From Theorem 24, we know that the time complexity of Algorithm MOR depends on k, i.e., the number of objects to be removed. In practical execution, we always stop the execution of searching the objects to be removed to make room for the new object when k reaches a certain number. This is based on the fact that it is not beneficial to remove many objects to accommodate only one object. So the practical time complexity of Algorithm MOR is $O(l\log l)$ since $p(k) \ll l$, which is the same as that of the algorithm proposed in [23]. However, from the algorithm we know that we have to search the entire cache for the other versions of the object and then recalculate the generalized aggregate cost savings for them whenever we insert or evict an object into or from the cache. Such operations are, in general, very costly. Here, we save calculated results for later computation, which will save a lot of computations. For example, after we finish computing $R^*(n)$, we save it using an array. When we hope to compute $R^*(n+1)$, we do not need to recalculate $R^*(k)$ for $1 \le k \le n$ again by reading it from the array directly.

4.2.3 Parameter Estimation

In the actual implementation, the parameters, such as $d_{i,j}$, $\lambda_{i,j}$, and μ_i, for computing the generalized aggregate cost saving are usually not constant. To realize our algorithm, these parameters may have to be estimated. Here, we adopt a "sliding window" technique [88] which has been widely applied [106]. It combines both the history data and the current value to estimate the parameters. Specifically, the parameters are estimated as follow.

$$\begin{aligned} d_{i,j} &= \alpha \cdot d_{i,j}^{new} + (1-\alpha)\cdot d_{i,j}^{old} \\ \lambda_{i,j} &= \frac{K_1}{t_{i,j}-t_{i,j}^{K_1}} \\ \mu_i &= \frac{K_2}{s_{i,1}-s_{i,1}^{K_2}} \end{aligned} \quad .$$

where $d_{i,j}^{new}$ is the newly measured cost of reading or writing $o_{i,j}$ from the client or the server and $d_{i,j}^{old}$ is the measured cost of reading or writing $o_{i,j}$ from the client or the server last time; $t_{i,j}$ is the time when the new request to $o_{i,j}$ is received from the client and $t_{i,j}^{K_1}$ is the time when the last K_1 request is received from the client; $s_{i,j}$ is the time when the new update to $o_{i,j}$ is sent from the server and $s_{i,j}^{K_2}$ is the time when the last K_2 update is sent from the server.

The effect of such estimates on the performance of our algorithm depends on the selection of the fining-tuning knobs: α, K_1, and K_2. The knob α determines how fast

$d_{i,j}$ adapts to the most recent sample, while the knobs K_1 and K_2 determine how many samples should be used to estimate $\lambda_{i,j}$ and μ_i. Obviously, the larger the value of K_1 and K_2, the more reliable the estimation of $\lambda_{i,j}$ and μ_i. However, large value of K_1 and K_2 will generate spatial cost to store the relevant data. In [88], the authors showed that the best performance could be achieved by small values of K_1 and K_2 and they further conjectured that such small values of K_1 and K_2 could be 2 or 3. Thus, the spatial overhead is relatively small.

4.2.4 Simulation Model

We have performed extensive simulation experiments to compare the performance of our algorithm with existing cache replacement algorithms. In the simulation, to generate the workload of clients' requests, we model a single server that maintains a collection of m multimedia objects[1]. The object popularity followed a Zipf-like distribution [11]. Specifically, the popularity of the ith video was proportional to $1/i^{\alpha}$. The default values of m and α were set to be 1000 and 0.75 respectively. The sizes of the videos followed a heavy tailed distribution with the mean value of $12K$ Bytes [74]. The clients are divided into five classes and we assume that the sizes of the five versions of each video are 100 percent, 80 percent, 60 percent, 40 percent, and 20 percent of the original video size. The access probabilities of the clients are described as a vector of $< 0.2, 0.15, 0.3, 0.2, 0.15 >$. The transcoding relationship of the five versions is shown in Figure 4.1 in Section 4.1.1.

Regarding the transcoding rate, we set it, as in [18], to be $20K$ bytes per second. The delays of fetching the videos from the server are given by an exponential distribution. We assume that there is no correlation between the video size and the delay of fetching it from the server. This is justified by Shim et al. in [88].

The synthetic workloads are generated according to the recent results on the web workload characterization [28,39,74]. Table 4.1 lists the parameters and their values used in the simulation.

Table 4.1: Parameters Used in Our Simulation

Parameter	Value
Number of Multimedia Objects	1000 objects
Delay of Fetching Objects	Exponential Distribution $p(x) = \theta^{-1}e^{-x/\theta}$ $(\theta = 0.45\ Sec)$
Web Object Size Distribution	Pareto Distribution $p(x) = \frac{ab^a}{a-1}$ $(a = 1.1, b = 8596)$
Web Object Access Frequency	Zipf-Like Distribution $\frac{1}{i^{\alpha}}$ $(i = 0.7)$
Transcoding Rate	$20KB/Sec$

[1]In the simulation, the multimedia objects are assumed to be videos.

4.2.5 Performance Evaluation

In this section, we compare the performance of our algorithm with those algorithms introduced in Section 4.2.4 in terms of several performance metrics. The main performance metrics employed in the simulation include: delay-saving ratio (DSR), request response ratio (RRR), object hit ratio (OHR), and staleness ratio (SR), defined as a fraction of cache hits which return stale objects. Here "stale" means that the time that an object was brought to the cache is less than the last-modified timestamp corresponding to the request. In the following figures OA denotes the results for the algorithm proposed in Section 4.2.2.

- **Without Considering Cache Consistency**

In this following, we compare the performance of the four algorithms without considering the issue of cache consistency.

In the first experiment set, we compare the performance of different algorithms across a wide range of cache sizes, from 0.04 percent to 15.0 percent of the total size of all the objects as shown in Figure 4.3.

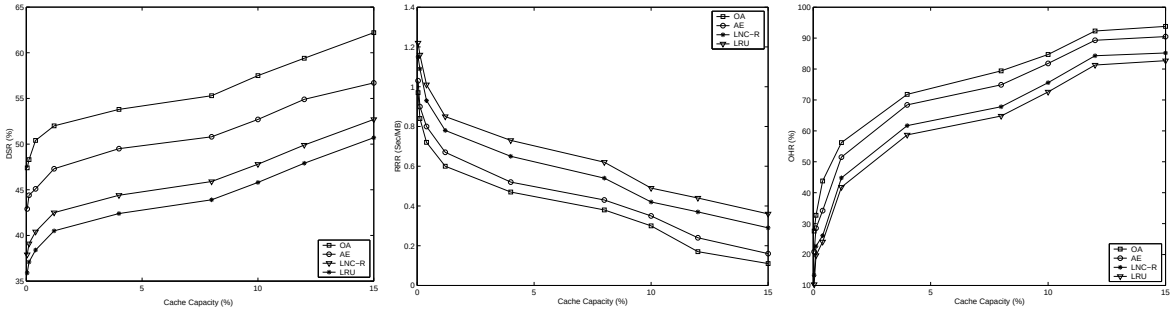

Figure 4.3: Experiment on DSR, RRR, and OHR

The first experiment investigates DSR as a function of the relative cache size. As presented in Figure 4.3, we can see that our algorithm outperforms the others. Specifically, the mean improvements of DSR over LRU, $LNC-R$, and AE are 9.5 percent, 21.6 percent, and 23.5 percent, respectively.

In the second experiment, we describe the results of RRR as a function of the relative cache size. Clearly, the lower the RRR, the better the performance. As we can see, all algorithms provide steady performance improvement as the cache size increases. We can also see that OA constantly improves RRR compared to AE, $LNC-R$ and LRU, since our algorithm determines the objects with minimal generalized aggregate cost saving to be removed, while the others do not satisfy such criterion. For RRR to achieve the same performance as OA, the other algorithms need 1.4 to 5 times as much cache size.

The third experiment investigates the results of OHR as a function of the relative cache size for different algorithms. By computing the objects with minimal generalized aggregate cost saving to be removed, we can see that the results for our algorithm constantly outperforms those of the others, especially for smaller cache sizes. We can also see that OHR steadily improves as the relative cache size increases, which conforms to the fact that more requests will be satisfied by the caches as the cache size becomes larger.

This experiment set examines the impact of object access frequency distribution on the performance of different algorithms. Figure 4.4 shows the performance results of DSR, RRR, and OHR for the values of Zipf parameter α from 0.2 to 1.0.

We can see that OA consistently provides the best performance over a wide range of object access frequency distributions. Specially, OA reduces or improves DSR by 25 percent, 21 percent, and 11 percent compared to LRU, by 18 percent, 15 percent, and 7 percent compared to $LNC-R$, and by 15 percent, 10 percent, and 5 percent compared to AE for Zipf parameters of 0.2, 0.6, and 1.0, respectively; the default cache size used here (4 percent) is fairly large in the context of caching due to the large network under consideration (e.g. that of a regional ISP).

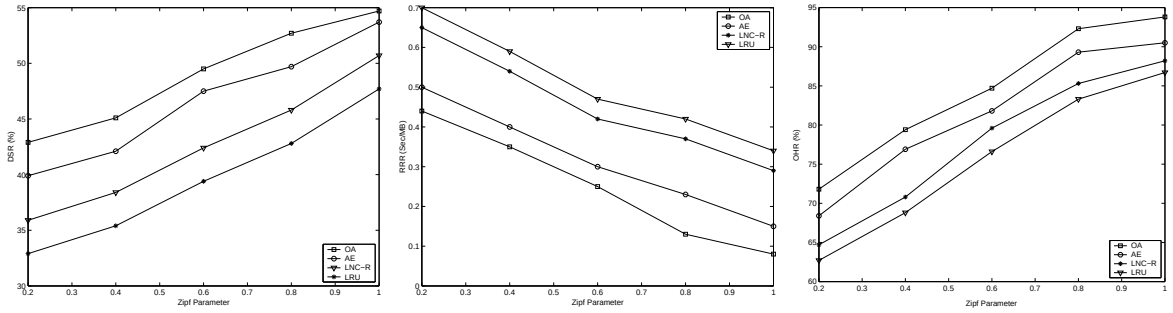

Figure 4.4: Experiment for DSR, RRR, and OHR

- **With Considering Cache Consistency**

In this following, we compare the performance of the four algorithms with considering the issue of cache consistency.

In the first experiment set, we compare the performance of different algorithms across a wide range of cache sizes, from 0.04 percent to 15.0 percent of the total size of all the objects as shown in Figure 4.5.

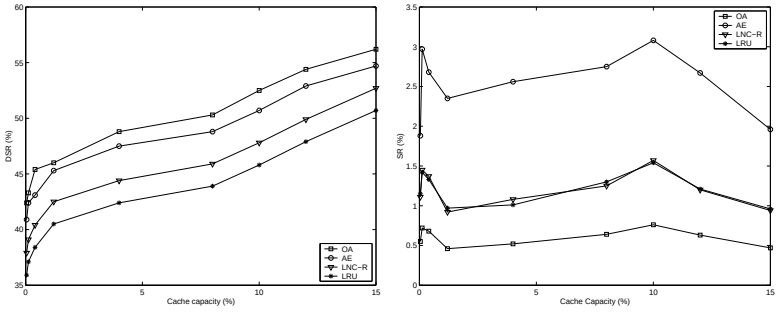

Figure 4.5: Experiment on DSR and SR

The first experiment investigates DSR as a function of the relative cache size. OA gives on average 13.4% improvement over LRU, 9.8% over $LNC-R$, and 3.7% over AE. The maximal improvements over LRU, $LNC-R$, and AE are 4.1% 10.8%, and 14.5%. From Figures 4.3 and 4.5, we can see that whether considering cache consistency or not will affect DSR. Since LRU and $LNC-R$ involves some form of consistency, there is less influence on them. However, OA still exhibits the best performance. The

second experiment studies SR as a function of the relative cache size. In addition to improving performance of the cache, OA also constantly reduces stale ratio. On average, OA achieves a staleness ratio which is by factor of 3.2 better than that of AE, in the worst case it improves SR of AE by factor of 1.9 when the cache size is 0.5%. OA also improves SR over LRU and $LNC-R$. On average, OA achieves a staleness ratio which is 50.8% better than that of LRU and 50.1% better than that of $LNC-R$. In the worst case, it improves SR of LRU by 10.2% when the cache size is 0.5% and improves SR of $LNC-R$ by 8% when the cache size is 2.0%.

In the second experiment set, we examine the performance of our algorithm by varying the parameter α of the Zipf-like distribution, where α changes from 0.2 to 1.0. Figure 4.6 shows the performance results of DSR and SR.

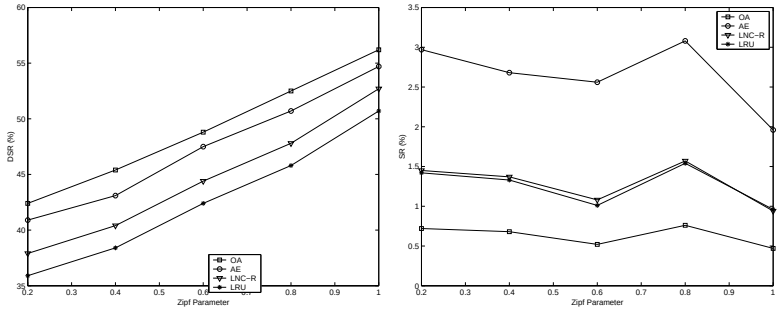

Figure 4.6: Experiment for DSR and SR

4.3 Coordinated Cache Replacement in Transcoding Proxies

4.3.1 Notations and Definitions

We model the network as a graph $G = (V, E)$ in this paper, where $V = \{v_0, v_1, \cdots, v_n\}$ is the set of nodes or vertices, and E is the set of edges or links. We assume that every node is associated with a cache with the same size B and there are m multimedia objects, i.e., $O_1, O_2, \cdots, O_l$, maintained by server v_0. For each multimedia object O_j, we assume that it has m_j versions: $O_{j,1}, O_{j,2}, \cdots, O_{j,m_j}$ and all versions have the same size. Thus, each node can hold at most B objects. We denote the set of objects cached at node v_i by $Y^i = \{A_1^i, A_2^i, \cdots, A_m^i\}$, where $A_j^i \subseteq \{O_{j,k_1}, O_{j,k2}, \cdots, O_{j,k_j}\}$ is the set of different versions of object O_j cached at node v_i. Obviously, $Y = \{Y^1, Y^2, \cdots, Y^n\}$ is the set of all objects cached. For each version of object O_j, we associate each link $(u, v) \in E$ a nonnegative cost $L_{j,k}(u, v)$, which is defined as the cost of sending a request for version $O_{j,k}$ and the relevant response over the link (u, v). In particular, $L_{j,k}(u, u) = 0$. If a request goes through multiple network links, the cost is the sum of the cost on all these links. The cost in our analysis is calculated from a general point of view. It can be different performance measures such as delay, bandwidth requirement, and access latency, or a combination of these measures. Let $r_{i,j,k}$ denote the request for $O_{j,k}$ at node v_i and $f_{i,j,k}$ be the frequency of $r_{i,j,k}$.

For notational tidiness, we omit argument j in all parameters and functions throughout the following analysis since our analysis is based on a specific object. For example, O_k

denotes version k of object j, A^i is the set of different versions of object j cached at node v_i, $L_k(u,v)$ denotes the cost of sending a request for version O_k and the relevant response over the link (u,v), $r_{i,k}$ denotes the request for O_k at node v_i, and $f_{i,k}$ denotes the frequency of $r_{i,k}$. We also make the following assumptions.

- *Assumption* 1*:* $L_k(v_{i_1}, v_{i_2}) = (i_1 - i_2)L$ for all $1 \leq k \leq m$ as there are $i_1 - i_2$ links on the path between node v_{i_1} and node v_{i_2}, and the cost on each link for each version of O_j is L.
- *Assumption* 2*:* The transcoding graph is a linear array and the transcoding cost between any two adjacent versions is constant, i.e., $t(O_{k_1}, O_{k_2}) = \sum_{k=k_1}^{k_2-1} t(O_k, O_{k+1}) = (k_2 - k_1)^+ T$, where $x^+ = x$ if $x \geq 0$ else $x^+ = \infty$.
- *Assumption* 3*:* There exists some positive integer δ such that $(\delta - 1)T \leq L$, and $\delta T > L$. If there does not exist such a δ, i.e., $L \gg T$ or $T \gg L$. Obviously, these are two trivial cases.

4.3.2 Problem Formulation

Before formulating the problem, we give some explanation on how the requests are served. As shown in Figure 4.7, a request goes along a routing path from the client (node v_n) to the server (node v_0). Note that any request $r_{i,k}$ could find the service from $S(r_{i,k})$, where $S(r_{i,k})$ denotes the serving object for $r_{i,k}$. Assume that $S(r_{i,k}) = O_{k_1} \in A^{i_1}$ with $k_1 \leq k$ and $i_1 \leq i$, then there may be the following ways of serving $r_{i,k}$ by $O_{k_1} \in A^{i_1}$.

- O_{k_1} is first sent from node v_{i_1} to node v_i and then transcoded to O_k at node v_i.
- O_{k_1} is first transcoded to O_k at node v_{i_1} and then O_k is sent from node v_{i_1} to node v_i.
- O_{k_1} is first sent from node v_{i_1} to node v_{i_2}, transcoded to O_k at node v_{i_2}, and then then O_k is sent from node v_{i_2} to node v_i.
- O_{k_1} is first sent from node v_{i_1} to node v_{i_2} and transcoded to O_{k_2} at node v_{i_2}, and then O_{k_2} is sent from node v_{i_2} to node v_{i_3} and transcoded to O_{k_3} at node v_{i_3}, then O_{k_3} is sent from node v_{i_3} to node v_i and transcoded to O_k at node v_i.
- $\vdots$

Figure 4.7: System Model for Multimedia Object Caching

All these cases would cost the same under our cost model even though in practice. However, when a new or updated version of a multimedia object to be cache, denoted by O_{i_0}, is passing through each node between nodes $v_{i'}$ and v_i, it should be decided where

O_{i_0} should be cached and which version should removed from the relevant cache to make room for it depending on how $r_{i,k}$ is served. Given X (i.e., the set of cached objects) and $O_{k'} \in A^{i'}$ $(i' \leq i)$. Let $d(r_{i,k}, O_{k'})$ denote the cost of serving $r_{i,k}$ by $O_{k'}$ at node $v_{i'}$. Then $d(r_{i,k}, O_{k'})$ is defined as follows:

$$d(r_{i,k}, O_{k'}) = (i - i')L + (k - k')^+ T \tag{4.4}$$

where $(x-y)^+ = \begin{cases} x - y & \text{if } x - y \geq 0 \\ 0 & \text{if } x - y < 0 \end{cases}$

Now we begin to formulate the problem addressed in this paper, i.e., determining where a new or updated version O_{i_0} should be cached among nodes $\{v_1, v_2, \cdots, v_n\}$ and which version of object j should be removed at that node to make room for O_{i_0} such that the total cost loss is minimized. Suppose that $P \subseteq V$ is the set of nodes at each of which $X_{i,k_i} \in A^i$ should be removed to make room for O_{i_0}, then this problem can be formally defined as follows:

$$L(P^*) = \min_{P \subseteq V} \{L(P)\} = \sum_{v_i \in P} (l(X_{i,k_i}) - g_i(O_{i_0})) \tag{4.5}$$

where $L(P)$ is the total relative cost loss, $l(X_{i,k_i})$ is the cost loss of removing X_{i,k_i} from node v_i, and $g_i(O_{i_0})$ is the cost saving of caching O_{i_0} at node v_i.

4.3.3 Dynamic Programming-based Solution

Before presenting the solution, we evaluate the two items, i.e., $l(X_{i,k_i})$ and $g_i(O_{i_0})$, shown in Equation (4.5) in detail .

First, we begin with presenting a solution for finding the best way of serving $r_{i,k}$, i.e., finding $S(r_{i,k})$. Based on Equation (4.4), the cost of serving $r_{i,k}$, denoted by $c(r_{i,k})$, is defined as follows:

$$c(r_{i,k}) = \min \left\{ \min_{O_{k'} \in A^{i'},\ 1 \leq i' \leq i} d(r_{i,k}, O_{k'}), iL \right\} \tag{4.6}$$

Therefore, the object for serving $r_{i,k}$, denoted by $S(r_{i,k})$, is determined as follows:

$$S(r_{i,k}) = \begin{cases} O_{k'} \in A^{i'} & \text{if } c(r_{i,k}) \geq d(r_{i,k}, O_{k'}) \\ v_0 & \text{if } c(r_{i,k}) = iL \end{cases} \tag{4.7}$$

The following property will help us simplify the problem of finding the best way of serving $r_{i,k}$.

Theorem 25 *If both O_{k_1} and O_{k_2} are cached at node $v_{i'}$, then we have $d(r_{i,k}, O_{k_1}) < d(r_{i,k}, O_{k_2})$ for $k > k_1 > k_2$.*

Proof Based on the definition of $d(r_{i,k}, O_k)$, we have $d(r_{i,k}, O_{k_1}) = (i - i')L + (k - k_1)^+ T$ and $d(r_{i,k}, O_{k_2}) = (i - i')L + (k - k_2)^+ T$. Since $(k - k_1)^+ < (k - k_2)^+$, we have $d(r_{i,k}, O_{k_1}) < d(r_{i,k}, O_{k_2})$. Hence, the theorem is proven. □

From Theorem 25, we can see that for request $r_{i,k}$, we can consider only the least detailed version that can be transcoded to version k. Thus, Equation (4.6) can be simplified as follows:

$$c(r_{i,k}) = \min\left\{\min_{1\leq i^{'}\leq i} d(r_{i,k}, O_{k^*}), iL\right\} \tag{4.8}$$

where O_{k^*} is the least detailed version of object j cached at node $v_{i'}$ that can be transcoded to version k.

It is easy to see that the time complexity for computing $S(r_{i,k})$ is $O(\log n)$, where n is the number of nodes in the network. So the total complexity for computing all $S(r_{i,k})$ ($1 \leq i \leq n$ and $1 \leq k \leq m$) is $O(mn \log n)$ since there are n nodes and object j has m different versions.

For each object $x \in X$, the set of requests served by x is expressed as $R(x) = \{r_{i,k}|S(r_{i,k}) = x\}$ and the total cost for the requests served by x is $C(x) = \sum_{r_{i,k}\in R(x)} f_{i,k}d(r_{i,k}, x)$. In this paper, we use R_s to denote the set of requests served by the server.

Regarding to $R(x)$, we have the following property.

Property 1 *If $r_{i,k} \in R(x)$, then $r_{i^{'},k^{'}} \in R(x^{'})$ $\forall$ $i^{'} \leq i$ and $k^{'} \leq k$.*

Proof Suppose that $x \in A^{i_1} = O_{k_1}$, $x^{'} \in A^{i_2} = O_{k_2}$ and there exists $i^{'} \leq i$ and $k^{'} \leq k$ such that $r_{i^{'},k^{'}} \in R(O_{i_2})$. Since $S(r_{i^{'},k^{'}}) = x^{'}$, we have $d(r_{i^{'},k^{'}}, x^{'}) \leq d(r_{i^{'},k^{'}}, x)$. Therefore we have $(i^{'}-i_2)L+(k^{'}-k_2)T \leq (i^{'}-i_1)L+(k^{'}-k_1)T$, i.e., $(i_2-i_1)L+(k_2-k_1)T \leq 0$. Therefore we have $d(r_{i,k}, x) = (i-i_1)L+(k-k_1)T = (i-i_2)L+(k-k_2)T+(i_2-i_1)L+(k_2-k_1)T = d(r_{i,k}, x^{'})+(i_2-i_1)L+(k_2-k_1)T \leq d(r_{i,k}, x^{'})$. So we have $S(r_{i,k}) = x^{'}$, which contradicts $r_{i,k} \in R(x)$. Hence, the property is proven. □

From Property 1, we can see that $R(x)$ should be a region that can be divided into several rectangular regions. This can be seen from Figure 4.8. For example, $R(x_4)$ can be divided into two regions by the vertical broken line from x_2.

Regarding to calculating $l(X_{i,k_i})$, we first give the following theorem.

Theorem 26 *Suppose that only X_{i,k_i} is cached at node v_i, then we have $l(X_{i,k_i}) = \sum_{r_{i,k}\in B_0} f_{i,k}[i\cdot L-d(r_{i,k}, X_{i,k_i})]+\sum_{i=1}^{n}\sum_{r_{i,k}\in B_i} f_{i,k}[d(r_{i,k}, X^i_{k_i})-d(r_{i,k}, X_{i,k_i})]$, where $B_0 = \{(\alpha,\beta)|\alpha = i_0, \beta \in R_0 \bigcap R(X_{i,k_i})\} \bigcap R(X_{i,k_i})$ and $B_i = \{(\alpha,\beta)|\alpha = i_0, \beta \in R(X^i_{k_i}) \bigcap R(X_{i,k_i})\} \bigcap R(X_{i,k_i})$.*

Proof It is obvious that $B_i \bigcap B_j = \phi$ for $i \neq j$. This guarantees that each request's access cost is only calculated one time. Now we prove the correctness of the calculation of $l(X_{i,k_i})$, i.e., the requests in B_i should be served by $X^i_{k_i}$. Suppose that there exists a request $r_{i^{'},k^{'}} \in b_i$ which is not served by $X^i_{k_i}$. Based on Property 1, we have all the requests in the region $B^{'}_i = \{(\alpha,\beta)|i \leq \alpha \leq i_0, k_i \leq \beta \leq k_0$ will be not served by $X^i_{k_i}$. It is easy to see that $R(X^i_{k_i}) \bigcap B^{'}_i \neq \phi$, i.e., there exist some requests in region $R(X^i_{k_i})$ that are not served by $X^i_{k_i}$. This obviously contradicts the fact that all the requests in region $R(X^i_{k_i})$ are served by $X^i_{k_i}$. Hence, the theorem is proven. □

For example, in Figure 4.8, if x_1 is removed, $R(x_1)$ can be divided into three regions (i.e., A, B, and C), which will be served by x_4, x_3, and the server, respectively. Thus, we

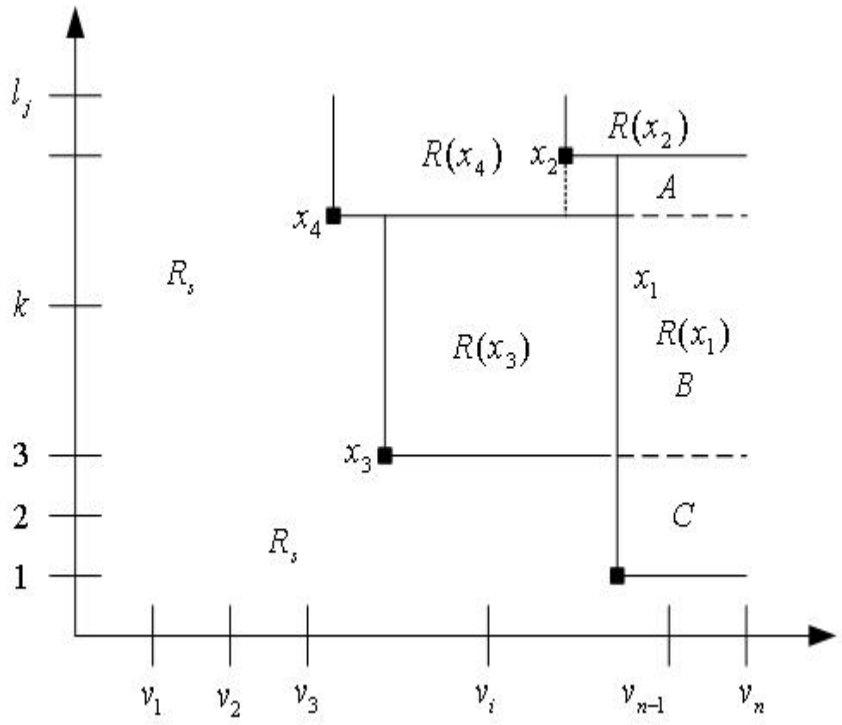

Figure 4.8: Example for Calculating $l(x)$

have $l(x_1) = \sum_{r_{i,k} \in A} f_{i,k}[d(r_{i,k}, x_4) - d(r_{i,k}, x_1)] + \sum_{r_{i,k} \in B} f_{i,k}[d(r_{i,k}, x_3) - d(r_{i,k}, x_1)] + \sum_{r_{i,k} \in B} f_{i,k}[i \cdot L - d(r_{i,k}, x_1)]$.

In practice, the general case is that several versions of the same multimedia object are cached at node v_i at the same time (see Figure 4.9). In this case, calculating $l(x)$ should also consider the mutual effect of the least more detailed cached version on the removed version since the requests served by the removed version could be satisfied by this detailed version. For example, when calculating $l(x_2)$, $R(x_2)$ might be divided into four parts A, B, C, and D which will be served by x_4, x_5, x_3, and x_1, respectively.

Taking into consideration the caching dependence along the path, calculating $l(X_{i,k_i})$ becomes more complex and it is so obvious to obtain an optimal solution.

Similarly, we can calculate the cost saving of caching O_{i_0} at node v_i. For example in Figure 4.9, if $i_0 = y_1$, then $R(x_6$ can be divided in to two parts: E and F; if $i_0 = y_2$, then $R(x_6$ can also be divided in to two parts. So we have $g(y_1) = \sum_{r_{i,k} \in G} f_{i,k}[d(r_{i,k}, y_1) - d(r_{i,k}, x_6)]$ and $g(y_2) = \sum_{r_{i,k} \in E} f_{i,k}[d(r_{i,k}, x_6) - d(r_{i,k}, y_1)] + \sum_{r_{i,k} \in F} f_{i,k}[i \cdot L - d(r_{i,k}, y_1)]$.

Now we begin to present an optimal solution for the problem as defined in Equation 4.5. In the following, we call the problem a k-optimization problem if we determine cache replacement candidates from nodes $\{v_1, v_2, \cdots, v_k\}$. Thus, the original problem (Equation (4.5)) is an n-optimization problem. Theorem 27 shows an important property that the optimal solution for the whole problem must contain optimal solutions for some subproblems.

Theorem 27 *Suppose that* $X = \{X_{i_1,k_{i_1}}, X_{i_2,k_{i_2}}, \cdots, X_{i_\alpha,k_{i_\alpha}}\}$ *is an optimal solution for*

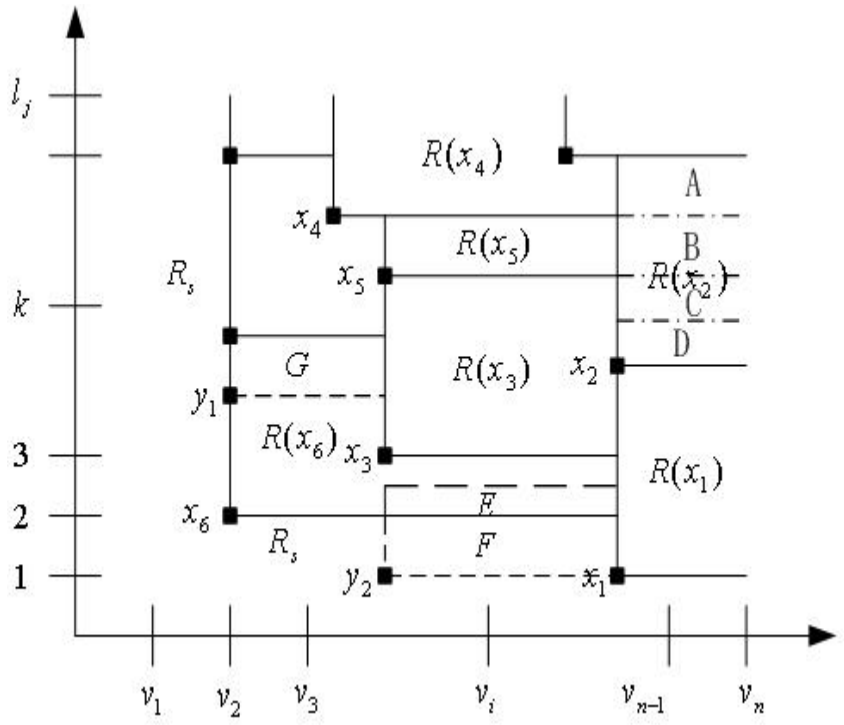

Figure 4.9: Example for Calculating $l(x)$

the α-optimization problem and $X' = \left\{ X_{i'_1,k_{i'_1}}, X_{i'_2,k_{i'_2}}, \cdots, X_{i'_\beta,k_{i'_\beta}} \right\}$ *is an optimal solution for the* $k_{i_\alpha} - 1$*-optimization problem. Then* $X^* = \left\{ X_{i'_1,k_{i'_1}}, X_{i'_2,k_{i'_2}}, \cdots, X_{i'_\beta,k_{i'_\beta}}, X_{i_\alpha,k_{i_\alpha}} \right\}$ *is also an optimal solution for the α-optimization problem.*

Proof By definition, we first have $L(X^*) = l(X_{i'_1,k_{i'_1}}) + l(X_{i'_2,k_{i'_2}}) + \cdots + l(X_{i'_\beta,k_{i'_\beta}}) + l(X_{i_\alpha,k_{i_\alpha}}) = L(X') + l(X_{i_\alpha,k_{i_\alpha}}) \geq l(X_{i_1,k_{i_1}}) + l(X_{i_2,k_{i_2}}) + \cdots + l(X_{i_\beta,k_{i_\beta}}) + l(X_{i_\alpha,k_{i_\alpha}}) = L(X)$. On the other hand, since X is an optimal solution for the α-optimization problem, we have $L(X) \geq L(X^*)$. Therefore,we have $L(X) = L(X^*)$. Hence, the theorem is proven. □

Based on Theorem 27, an optimal solution for the n-optimization can be obtained by checking all possible removed candidates from node v_1 to node v_n in order. Therefore, it is east to get that the time complexity of this solution is $O(n^2 + mn \log n)$ based on our previous result that the complexity for computing all $S(r_{i,k})$ is $O(mn \log n)$, where n is the number of nodes in the network and m is the number of versions of object j.

4.3.4 Cooperative Cache Replacement Scheme

Based on the previous analysis, we present the following cooperative cache replacement scheme. In our scheme, every cache maintains some information about the objects in the form of object descriptors. An object descriptor contains information that includes the object size and the access frequencies for all versions of the multimedia object. When

an updated version of a multimedia object is to be cached, it should be cached at those nodes where the cache replacement candidates has been calculated by our solution.

Since the cache contents change over time, the access frequency and the cost loss of an object with respect to a node must be refreshed from time to time. The access frequency can be estimated based on recent request history, which is locally available (e.g. by using a "sliding window" technique [88]). The cost loss is updated by the response messages. Specifically, a variable with an initial value of zero is attached to each object. At each intermediate node along the way, the variable is increased by the cost of the last link the object has just traversed. The value is then used to update the cost loss of the object maintained by the associated cache. If the object is inserted into the cache, the node resets the value to zero before forwarding the object downstream. In this way, the updated cost loss is disseminated to all the caches on the way.

4.3.5 Simulation Model

To the best of our knowledge, it is difficult to find true trace data in the open literature to simulate our model. Therefore, we generated the simulation model from the empirical results presented in [1, 7, 11, 14, 23, 53].

The network topology was randomly generated by the Tier program [14]. Experiments for many topologies with different parameters have been conducted and the relative performance of our model was found to be insensitive to topology changes. Here, only the experimental results for one topology was listed due to space limitations. The characteristics of this topology and the workload model are shown in Table 4.2, which are chosen from the open literature and are considered to be reasonable.

The WAN (Wide Area Network) is viewed as the backbone network to which no servers or clients are attached. Each MAN (Metropolitan Area Network) node is assumed to connect to a content server. Each MAN and WAN node is associated with an en-route cache. Similar to the studies in [11, 15, 46, 88], cache size is described as the total relative size of all objects available in the content server. In our experiments, the object sizes are assumed to follow a Pareto distribution and the average object size is $26KB$. We also assume that each multimedia object has five versions and that the transcoding graph is as shown in Figure 4.1 in Section 4.1.1. The transcoding delay is determined as the quotient of the object size to the transcoding rate. In our experiments, the client at each MAN node randomly generates the requests, and the average request rate of each node follow–s the distribution of $U(1, 9)$, where $U(x, y)$ represents a uniform distribution between x and y. The access frequencies of both the content servers and the objects maintained by a given server follow a Zipf-like distribution [11,69]. Specifically, the probability of a request for object O in server S is proportional to $1/(i^{\alpha} \cdot j^{\alpha})$, where S is the ith most popular server and O is the jth popular object in S. The delay of both MAN links and WAN links follows an exponential distribution, where the average delay for WAN links is 0.46 seconds and the average delay for WAN links is 0.06 seconds.

The cost for each link is calculated by the access delay. For simplicity, the delay caused by sending the request and the relevant response for that request is proportional to the size of the requested object. Here, we consider the average object sizes for calculating all delays, including the transmission delay, and transcoding delay. The cost function is taken to be the delay of the link, which means that the cost in our model is interpreted as the access latency in our simulation.

Table 4.2: Parameters Used in Simulation

Parameter	Value
Number of WAN Nodes	200
Number of MAN Nodes	200
Delay of WAN Links	Exponential Distribution $p(x) = \theta^{-1}e^{-x/\theta}$ $(\theta = 0.45\ Sec)$
Delay of MAN Links	Exponential Distribution $p(x) = \theta^{-1}e^{-x/\theta}$ $(\theta = 0.06\ Sec)$
Number of Servers	100
Number of Web Objects	1000 objects per srever
Web Object Size Distribution	Pareto Distribution $p(x) = \frac{ab^a}{a-1}$ $(a = 1.1, b = 8596)$
Web Object Access Frequency	Zipf-Like Distribution $\frac{1}{i^\alpha}$ $(i = 0.7)$
Relative Cache Size Per Node	4%
Average Request Rate Per Node	$U(1, 9)$ requests per second
Transcoding Rate	$20KB/Sec$

We apply a "sliding window" technique to estimate the access frequency to make our model less sensitive to transient workload [88]. Specifically, for each object O, $f(O, v)$ is calculated by $K/(t - t_K)$, where K is the number of accesses recorded, t is the current time, and t_K is the Kth most recently referenced time (the time of the oldest reference in the sliding window). K is set to 2 in the simulation. To reduce overhead, the access frequency is only updated when the object is referenced and at reasonably large intervals, e.g., several minutes, to reflect aging, which is also applied in [90].

4.3.6 Performance Evaluation

In this section, we evaluate the performance of our model (proposed in Section 4.3.3) in terms of several performance metrics. The performance metrics we used in our simulation include delay-saving ratio (DSR), which is defined as the fraction of communication and server delays which is saved by satisfying the references from the cache instead of the server, average access latency (ASL), request response ratio (RRR), which is defined as the ratio of the access latency of the target object to its size, object hit ratio (OHR), which is defined as the ratio of the number of requests satisfied by the caches as a whole to the total number of requests, and highest server load (HSL), which is defined as the largest number of bytes served by the server per second. In the following figures CCR shows the results for our coordinated cache replacement model proposed in Section 4.3.3. Table 4.3 lists the abbreviations used in this section.

Table 4.3: Abbreviations Used in Performance Analysis

Meaning	Abbreviation	Decription
Performance Metric	DSR	Delay-Saving Ratio (%)
	ASL	Average Access Latency (Sec)
	RRR	Request Response Ratio (Sec/MB)
	OHR	Object Hit Ratio (%)
	HSL	Highest Server Load (MB/Sec)
Caching Model	CCR	Coordinated Cache Replacement
	AE	Standing for Aggregate Effect
	$LNC-R$	Least Normalized Cost Replacement
	LRU	Least Recently Used

Impact of Cache Size

In this experiment set, we compare the performance results of different models across a wide range of cache sizes, from 0.04 percent to 15.0 percent.

The first experiment investigates DSR as a function of the relative cache size per node and Figure 4.10 shows the simulation results. As presented in Figure 4.10, we can see that our model outperforms the other models since our coordinated cache replacement model determines the replacement candidates cooperatively among all the nodes on the path from the server to the client, whereas existing solutions, including LRU, $LNC-R$, and AE, consider decide cache replacement candidates locally, i.e., only from the view of a single node. Specifically, the mean improvements of DSR over AE, $LNC-R$, and LRU are 21.2 percent, 18.9 percent, and 13.0 percent, respectively.

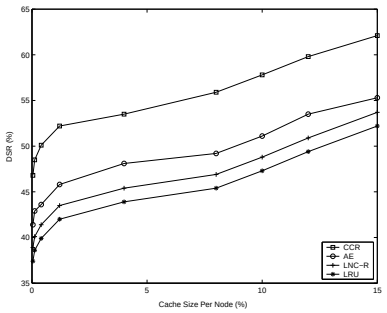

Figure 4.10: Experiment on DSR

Figure 4.11 shows the simulation results of ASL and RRR as a function of the relative cache size at each node. Clearly, the lower the ASL or the RRR, the better the performance. As we can see, all models provide steady performance improvement as the cache size increases. We can also see that CCR significantly improves both ASL and RRR compared to AE, $LNC-R$ and LRU, since our model determines the cache replacement candidates in an optimal and coordinated way, while the others decide the replacement candidates only by considering the situation of a single node. For ASL to achieve the same performance as CCR, the other models need 2 to 6 times as much cache size.

Figure 4.12 shows the results of OHR and HSL as a function of the relative cache size

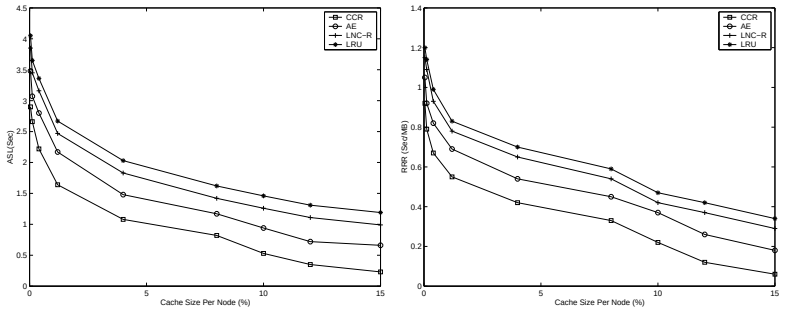

Figure 4.11: Experiment for ASL and RRR

for different models. By computing the optimal replacement candidates, we can see that the results for our model can greatly outperform those of the other solutions, especially for smaller cache sizes. We can also see that OHR steadily improves as the relative cache size increases, which conforms to the fact that more requests will be satisfied by the caches as the cache size becomes larger. Particularly, the mean improvements of DSR over AE, $LNC-R$, and LRU are 27.1 percent, 22.5 percent, and 13.9 percent, respectively. It can also be seen that HSL for our model is lower than that of the other solutions. We can also see that HSL decreases as the relative cache size increases.

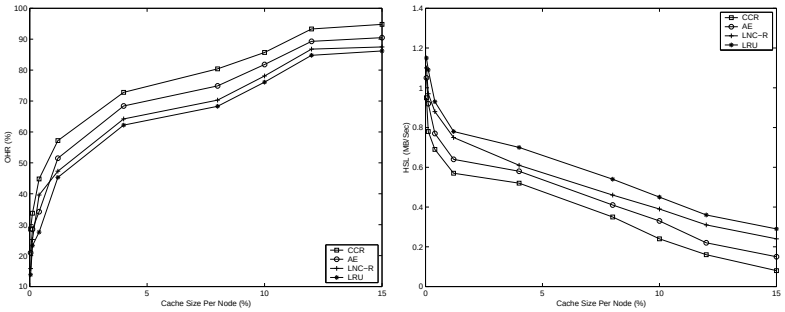

Figure 4.12: Experiment for OHR and HSL

Impact of Object Access Frequency

This experiment set examines the impact of object access frequency distribution on the performance results of different models. Figure 4.13 shows the performance results of DSR, RRR, and OHR for the values of Zipf parameter α from 0.2 to 1.0.

We can see that CCR consistently provides the best performance over a wide range of object access frequency distributions. Specially, CCR reduces or improves DSR by 17.7.4 percent, 15.0 percent, and 7.5 percent compared to LRU, $LNC-R$, and AE, respectively; the default cache size used here (4 percent) is fairly large in the context of en-route caching due to the large network under consideration.

Impact of the Number of Client Classes

This experiment set examines the impact of the number of client classes on the performance results of different solutions. The number of client classes refers to the number of transcodable versions. In our experiments, the number of transcodbale versions is $1-5$

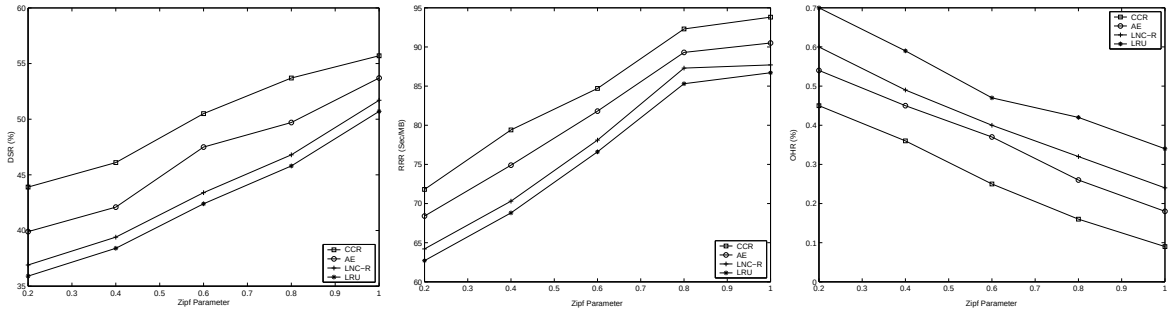

Figure 4.13: Experiment for DSR, RRR, and OHR

and the relevant vectors are $(100\%, 0, 0, 0, 0)$, $(50\%, 0, 50\%, 0, 0)$, $(50\%, 0, 30\%, 0, 20\%)$, $(40\%, 0, 30\%, 20\%, 10\%)$, and $(20\%, 15\%, 20\%, 15\%, 30\%)$.

Figure 4.14 shows the simulation results on DSR and RRR. We can see that DSR and RRR decrease as the number of the transcodable versions increase due to the fact that the requests from the clients will tend to disperse with increasing the number of the transcodable versions. Specifically, the mean improvements of DSR over AE, $LNC-R$, LRU are 9.5 percent, 8.2 percent, and 5.1 percent, respectively.

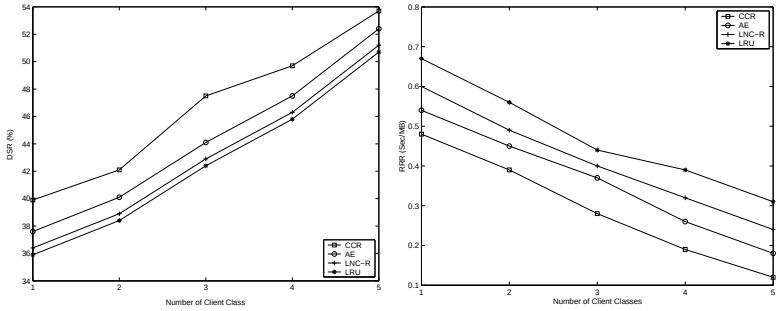

Figure 4.14: Experiment for DSR and RRR

4.4 Chapter Summarization

In this chapter, we proposed two effective cache replacement algorithms for transcoding proxy caching and conducted some simulation experiments to evaluate our algorithms. Some of the contents in this chapter can be found in [60, 62].

Chapter 5

Conclusions

5.1 Summarization

In this dissertation, we studied some key problems for coordinated en-route web caching, including (multimedia) object caching, (transcoding) proxy placement, and cache replacement. The main contents of this dissertation are generalized as follows:

- In Chapter 2, we addressed some key problems (i.e. object caching and proxy placement) for coordinated en-route web caching and presented extensive simulation results to evaluate our solutions.
- In Chapter 3, we studied some key problems for coordinated en-route transcoding proxy caching (i.e. multimedia object caching and transcoding proxy placement). We also addressed the problem of multimedia object placement for transparent data replication. We presented extensive simulation results to compare the performance of our models with those proposed in the literature.
- In Chapter 4, we proposed two effective cache replacement schemes for transcoding proxy caching and presented some simulation experiments to evaluate our schemes.

5.2 Future Work

The following issues can be considered for future research.

- Coordinated en-route web object caching in other types of regular networks: The techniques of applying dynamic programming showed in this dissertation may serve as useful tools for deriving such solutions in other networks, such as mesh networks, lattice networks, content distribution networks ($CDNs$).
- Dynamic web object caching: In this dissertation, the object contents addressed are static. Unlike static Web contents, which only involve file fetches when they are requested, dynamic object contents are constructed by running application programs on base data which often change frequently. Although the multimedia objects we studied can be viewed a kind of dynamic Web contents, more issues should be studied in future research. For example, a user's request on a streaming media can be satisfied by the combination of several streaming chips by applying some

programs. Studying this problem for en-route web caching is more complex and interesting.

- Caching for peer-to-peer system: Peer-to-peer ($P2P$) systems that provide persistence storage and allow users to interact and share distributed resources in wide-area environments are gaining an increasing popularity. Many interesting research challenges have been raised on caching for peer-to-peer systems, such as caching architecture, caching protocol, object caching, etc.

Bibliography

[1] C. Aggarwal, J. L. Wolf, and P. S. Yu:“ Caching on the World Wide Web”, IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1, pp. 94-107 (1999).

[2] M. Arlitt and C. Williamson:“ Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers”, Simulation Journal, Vol. 68, No. 1, pp. 23-33 (January 1997).

[3] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox :“ Caching Proxies: Limitations and Potentials”, Proceedings of the 4th International WWW Conference, Boston, MA (December 1995).

[4] B. Awerbuch, Y. Bartal, and A. Fiat: “Distributed Paging for General Networks”, Journal of Algorithms, Vol. 28, pp. 67-104 (1998).

[5] A. Balamash and M. Krunz:“ An Overview of Web Caching Replacement Algorithms”, IEEE Communications Surveys & Tutorials, Vol. 6, No. 2, pp.44-56 (2004).

[6] H. S. Bassali, K. M. Kamath, R. B. Hosamani, and L. Gao:“ Hierarchy-Aware Algorithms for CDV Proxy Placement in the Internet”, Computer Communications surveys, Vol. 26, pp.251-263 (2003).

[7] P. Barford and M. Crovella: “Generating Representative Web Workloads for Network and Server Performance Evaluation”, Proceedings of ACM SIGMETRICS’98, pp. 151-160 (1998).

[8] T. Bates, E. Gerich, L. Joncheray, J. M. Jouanigot, D. Karrenberg, M. Terpstra and J. Yu: “Representation of IP Routing Policies in a Routing Registry”, RFC 1786 (1995).

[9] M. Bhide, P. Deolasee, A. Katkar, and A. Panchbudhe:“ Adaptive Push-Pull: Disseminating Dynamic Web Data”, IEEE Transactions on Computers, Vol. 51, No. 6, pp. 652-667 (June 2002).

[10] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura:“ Self-Organizing Wide-Area Network Caches”, Proceedings of IEEE INFOCOM’98, pp. 600-608 (1998).

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker: “ Web Caching and Zip-like Distributions: Evidence and Implications”, Proceedings of IEEE INFOCOM’99, pp. 126-134 (1999).

[12] S. Buchholz and T. Buchholz: " Replica Placement in Adaptive Content Distribution Networks", Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 1705-1710 (2004).

[13] R. Caceres, E Douglis, A. Feldmann, G. Glass, and M. Rabinovich:" Web Proxy Caching: the Devil Is in the Details", ACM Performance Evaluation Review, Vol. 26, No. 3, pp. 11-15 (December 1998)

[14] K. L. Calvert, M. B. Doar, and E. W. Zegura: " Modelling Internet Topology", IEEE Communication Magazine, Vol. 35, No. 6, pp. 160-163 (1997).

[15] P. Cao and S. Irani: " Cost-Aware WWW Proxy Caching Algorithms", Proceedings of the First USENIX Symposium Internet Technologies and Systems (USITS), pp. 193-206 (1997).

[16] P. Cao and C. Liu: " Maintaining Strong Cache Consistency in the World Wide Web", Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, pp. 193-206 (May 1997).

[17] P. Cao, J. Zhang, and K. Beach:" Active Cache: Caching Dynamic Contents on the Web", Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), pp. 373-388 (1998).

[18] V. Cardellini, P. Yu, and Y. Huang:" Collaborative Proxy System for Distributed Web Content Transcoding", Proceedings of ACM International Conference Information and Knowledge Management, pp. 520-527 (2000).

[19] V. Cate:" Alex - a Global Filesystem", Proceedings of the 1992 USENIX File System Workshop, pp. 1-12 (May 1992).

[20] J. Challenger, A. Iyengar, and P. Dantzig:" A Scalable System for Consistently Caching Dynamic Web Data", Proceedings of Infocom'99 (1999).

[21] C. Chandra and C. S. Ellis:" JPEG Compression Metric as a Quality-Aware Image Transcoding", Proceedings of USENIX Second Symposium Internet Technology and Systems, pp. 81-92 (1999).

[22] S. Chandra, C. S. Ellis, and A. Vahdat:" Differentiated Multimedia Web Services Using Quality Aware Transcoding", IEEE Journal on Selected Areas in Communications, Vol. 18, No. 12, pp. 2544 - 2565 (December 2000).

[23] C. Chang and M. Chen:" On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies", IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 6, pp. 611-624 (June 2003).

[24] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell: " A Hierarchical Internet Object Cache", Proceedings of the USENIX1996 Technical Conference, pp. 22-26 (1996).

[25] L. Cherkasova, Y. Fu, W. Tang, and A. Vahdat: " Architecture and Performance of Server-Directed Transcoding", ACM Transactions on Internet Technology (TOIT), Vol. 3, No. 3, pp. 22-26 (November 2003).

[26] E. Cohen, B. Krishnamurthy, and J. Rexford:“ Efficient Algorithms for Predicting Requests to Web Servers”, Proceedings of Infocom’99 (February 1999).

[27] K. Chinen and S. Yamaguchi: “ An Interactive Prefetching Proxy Server for Improvement of WWW Latency”, Proceedings of INET 97 (June 1997).

[28] C. Cunha, A. Bestavros, and M. Crovella: “ Characteriatics of WWWW Client-Based Traces”, Technical Report TR-95-010, Boston University (April 1995).

[29] C. Cunha and C. F. B. Jaccoud: “ Determining WWW User’s Next Access and Its Application to Pre-fetching”, Proceedings of the 2nd IEEE Symposium on Computers and Communications (July 1997).

[30] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson: “ Cooperative Caching: Using Remote Client Memory to Improve File System Performance”, Proceedings of First Symposium Operating Systems Design and Implementations, pp. 267-280 (1994).

[31] B. D. Davison:“ A Web Caching Primer”, IEEE Internet Computing, Vol. 5, No. 4, pp. 38-45 (2001).

[32] P. Cao and S. Irani:“ Improving Proxy Cache Performance: Analysis of three Replacement Policies”, IEEE Internet Computing, Vol. 3, No. 6, pp. 44-50 (1999).

[33] E. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul:“ Rate of Change and Other Metrics: A Live Study of the World- Wide Web”, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (December 1997).

[34] B. M. Duska, D. Marwood, and M. J. Feelay:“ The Measured Access Characteristics of World Wide Web Client Proxy Caches”, Proceedings of USENIX Symposium on Internet Technologies and Systems (December 1997).

[35] L. Fan, P. Cao, J. Almeida and A. Z. Broder:“ Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol”, IEEE/ACM Transactions on Networking, Vol. 5, No. 3, pp. 281-293 (June 2000).

[36] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Ra binovich:“ Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments”, Proceedings of Infocom’99 (1999).

[37] R. Floyd and B. Housel:“ Mobile Web Access Using Network Web Express”, IEEE Personal Comm., Vol. 5, No. 5, pp. 47-52 (December 1998).

[38] M. R. Garey and M. Arlitt:“ Computer & Intractability: A Guide to the Theory of NP-Completeness”, W H Freeman (November 1979).

[39] S. Glassman: “ A Caching Relay for the World Wide Web”, Computer Networks and ISDN Systems, Vol 27, No. 2, pp, 165-173 (1994).

[40] J. Gwetzman and M. Seltzer:“ World Wide Web Cache Consistency”, Proceedings of the USENIX Technical Conference, pp. 141-152 (January 1996).

[41] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas:“ Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing”, IEEE Personal Comm., Vol. 5, No. 6, pp. 8-17 (December 1998).

[42] S. Jamin, C. Jin, Y. Jin, D. Raz, and Y. Shavitt:“ Constrained Mirror Placement on the Internet”, Proceedings of IEEE INFOCOM 2001 Conference, Anchorage, Alaska (April 2001).

[43] S. Jamin, C. Jin, D. Raz, Y. Shavitt, and L. Zhang:“ On the placement of Internet Instrumentation”, Proceedings of INFOCOM'2000, Israel, 26-30 (March 2000).

[44] X. Jia, D. Li, X. Hu, and D. Du: “ Optimal Placement of Web Proxies for Replicated Web Servers in the Internet”, The Computer Journal, Vol. 44, No. 5, pp. 329-339 (2001).

[45] A. Jiang and J. Bruck: “ Optimal Content Placement for En-Route Web Caching”, Proceedings of the Second International Symposium on Network Computing and Applications (NCA'03), pp. 9-16 (2003)

[46] S. Jin and A. Bestavros: “ Greeddual* Web Caching Algorithm Exploiting the Two Sources of Temporal Locality in Web Request Streams”, Computer Comm., Vol. 4, No. 2, pp. 174-183 (2001).

[47] M. R. Korupolu and M. Dahlin: “ Coordinated Placement and Replacement for Large-Scale Distributed Caches”, IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No. 6, pp. 1317-1329 (2002).

[48] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman: “ Placement Algorithms for Hierarchical Coorperative Caching”, Proceedings of 10th Annual ACM-SIAM Symposium Discrete Algorithms, pp. 586-595 (1999).

[49] P. Krishnan, D. Raz, and Y. Shavitt: “ The Cache Location Problem”, IEEE/ACM Transactions on Networking, Vol. 8, No. 5, pp. 568-582 (2000).

[50] T. M. Kroeger, D. D. Long, and J. C. Mogul: “ Exploring the Bounds of Web Latency Reduction from Caching and Prefetching”, Proceedings of USENIX Symposium on Internet Technologies and Systems, pp. 13-22 (December 1997).

[51] A. Leff, J. L. Wolf, and P. S. Yu: “ Replication Algorithms in a Remote Caching Architecture”, IEEE Transaction on Parallel and Distributed Systems, Vol. 4, No. 11, pp. 1185-1204 (1993).

[52] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias:“ Design and Performance of Web Server Accelerator”, Proceedings of Infocom'99 (1999).

[53] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby: “ On the Optimal Placement of Web Proxies in the Internet”, Proceedings of IEEE INFOCOM'1999, pp. 1282-1290 (1999).

[54] K. Li and H. Shen: “Coordinated En-Route Multimedia Object Caching in Transcoding Proxies for Tree Networks”, ACM Transactions on Multimedia Computing, Communications and Applications (TOMCAPP), Vol. 5, No. 3, pp. 1-26, (August 2005).

[55] K. Li, H. Shen, F. Chin, and S. Zheng: "Optimal Methods for Coordinated En-Route Web Caching for Tree Networks", ACM Transactions on Internet Technology (TOIT), Vol. 5, No. 2 (May 2005).

[56] K. Li and H. Shen: "Optimal Methods for Object Placement in En-Route Web Caching for Tree Networks and Autonomous Systems", International Journal of High Performance Computing and Networking (IJHPCN), Vol. 3, No. 5 (September 2004).

[57] K. Li and H. Shen: "Optimal Methods for Proxy Placement in Coordinated En-Route Web Caching," IEICE Transactions on Communications, Vol. E88-B, No. 4, 1458-1466 (April 2005).

[58] K. Li and H. Shen: "Optimal Proxy Placement for Coordinated En-Route Transcoding Proxy Caching", IEICE Transactions on Information and Systems, Vol. E87-D, No. 12, pp. 2689-2696 (December 2004).

[59] K. Li and H. Shen: "Proxy Placement Problem for Coordinated En-Route Transcoding Proxy Caching", International Journal of Computer Systems, Science and Engineering (CSSE), Vol. 19, No. 5, pp. 95-103 (September 2004).

[60] K. Li, H. Shen, and F. Chin: "Cooperative Determination on Cache Replacement Candidates for Transcoding Proxy Caching", Proceedings of The 2005 International Conference on Computer Networks and Mobile Computing (ICCNMC 2005), pp. 178-187, Zhangjiajie, China (August 2005).

[61] K. Li, H. Shen, F. Chin, and L. Huang: "Multimedia Object Placement Solutions for Hybrid Transparent Data Replication", Proceedings of The IEEE Global Telecommunications Conference (GLOBECOM2005), St. Louis, USA (December 2005).

[62] K. Li, H. Shen, K. Tajima, and L. Huang: "An Effective Cache Replacement Algorithm for Transcoding Proxy Caching", Journal of Supercomputing, accepted.

[63] H. Lin and K. Yang:" Placement of Repair Servers to Support Server-Based Reliable Multicast", Proceedings of IEEE ICC, pp. 1802-1806 (2001).

[64] T. S. Lonn and V. Bharghavan: " Alleviating the Latency and Bandwidth Problems in WWW Browsing", Proceedings of USENIX Symposium on Internet Technologies and Systems, pp. 219-230 (December 1997).

[65] B. S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson:" Adaptive Web Caching: Towards A New Global Caching Architecture", Computer Networks, Vol. 30, nos 22-23, pp. 2169-2177 (1998).

[66] R. Mohan, J. R. Smith and C. Li:" Adapting Multimedia Internet Content for Universal Access", IEEE Transactions on Multimedia, Vol. 1, No. 1, pp. 104-114 (March 1999).

[67] M. Nabeshima:" The Japan Cache Project: An Experiment on Domain Cache", Computer Network and ISDN Systems, 29, 8-13 (1997)

[68] V. N. Padmanabhan and J. C. Mogul:" Using predictive prefetching to improve World Wide Web latency", Proceedings of Sigcomm'96 (1996).

[69] V. N. Padmanabhan and L. Qiu:“ The Content and Access Dynamics of a Busy Site: Findings and Implications”, Proceedings of ACM SIGCOMM’00, pp.111-123 (August 2000).

[70] T. Palpanas and A. Mendelzon:“ Web prefetching using partial match prediction”, Proceedings of WCW’99 (1999).

[71] S. C. Park, Y. W. Park, and .Y E. Son:“ A Proxy Server Management Scheme for Continuous Media Objects Based on Object Partitioning”, Proceedings of the Eighth International Conference on Parallel and Distributed Systems (ICPADS), pp. 757-762, (June 2001)

[72] V. Paxson:“ End-To-End Routing Behavior in the Internet”, IEEE/ACM Transactions on Networking, Vol. 5, No. 5, pp. 601-615 (1997).

[73] G. Pierre and M. Steen: “ Dynamically Selecting Optimal Distribution Strategies for Web Documents”, IEEE Transactions on Computers, pp. 637-651, Vol. 51, No. 6 (2002).

[74] J. Pitkow:“ Summary of WWW Characterizations”, World Wide Web, Vol. 2, No. 1-2, pp. 3-13 (1999).

[75] D. Povey and J. Harrison:“ A Distributed Internet Cache”, Proceedings of the 20th Australian Computer Science Conference, Sydney, Australia, (February 1997).

[76] K. Psounis and B. Prabhakar:“ Efficient Randomized Web-Cache Replacement Schemes Using Samples from Past Eviction-Times”, IEEE/ACM Transactions on Networking, Vol. 10, No. 4, pp. 441-454 (August 2002).

[77] L. Qiu, V. N. Padmanabhan, and G. M. Voeljer :“ On the Placement of Web Server Replicas”, Proceedings of IEEE INFOCOM 2001 Conference, Anchorage, Alaska (April 2001).

[78] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell:“ A Hierarchical Internet Object Cache. Proceedings of USENIX Annual Technical Conference, pp. 153-164 (1996).

[79] M. Rabinovich and O. Spatscheck: “ Web Caching and Replication”, Addison-Wesley (2002).

[80] P. Rabinovich and H. Wang: “ Dhttp: An Efficient and Cache-Friendly Transfer Protocol for Web Traffic”, Proceedings of IEEE INFOCOM’01, pp. 1597-1606 (2001).

[81] L. Ramaswamy and L. Liu : “ An Expiration Age-Based Document Placement Scheme for Cooperative Web Caching”, IEEE Transactions on Knowledge and Data Engineering, Vol. 16, No. 5, pp. 585-600 (May 2004).

[82] L. Rizzo and L. Vicisano:“ Replacement Policies for a Proxy Cache”, IEEE/ACM Transactions on Networking, Vol. 8, No. 2, pp. 158-170 (April 2000).

[83] P. Rodriguez and S. Sibal: “ Spread: Scalable Platform for Reliable and Efficient Automated Distribution”, Computer Networks, Vo. 33, pp. 33-49 (2000).

[84] P. Rodriguez, S. Sibal, and O. Spatscheck: " Tpot: Translucent Proxying of TCP", Proceedings of Fifth International Web Caching and Content Delivery Workshop (WCW) (2000).

[85] P. Rodriguez, C. Spanner, and E. W. Biersack: " Analysis of Web Caching Architectures: Hierarchical and Distributed Caching", IEEE/ACM Transactions on Networking, Vol. 9, No. 4, pp. 404-418 (2001).

[86] P. Scheuermann, J. Shim, and R. Vingralek: " A Case for Delay-Conscious Caching of Web Documents", Computer Networks and ISDN Systems, Vol. 29, No. 8-13, pp, 997-1005 (1997).

[87] B. Shen, S.-J. Lee, and S. Basu: " Caching Strategies in Transcoding-Enabled Proxy Systems for Streaming Media Distribution Networks", IEEE Transactions on Multemidia, Vol. 6, No. 2, pp. 375-386 (April 2004).

[88] J. Shim, P. Scheuermann, and R. Vingralek: " Proxy Cache Algorithms: Design, Implementation, and Performance", IEEE Transactions Knowledge and Data Engineering, Vol. 11, No. 4, pp, 549-562 (1999).

[89] D. Starobinski and D. Tse:" Probabilistic Methods for Web Caching", Performance Evaluation , Vol. 46, Nos. 2-3, pp. 125-137 (October 2001).

[90] X. Tang and S. T. Chanson: " Coordinated En-Route Web Caching", IEEE Transactions on Computers, Vol. 51, No. 6, pp. 595-607 (2002).

[91] X. Tang, F. Zhang, and S. T. Chanson:" Streaming Media Caching Architectures for Transcoding Proxies", Performance Evaluation, Vol. 46, Nos. 2-3, pp. 125-137 (October 2001).

[92] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden: " A Survey of Active Network Research", IEEE Comm. Magazine, Vol. 35, No. 1, pp. 80-86 (1997).

[93] X. Tewari, M. Dahlin, H. M. Vin, and J. S. Kay: " Design Considerations for Distributed Caching on the Internet", Proceedings of 19th International Conference Distributed Computing Systems (ICDCS), pp. 273-284 (1999).

[94] A. Vakali, E. Terzi, E. Bertino, and A. Elmagarmid: " Hierarchical Data Placement for Navigational Multimedia Applications", Data & Knowledge Engineering, Vol. 44, pp. 49-80 (2003).

[95] V. Valloppillil and K W. Ross: " Cache Array Routing Protocol V1.0", Internet Draft, ¡draft-vinod-carp-v1-03.txt¿ (February 1998).

[96] V. V. Vazirani:" Approximation Algorithms", Springer (2001).

[97] A. Vetro, C. Christopoulos, and H. Sun: " Video Transcoding Architectures and Techniques: An Overview", IEEE Signal Processing Magazine, Vol. 20, No. 2, pp. 18-29 (March 2003).

[98] J. Wang: " A Survey of Web Caching Schemes for the Internet", ACM SIGCOMM Computer Communication Review, Vol. 29, No. 5, pp, 36-46 (1999).

[99] D. Wessels and K. Claffy:" ICP and the Squid Web Cache", IEEE Journal on Selected Areas in Communications, Vol. 16, No. 3, pp. 345-357 (April 1998).

[100] D. Wessels and K. Claffy:" Internet Cache Protocol (IPC)", Version 2, RFC 2186.

[101] D. Wessels and K. Claffy:" Application of Internet Cache Protocol (IPC)", Version 2, RFC 2187.

[102] S. Williams, M. Abrams, C. R. Standbridge, G. Abdulla, and E. A. Fox: " Removal Policies in Network Caches for World Wide Web Documents", Proceedings of ACM SIGCOMM'96, pp, 293-305 (1996).

[103] O. Wolfson and A. Milo:" The Multicast Policy and its Relationship to Replicated Data Placement", ACM Transactions on Database Systems, Vol. 16, No. 1, pp. 181-205 (1991).

[104] R. P. Wooster and M. Abrams:" Proxy Caching That Estimates Page Load Delays", Computer Networks and ISDN Systems, Vol. 29, nos 8-13, pp. 977-986 (September 1997).

[105] J. Xu, B. Li, and D. L. Li:" Placement Problems for Transparent Data Replication Proxy Services", IEEE Journal on Selected Areas in Communications, Vol. 20, No. 7, pp. 1383-1398 (2002).

[106] L. Yin, G. Cao, and Y. Cai:" A Generalized Cache Replacement Policy for Mobile Environments", Proceedings of the 2003 Symposium on Applications and the Internet (SAINT'03), pp. 14-21 (2003).

[107] P. S. Yu and E. A. MacNair: " Performance Study of a Collaborative Method for Hierarchical Caching in Proxy Servers", Computer Networks and ISDN Systems, Vol. 30, pp. 215-224 (1998).

Publications

Journal Papers

[1] Keqiu Li, Hong Shen, Keishi Tajima, and Liusheng Huang: "An Effective Cache Replacement Algorithm for Transcoding Proxy Caching", Journal of Supercomputing, accepted.

[2] Keqiu Li and Hong Shen: "Optimal Methods for Object Placement in En-Route Web Caching for Tree Networks and Autonomous Systems", International Journal of High Performance Computing and Networking (IJHPCN), Vol. 4, No. 5 (September 2005).

[3] Keqiu Li and Hong Shen: "Coordinated En-Route Multimedia Object Caching in Transcoding Proxies for Tree Networks", ACM Trans. on Multimedia Computing, Communications and Applications (TOMCAPP), Vol. 5, No. 3, pp. 1-26 (August 2005).

[4] Keqiu Li, Hong Shen, Francis Y. L. Chin, and Siqing Zheng: "Optimal Methods for Coordinated En-Route Web Caching for Tree Networks", ACM Trans. on Internet Technology (TOIT), Vol. 5, No. 3 (August 2005).

[5] Keqiu Li and Hong Shen: "Optimal Methods for Proxy Placement in Coordinated En-Route Web Caching," IEICE Trans. on Communications, Vol. E88-B, No. 4, 1458-1466 (April 2005).

[6] Keqiu Li and Hong Shen: "Optimal Proxy Placement for Coordinated En-Route Transcoding Proxy Caching", IEICE Trans. on Information and Systems, Vol. E87-D, No. 12, pp. 2689-2696 (December 2004).

[7] Keqiu Li and Hong Shen: "Proxy Placement Problem for Coordinated En-Route Transcoding Proxy Caching", International Journal of Computer Systems, Science and Engineering (CSSE), Vol. 19, No. 6, pp. 327-335 (November 2004).

Conference Papers

[8] Keqiu Li, Hong Shen, and Di Wu: "Performance Evaluation of An Optimal Solution for Coordinated Cache Replacement in Transcoding Proxies", Proc. of The 4th International Conference on Grid and Cooperative Computing (GCC 2005), Beijing, China (December 2005).

[9] Keqiu Li, Hong Shen, Francis Y. L. Chin, and Liusheng Huang: "Multimedia Object Placement Solutions for Hybrid Transparent Data Replication", Proc. of The IEEE Global Telecommunications Conference (GLOBECOM 2005), St. Louis, USA (November 2005).

[10] Keqiu Li, Keishi Tajima, and Hong Shen: "Cache Replacement for Transcoding Proxy Caching", Proc. of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2005), France (September 2005).

[11] Keqiu Li, Hong Shen, and Francis Y. L. Chin: "Cooperative Determination on Cache Replacement Candidates for Transcoding Proxy Caching", Proc. of the 2005 International Conference on Computer Networks and Mobile Computing (ICCNMC 2005), pp. 178-187, Zhangjiajie, China (August 2005).

[12] Keqiu Li, Hong Shen, and Francis Y. L. Chin: "Placement Solutions for Multiple Versions of a Multimedia Object", Proc. of the 8th IEEE International Symposium on Object-oriented Real-time distributed Computing, pp. 224-231, Seattle, Washington, USA (May 2005).

[13] Keqiu Li and Hong Shen: "Dynamically Selecting Distribution Strategies for Web Documents According to Access Pattern", Proc. of the Fifth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 04), pp. 554-557, Singapore (December 2004).

[14] Keqiu Li, Hong Shen, and Keishi Tajima: "Cache Design for Transcoding Proxy Caching", Proc. of IFIP International Conference on Network and Parallel Computing 2004 (NPC04), pp. 187-194, Huhan, China (October 2004).

[15] Keqiu Li and Hong Shen:" An Improved GreedyDual* Cache Document Replacement Algorithm", Proc. of IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), pp. 457-460, Beijing, China (September 2004).

[16] Keqiu Li and Hong Shen:" Coordinated En-Route Transcoding Caching for Tree Networks", Proc. of the Tenth International Conference on Parallel and Distributed Systems (ICPADS'04), pp. 109-116, California, USA (July 2004).

[17] Keqiu Li and Hong Shen:" Transcoding Proxy Placement in En-Route Web Caching", Proc. of the 2004 Communication Networks and Services Research Conference (CNSR-2004), pp. 276-285, Fredericton, Canada (May 2004).

[18] Keqiu Li and Hong Shen:" Proxy Placement in Coordinated En-Route Transcoding Caching for Tree Networks", Proc. of the Seventh Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN), pp.226-231, Hong Kong, China (May 2004).

[19] Keqiu Li and Hong Shen:" Coordinated En-Route Web Caching in Transcoding Proxies", Proc. of the Sixth Asia Pacific Web Conference (APWEB'04), pp 772-781, Hangzhou, China (April 2004).

[20] Keqiu Li and Hong Shen:" Optimal Placement of Web Proxies for Tree Networks", Proc. of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-04), pp. 1479-1487, Taibei, Taiwan (March 2004).

[21] Keqiu Li and Hong Shen:“ Optimal Methods for Object Placement in En-Route Web Caching for Tree Networks and Autonomous Systems”, Proc. of the Second International Workshop on Grid and Cooperative Computing (GCC 2003), pp 263-270, Shanghai, China (December 2003).

[22] Keqiu Li and Hong Shen:“ Constrained Coordinated En-Route Web Caching in Tree Networks”, Proc. of the Third International Conference on Hybrid Intelligent Systems (HIS'03), pp 1054-1063, Melbourne, Australia (December 2003).

[23] Keqiu Li and Hong Shen:“ An Optimal Method for Coordinated En-Route Web Object Caching”, Proc. of the Fifth International Symposium on High Performance Computing (ISHPC-V), pp 368-375, Tokyo, Japan (October 2003).