NETWORK-AWARE CLIENT CLUSTERING AND APPLICATIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Jia Wang

May 2001

NETWORK-AWARE CLIENT CLUSTERING AND APPLICATIONS

Jia Wang, Ph.D.

Cornell University 2001

The Internet consists of a large collection of hosts (IP addresses) connected by networks of links and routers. Many applications have been developed and deployed on the Internet. A good understanding of the Internet topology and client locations will help to develop new schemes that improve the performance of applications. One interesting and important problem to be addressed is how to group IP addresses based on their location on the Internet. A network-aware grouping of IP addresses providing a good level of address aggregation will help researchers obtain a solid understanding of the Internet topology and traffic load patterns. It is also useful in applications such as the World Wide Web. In this thesis, we define *cluster* to be a grouping of hosts that are topologically close together and likely to be under common administrative control. We propose a "network-aware" method to identify clusters based on information available from BGP routing table snapshots.

The experimental results on a wide range of Web server logs show that the entirely automated approach is able to identify clusters for 99.9% of the clients in the Web server logs. Sampled validation results show that the identified clusters meet the proposed validation tests using *nslookup* and *traceroute* in over 90% of the cases. The method is insensitive to BGP dynamics and is independent of Web server logs. Finally, we present an efficient self-corrective mechanism to increase

the accuracy of the initial approach and to make it adaptive to network dynamics.

The client cluster identification mechanism is applicable to Web caching, content distribution, server-based access prediction, server replication, Internet map discovery, and network management. In this thesis, we examine Web caching and server replication. The real-time client clustering information gives the service provider a global view of where their customers are located and how their demands change from over time. This is crucial information for service providers to enhance their services, reduce costs, and extend global presence.

We also study several potential improvements to the basic clustering scheme. We show experimentally that prefix-level clustering is necessary while super-clustering based on the originating AS information and sub-clustering based on the client access patterns will provide additional information. We also show that the same method can be applied on proxy logs for network-aware server clustering.

Biographical Sketch

Jia Wang was born on September 16, 1973 in Beijing, P. R. China. During most of her first fifteen years, she lived with her parents and her twin sister in a small city, Baoding, in the Hebei province of China. In her early age, her parents, who are professors of mathematics and physics at Hebei University, fostered her interests in science. Soon after she moved to Tianjin, China, in the third year of her high school study, she entered Nankai University in 1990 and selected mathematics as her major. After three years undergraduate study in China, she emigrated to the United States in 1993 with her family and transferred to the State University of New York, Binghamton. She accepted the challenge of working in computer science, a new discipline for her, and finished her B.S. in 1996. After graduation, she came to Cornell University to pursue her Doctoral degree at the Department of Computer Science in August 1996 and was awarded a National Science Foundation Graduate Fellowship. Three and half years later, she married Lusheng Ji, who has supported her throughout her study at Cornell, and in the meantime pursues his own Ph.D. degree in computer science at University of Maryland, College Park.

To my parents, Zhenyuan Wang, Xuezhen Zhu, my sister, Wei Wang, and my husband, Lusheng Ji.

Acknowledgements

I would like to thank my thesis advisor Balachander Krishnamurthy at AT&T Labs - Research, Florham Park, NJ, who has guided me through my Ph.D. thesis research work, for his constant support and encouragement. He has taught me everything important in high quality research, from exploring new ideas, formulating the results, to writing papers. I cannot imagine how I could have reached this point without his help and guidance.

I am extremely grateful to my thesis advisor Robbert van Renesse, who provided invaluable guidance on writing my thesis. Many thanks to my former advisor Srinivasan Keshav who taught me how to do research in my early stage of study. I am thankful to Zygmunt Haas for his support and serving on my committee as minor field advisor. Thanks to Eva Tardos for serving as the proxy of Srinivasan Keshav in my thesis defence.

I am greatly indebted to Adam L. Buchsbaum, Glenn S. Fowler, Phong Vo, Dave Belanger, and many others at AT&T Labs for their invaluable help. Also thanks to my former groupmates Wilson Huang, Rosen Sharma, and Yu Zhang, for their help and encouragement during my early years in graduate school. I would like to offer my special thanks to all my friends I have had in Ithaca, NY over the last four years, who shared my happiness, sadness, and worries.

And last, but not least, thanks to my family - my parents and my twin sister for their love, care, and support over all these years, and to my husband Lusheng Ji, who supports me in every circumstance, and without whom none of this would matter.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet consists of a large collection of hosts (IP addresses) connected by networks of links and routers. Many applications, such as the World Wide Web, have been developed and deployed on the Internet. Researchers have been studying techniques and schemes to improve the performance of applications. A good understanding of the Internet topology and clients' location will improve the current proposals, such as Web caching, and help to develop new schemes. One interesting and important problem to be addressed is how to group hosts (IP addresses) based on their location on the Internet. A network-aware grouping of IP addresses providing a good level of address aggregation will help researchers obtain a solid understanding of the Internet topology and traffic load patterns. It is also useful in many applications. For example, in the World Wide Web application, it can be used to identify the group of clients that are responsible for a most of a Web site's requests.

The World Wide Web is one of the most popular applications on the Internet. The number of Web sites have grown extremely fast. Some of the Web sites get a lot of requests from many clients. Some clients send many requests to particular

Web sites. Previous work [ACF+98, WVS+99] show that the geographical issue has a strong influence on the Web traffic. The clients belonging to an organization are more likely to request the same documents than a set of clients with the same size chosen at random from all the clients in the population. With Web traffic on the rise, Web sites have a particular interest in being able to identify clients that send many requests. For example, it is beneficial to move content closer to groups of clients that are responsible for large subsets of requests to an origin server. This lowers the latency perceived by the clients as well as the load on the Web server. If a group of clients are topologically close and under common administrative control (e.g., the same department in a university or division in a company), then the administrator could install one or more proxy caches in front of the clients to lower the client-perceived latency.

More generally, groups of IP address are observed in many places such as logs and traces. A partitioning of a set of IP addresses into non-overlapping groups, where all the IP addresses in a group are topologically close and under common administrative control, would be useful for several applications. In this thesis, we use the term – *cluster* – to denote such a grouping and examine the problem of IP address clustering and its applications. We assume that readers of this thesis have a working knowledge of computer networking. Please refer to [Kes97, WS94] for a good overview of computer networking.

1.1 Simple approaches to IP address clustering

IPv4 addresses are 4 bytes long and are divided in a two-part hierarchy: network number and host number. A subnet mask (specified as prefix length) describes

the partition between the two. There are a total of 128 Class A networks with 16,777,216 (2^{24}) hosts per network and a network prefix length of 8. There are a total of 16,384 (2^{14}) Class B network with 65,536 (2^{16}) hosts per network and a network prefix length of 16. There are a total of 2,097,152 (2^{21}) Class C networks with 256 (2^{8}) hosts per network and a network prefix length of 24. The Internet's growth led to exhaustion of the Class B network address space and routers not being able to manage the size of routing tables. CIDR (*Classless InterDomain Routing*) addressing was proposed as a mechanism to slow the growth of routing tables and the need for allocating new IP network numbers. For example, instead of an entire block of one Class A, Class B or Class C address being allocated to a typical network, more blocks of Class C addresses can be allocated to a single network (i.e., the network prefix lengths are not necessarily 8, 16, or 24). The Internet address space is allocated in such a manner as to allow aggregation of routing information along topological lines [Hal97, Moy98].

IP address clustering is not trivial because there is no reliable global registry. An Internet registry service such as ARIN [ari99] will only provide out-of-date and incomplete information on IP address clusters. One could use network tools, such as *nslookup* and *traceroute* [WS94], to obtain information about client locations. However, clustering IP addresses using *nslookup* and *traceroute* is not feasible. The overhead of running *nslookup* and *traceroute* on each IP address is much too high. Moreover, neither *nslookup* nor *traceroute* yields complete information about client locations. In addition, a complete partition of an arbitrary set of IP addresses would require knowledge that is not available to anyone (e.g., a Web server) outside the administrative entities of the IP addresses.

A simple way to identify client clusters is to group all the clients which share

the same first 24 bits in their IP addresses into the same cluster based on the assumption that the network prefix length is 24 [SCA99, ZQK00]. This is one approach to cluster identification if one relied solely on client information extracted from logs. However, it is well-known that IP addresses are not all organized in this fashion.

To illustrate this, Table 1.1 shows the distribution of prefix lengths extracted from the MAE-WEST NAP routing table snapshots taken from July 3 to July 6, 1999 [merle]. The corresponding histograms are shown in Figure 1.1. While around 50% of the prefix lengths are 24 bits, there are a large number of prefixes that have either longer or shorter lengths. Among non-24 bit prefixes there are more shorter prefixes (< 24) than longer ones (> 24) mainly due to the allocation of CIDR addresses and route aggregation[1]. From Table 1.1 and Figure 1.1, we also observe that there are slight changes on prefixes distribution from day to day. For example, the change of prefix distribution between July 3, 1999 and July 4, 1999 is about 0.5%. These changes are mainly due to the network routing dynamics, not IP address allocation.

Identifying client clusters based on the first three bytes of IP addresses has two main drawbacks. First, it mis-identifies small clusters, which share the same first three bytes of their prefixes, by considering them as one single Class C network (Figure 1.2(a)). For example, the hosts 151.198.194.17, 151.198.194.34, and 151.198.194.50 share the same first three bytes of IP addresses. The simple approach will consider them to be in a single Class C network with prefix 151.198.194 and netmask length 24. In reality they reside in three different networks with pre-

[1]With CIDR address allocation, the routing table can be shrunk by aggregating routing entries with adjacent IP address blocks and the same routing path [Hal97, Moy98].

Table 1.1: Distribution of prefix lengths at the Mae-West NAP.

	Date			
Prefix length	7/3/1999	7/4/1999	7/5/1999	7/6/1999
8	20	20	20	20
9	1	1	1	1
10	3	3	3	3
11	3	3	3	3
12	12	12	12	12
13	25	24	24	24
14	73	72	72	72
15	111	100	100	100
16	3,098	3,099	3,095	3,097
17	333	331	331	332
18	706	702	703	704
19	2,092	2,091	2,074	2,084
20	1,009	1,003	1,006	1,006
21	1,275	1,268	1,263	1,265
22	1,805	1,797	1,798	1,794
23	2,227	2,220	2,220	2,220
24	13,937	14,029	14,013	14,018
25	31	31	31	31
26	34	33	33	33
27	1	1	1	1
28	4	4	4	4
29	3	3	3	3
30	1	1	1	1

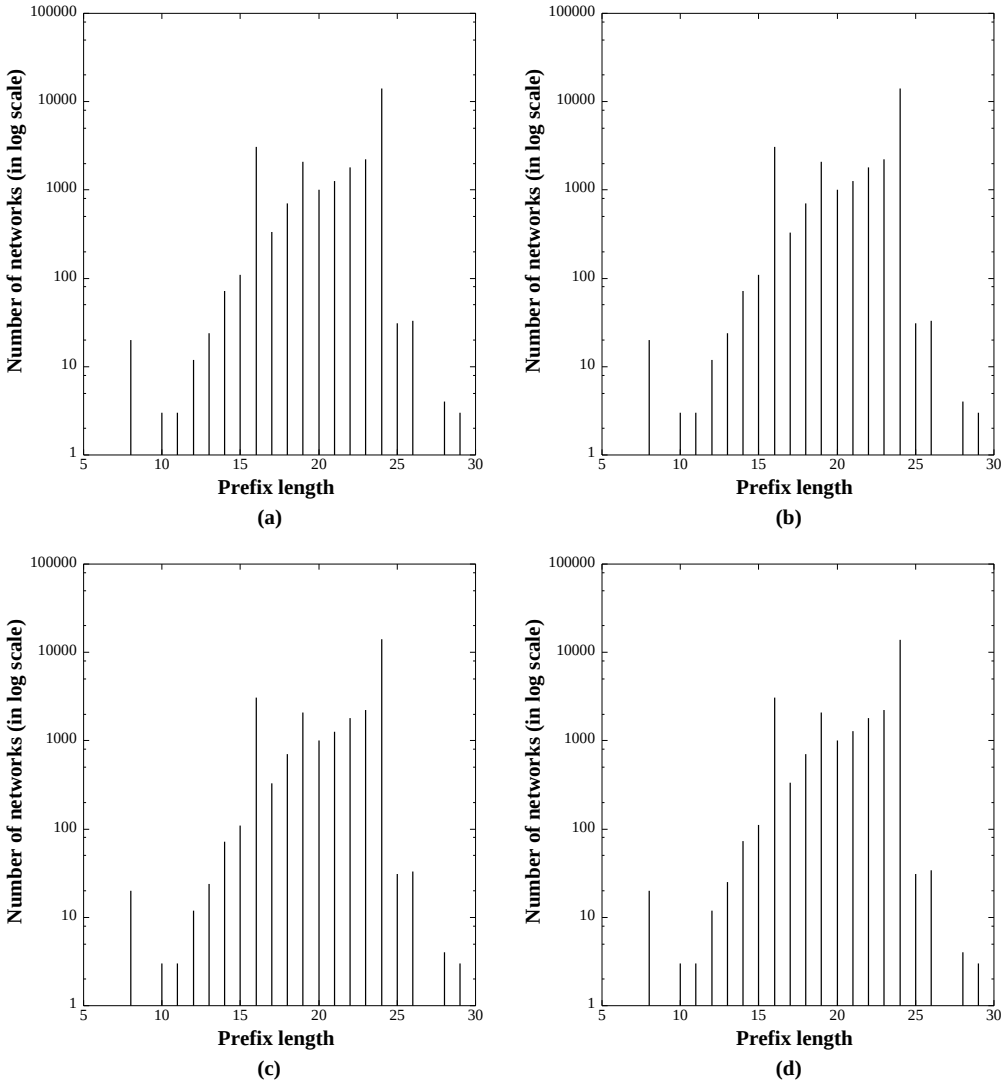


Figure 1.1: Histogram of prefix lengths at the Mae-West NAP: (a) 07/03/1999, (b) 07/04/1999, (c) 07/05/1999, (d) 07/06/1999.

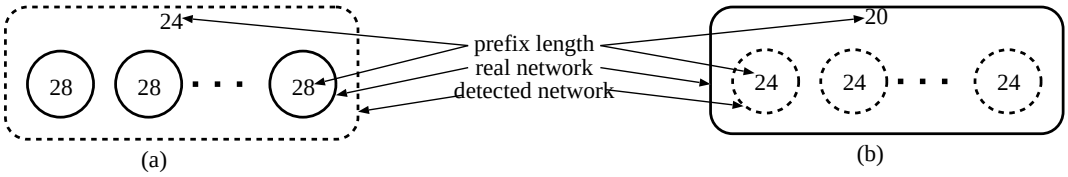


Figure 1.2: Two cases of misdetecting client clusters using the simple approach.

Table 1.2: A counter-example of the assumption that the prefix length is 24 bits.

IP address	Name	Prefix
151.198.194.17	client-151-198-194-17.bellatlantic.net	151.198.194.16/28
151.198.194.34	mailsrv1.wakefern.com	151.198.194.32/28
151.198.194.50	firewall.commonhealthusa.com	151.198.194.48/28

fixes 151.198.194.16, 151.198.194.32, and 151.198.194.48, and netmask length of 28 (Table 1.2). The names of these hosts are *client-151-198-194-17.bellatlantic.net*, *mailsrv1.wakefern.com*, and *firewall.commonhealthusa.com*, respectively, indicating that they most likely belong to different administrative entities. They are not likely to make common decisions on sharing proxies, for example. They may even be located geographically far away from each other.

Secondly, it is obvious that the simple prefix clustering technique may mis-cluster all Class A, Class B, and CIDR networks by dividing them into a number of Class C networks (Figure 1.2(b)). For example, the clients belonging to a single network with prefix length of 20 bits will be grouped into 16 Class C networks with prefix length of 24 bits, that is, a network with prefix length of 20 bits is false-partitioned into 16 Class C networks. In this case, a trace-driven simulation may underestimate the benefit of Web proxies due to less proxy sharing across the mis-identified clusters for example. In our experiments, the simplified assumptions of this approach fail a validation test in over 50% of the sampled cases.

As an alternate approach, one could identify client clusters based on Class A,

Class B, and Class C networks. While this method may give better client clusters than the simple approach, it is clear that there will still be inaccuracies, due both to CIDR addressing and to subnetting within the Class A and B networks. In the rest of this thesis, we use the simple approach as a comparison case for lack of other approaches.

1.2 Thesis contribution

The main contribution of this thesis consists of three parts:

- We propose a network-aware method to identify client clusters using the information available in BGP routing table snapshots.
- We examine the applications of network-aware IP address clustering.
- We study several potential improvements to the basic clustering scheme.

We explain each of them next.

1.2.1 BGP-based client clustering method

This thesis presents an entirely automated technique for a novel "network-aware" method of identifying client clusters starting with sources of IP addresses such as Web server logs, and using readily available information in BGP routing table snapshots.

We conducted experiments on a wide range of Web server logs. The results show that our method is able to identify clusters for 99.9% of the clients. We validate our network-aware approach by examining 1% samples of client clusters. We use two validation techniques, one based on *nslookup* and one based on *traceroute*. We

find that our method identifies clusters that meet the proposed tests in over 90% of the cases. We also propose an efficient self-corrective mechanism to increase the applicability and accuracy of our initial approach and to make it adaptive to network dynamics.

1.2.2 Applications

The applications of client clustering include Web caching, server replication, content distribution, server-based access prediction, traffic engineering and load balancing. In this thesis, we discuss two applications: Web caching and server replication.

Web caching

As a primary application of clustering, we examine Web caching in this thesis. Caching popular objects at locations close to the clients has been recognized as one of the effective solutions to alleviate Web service bottlenecks, reduce traffic over the Internet and improve the scalability of the WWW system. Trace-driven simulation using Web server logs has been widely used in designing and evaluating Web caching systems. A proxy acts as a server to clients and as a client to origin servers. A primary role played by a proxy is to cache frequently accessed resources to reduce latency for future accesses. In order to determine the optimal places to install proxies, identifying the group of clients that are responsible for a large portion of the requests sent to the Web server is very important. Previous work [SCA99, ZQK00] either use a simple heuristic to obtain client clusters (e.g., the 24-bit heuristic) or rely on a sampling scheme of clients. Neither of them does a good job in capturing the client request pattern and its geographical distribution. The

network-aware client cluster is useful for Web servers to identify client clusters that send a lot of requests, monitor access patterns of client clusters, identify unusual access patterns such as spiders, determine optimal placement of proxies (and/or server replicas), and data movement among replicated servers/proxies, etc.

In this thesis, we examine how to use network-aware clustering in a Web caching simulation and contrast it with the simple approach (i.e., the 24-bit heuristic). We also present an approach to identifying hosts with unusual access patterns, such as spiders[2] and proxies, and then describe proper placement of proxies between clients and servers.

Server replication

As the second example, we examine a typical application of clustering in server replication: automatic request redirecting among mirror servers. To accommodate users' demands, a Web site usually replicates its content at several places (mirror servers) to lower load on the main server and to reduce user access latency. User requests redirecting among the mirror servers will help reduce the user access latency as well as balance the load among the servers. Without a client clustering scheme, requests redirection is conducted on a per client basis. Network-aware client clustering allows clients belonging to the same cluster to cooperate. Moreover, cluster-based request redirecting will also reduce the overhead of redirecting requests at the server.

In this thesis, we propose a scheme that can be deployed at Web servers to automatically redirect client requests to one of the good servers to reduce the

[2]A spider is a program whose task is to obtain all the resources on a large number of sites, primarily for the purpose of generating an inverted index to be used in a search application.

access latency perceived by clients. We select good servers for a client cluster based on the "follow-the-herd" heuristic. We also propose a mechanism to dynamically reconstruct a busy cluster list. We conduct experiments on a set of server logs of a digital library site. *Traceroute* results show that our selection of good servers usually have a smaller RTT to the client clusters than that of other servers.

1.2.3 Improvements

We improve our basic clustering approach in four distinct ways.

1. We examine if busy clusters can be combined into a super-cluster based on the *Autonomous Systems* (AS)[3] that originates the prefix of cluster's IP addresses [Hal97];

2. We examine if small (non-busy) clusters can be combined into a super-cluster if they originate from the same AS;

3. We examine if large clusters can be sub-clustered based on their access pattern;

4. We examine if server clustering is possible as well.

1.3 Thesis overview

After examining related work in Chapter 2, Chapter 3 describes the network-aware approach to identifying client clusters. The thesis presents the basic scheme as

[3]The Internet consists of a large collection of hosts connected by networks of links and routers. It is divided into thousands of distinct regions of administrative control, referred to as *Autonomous Systems* (AS), ranging from college campuses and corporate networks to large Internet Service Providers (ISPs).

well as the experimental results on varied Web server logs. Chapter 4 addresses applications of client clustering. We use Web caching and server replication as two examples to illustrate the usefulness of client clustering. Chapter 5 presents four new cluster-related ideas. Finally, Chapter 6 concludes the thesis by discussing potential improvements to our work.

Chapter 2

Related work

In this chapter, we examine work related to our own. We have divided this into three classes: IP address clustering, Web caching, and content distribution. The following sections discuss each of these in turn.

2.1 IP address clustering

We are not aware of any work that uses BGP [Hal97] routing information to do clustering of clients for the range of applications we have examined. Simply using *nslookup* to do clustering is expensive. It requires to query DNS servers for each IP address. In addition, the *nslookup*-based approach is unlikely to yield full results. This is partially because not all the IP addresses can be resolved from querying DNS servers. Even if for those IP addresses that are resolvable, grouping IP addresses into clusters based on their name will yield inaccurate results. For example, all the clients on the AT&T network have a name of the format $*.att.com$. However, they could be very far away from each other and belong to different clusters.

A clustering model based on distance between servers and clients [BC96] was used to lower the cost of document dissemination. The server classifies documents into globally, remotely, and locally popular, based on the remote-to-local access ratio. In [BC96], information is disseminated from *home servers* to *service proxies* closer to *clients* with the assumption that there is a many-to-many mapping between home servers and service proxies. A service proxy along with the set of home servers it represents form a *cluster*. The Internet is modeled as a hierarchy of such clusters. In [BC96], the clustering is based on the structure of the routes between a server and its clients. A server-to-client routing tree is constructed based on *traceroute*, on which the dissemination strategy is performed. Replicas of the most popular files are disseminated down to the proxies closer to the clients.

Besides [BC96], [KRS00] also made use of *traceroute* although for a different purpose. Using *traceroute* along the lines of [KRS00] or [BC96] is more expensive than using routing table information. In particular, it is not feasible for using it under real-time considerations.

2.2 Web caching

The World Wide Web can be considered as a large distributed information system that provides access to shared data objects. The estimated size of the World Wide Web is shown in Figure 2.1 [BB]. As the World Wide Web continues its exponential growth (the number of static Web pages increases approximately 15% per month), two of the major problems that Web users are suffering today are network congestion and server overloading. The rapid growth of WWW could be attributed to the fact that, at least till now, its usage is quite inexpensive, and accessing

information is fast and easy. Also, the World Wide Web has documents that appeal to a wide range of interests, e.g., news, education, scientific research, sports, entertainment, stock market, travel, shopping, weather, maps, etc. Although the Internet backbone capacity increases 60% per year, the demand for bandwidth is likely to outstrip supply in the foreseeable future as more and more information services are moved onto the Web. If some kind of solution is not undertaken for its rapidly increasing growth, it will be too congested and its entire appeal may be lost.

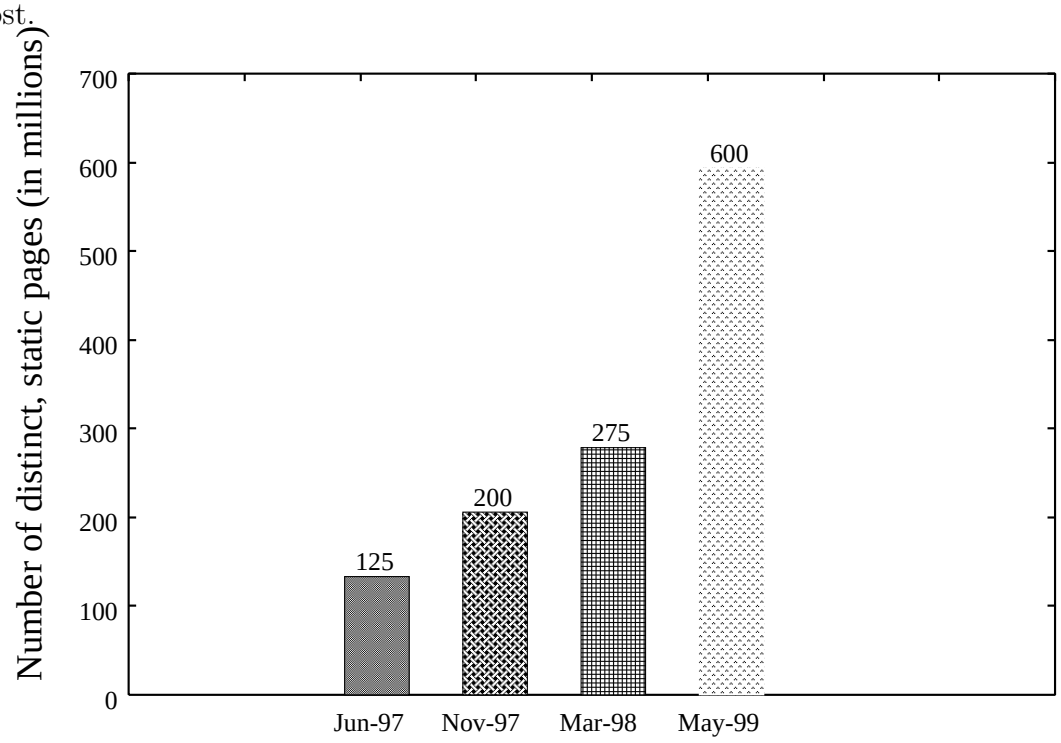


Figure 2.1: Number of distinct, static Web pages.

Researchers have been working on improving Web performance since the early 90's. Caching popular objects close to the clients has been recognized as an effective solution to alleviate Web server bottlenecks, reduce traffic over the Internet, and improve scalability.

Documents can be cached on clients, proxies, and servers. The effects of Web

caching are two-fold. First, it has been shown that caching documents can improve Web performance significantly [CDF+98] [DMF] [KLM97]. The advantages of Web caching are:

1. It reduces bandwidth consumption, thereby decreasing network traffic and congestion.

2. It reduces access latency because:

   (a) By fetching frequently accessed documents from a nearby proxy cache, the access latency is minimized.

   (b) Because of reduction in network traffic, documents that are not cached can be retrieved faster.

3. It reduces the workload of the remote Web server by disseminating data among the proxy caches over the wide area network.

4. If the remote server is not available due to a remote server's crash or network partitioning, the client can access a cached copy at proxy. Thus, the robustness of the Web service is enhanced.

5. A side effect of Web caching is that it provides an opportunity to analyze an organization's usage patterns.

Furthermore, it has been shown that the cooperation of proxy caches is a powerful paradigm to improve cache effectiveness where caches cooperate both in serving each other's requests and in making storage decisions (e.g. object placement and replacement) [KD99] [KS98]. However, note that there are several disadvantages of using a caching system in Web services:

1. A client might be looking at stale data due to a lack of proper updating.

2. The access latency may increase in the case of a cache miss. Hence, cache hit rate should be maximized and the cost of a cache miss should be minimized when designing a caching system.

3. A single proxy cache is always a bottleneck. A limit has to be set to the number of clients it can service, so that the mechanism is at least as efficient as when directly contacting with the remote servers.

4. A single proxy is a single point of failure.

5. Using a proxy cache will reduce access to the original remote server, making it impossible for information providers to maintain a true log of the hits to their pages. Hence, they might decide not to allow their documents to be cacheable.

2.2.1 Elements of a World Wide Web caching system

Besides the obvious goals of a Web caching system, we would like a Web caching system to have a number of properties. They are fast access, robustness, transparency, scalability, efficiency, adaptivity, stability, load balancing, ability to deal with heterogeneity, and simplicity [Wan99]. A generic model of a Web caching system is shown in Figure 2.2. In such a caching system, documents can be cached at clients, proxies, and servers. A client always requests a page from its local proxy if it does not have a valid copy in its own browser's cache. Upon receiving a request from a client, the proxy first checks to see if it has the requested page. If so, it returns the page to the client. If it does not have the requested page in its cache, it sends a request to its cooperative proxies or the server. On receiving a request

from another proxy, the proxy checks if it has the requested page. If so, it returns the page to the proxy issuing the request. If not, the proxy may further forward the request to other proxies or the server. If none of the cooperative proxies has the page, the proxy will fetch the page from the server.

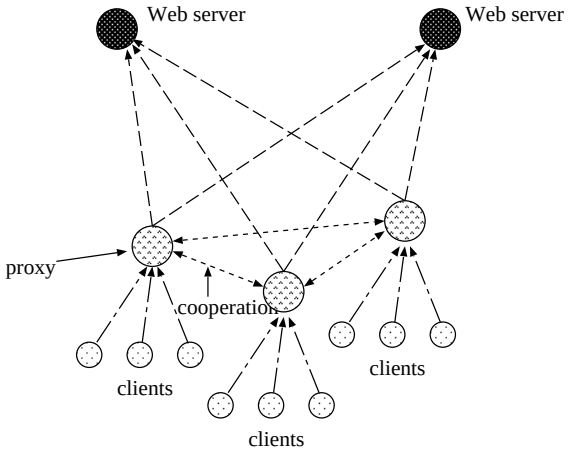


Figure 2.2: A generic WWW caching system.

In order to make a WWW caching system work, the following problems need to be solved:

- How are the cache proxies organized: hierarchically, distributed, or hybrid? (caching system architecture)
- Where is a cache proxy placed in order to achieve optimal performance? (proxy placement)
- What can be cached in the caching system: data, connection, and computation? (caching content)
- How do proxies co-operate with each other? (proxy co-operation)
- What kind of data/information can be shared among co-operative proxies? (data sharing)

- How does a proxy decide where to fetch a requested page? (cache resolution/routing)
- How does a proxy decide what and when to prefetch from a Web server or other proxies to reduce access latency in the future? (prefetching)
- How does a proxy decide which pages to store in its cache and which ones to remove? (cache placement and replacement)
- How does the proxy maintain data consistency? (cache coherency)
- How is the control information (e.g. URL routing information) distributed among proxies? (control information distribution)
- How does the system deal with data that is not cacheable? (dynamic data caching)

These questions must be addressed in every proposed caching system. Depending upon the choices made in answering each question, a variety of schemes have been proposed and these are discussed next.

2.2.2 Caching architectures

The performance of Web caches depends on the size of the client community connected to it; the more people use it, the higher the probability that a given document has already been requested and is present in the cache. Caches cooperate to increase the probability of a document hit. A caching architecture should provide a paradigm that assists proxies to cooperate efficiently with each other.

In order to achieve this goal, caches could be set up either hierarchically [CDN+96] [MNR+98] [YWMW], distributed [FCAB98] [GRC97] [PH97] [REL98]

[RW98] [TDVK98] [Wan97] [WC], or in a hybrid way [RCG98] [WC]. The main performance metric of caching architectures is the expected latency to retrieve a Web document. It is debatable which caching architecture can achieve optimal performance. Recent research [RSB99] shows that hierarchical caching has shorter connection times than distributed caching, and hence, placing additional copies at intermediate levels reduces the retrieval latency for small documents. It has also been shown that distributed caching has shorter transmission times and higher bandwidth usage than hierarchical caching. A "well configured" hybrid scheme can combine the advantages of both hierarchical and distributed caching, reducing the connection time as well as the transmission time.

2.2.3 Cache resolution/routing

Scalability and deployment concerns have led most designers of Web caching infrastructures to design schemes based on deploying a large number of Web caches scattered around the Internet. The main challenge in such approaches is being able to quickly find a cache containing the desired document. While this problem is similar to the general problem of network routing, it cannot be solved in the same way. Conventional routing protocols scale because of the route aggregation made possible by hierarchical addressing. However, since documents with the same URL prefix or server address prefix will not necessarily be delivered to the same clients, there is no necessary correspondence among their cache locations. With no way to aggregate routes, the cache routing tables would be unmanageably large. In addition, they have to be updated frequently. This leads to cache misses when proxies do not have updated cache routing information. In order to minimize the cost of a cache miss, an ideal cache routing algorithm should route requests to the

next proxy which is believed to cache the desired document along (or close to) the path from the client to the Web server.

The common approach is to grow a caching distribution tree away from each popular server towards sources of high demand and do cache resolution either via a *cache routing table* [CDN+96] [LG98] [MLB95] [MNR+98] [PH97] [Wan97] [WC] or via *hash functions* [Blo70] [FCAB98] [KLL+97] [VR].

Harvest cache system [CDN+96] uses a cache routing table to do cache routing. Caches are organized in a hierarchy and use a cache resolution protocol called Internet Cache Protocol (ICP) [WC]. Requests for Web documents are forwarded up the hierarchy in search of a cached copy. In attempt to prevent overloading caches at the root, caches query their siblings before passing requests upwards.

Adaptive Web Caching [MNR+98] also uses a cache routing table to help cache routing. In particular, it uses a mesh of caches in which distribution trees for each server are built. The caches in the mesh are organized into overlapping multicast groups through which a request travels in search of a cached document. This scheme benefits from constructing different distribution trees for different servers (so no root node will be overloaded) and being robust and self-configuring. For less popular objects, queries travel through many caches, and each check requires a query to and responses from a group of machines. The authors suggest dealing with this problem by limiting the number of caches a request will access.

The Cache Array Routing Protocol (CARP) [VR] uses a hashing function to facilitate cache routing. It allows for "queryless" distributed caching by using a hash function based upon the "array membership list" and URL to provide the exact cache location of an object, or where it will be cached upon downloading from the Internet. When one proxy server is added or removed, $1/n$ URLs need

to be reassigned and the new hash function need to be distributed among proxies, where n is the number of proxy servers.

Another example of using a hashing function in cache routing is the Summary cache [FCAB98]. In a Summary cache, each proxy keeps a summary of the URLs of cached documents at each participating proxy and checks these summaries for potential hits before sending any queries. To reduce overhead, the summaries are stored as a Bloom filter [Blo70] and updated only periodically. Experiments have shown that Summary caches reduce the number of inter-cache protocol messages, bandwidth consumption, and protocol CPU overhead significantly while maintaining almost the same cache hit ratio as ICP (Internet Caching Protocol) [WC].

These approaches work well for requests for very popular documents, because these documents will propagate out to many caches, and so will be found quickly. For less popular documents, the search may follow a long and circuitous path of numerous failed checks. The impact of this is substantial since the hit rate on Web caches is typically less than 50%, indicating a large number of documents of only low to moderate popularity [ASA+95].

2.2.4 Prefetching

Prefetching can be applied in three ways in the Web:

1. Between browser clients and Web servers ([BC96] [CB98] [CJ97] [PM99] [PM96]).

2. Between proxies and Web servers ([CKR98b] [CY97] [GS94] [KLM97] [MC98]).

3. Between browser clients and proxies ([FCLJ99] [LB97]).

Between browser clients and Web servers

Early studies focus on prefetching schemes between browser clients and Web servers. Padmanabhan and Mogul [PM96] analyze the latency reduction and network traffic of prefetching using Web server traces and trace-driven simulation. The prediction algorithm they used is based on the Prediction-by-Partial-Matching (PPM) data compressor with a prefix depth of 1. The study shows that prefetching from Web servers to individual clients can reduce client latency by 45% at the expense of doubling the network traffic. Bestavros and Cunha [BC96] present a model for speculative dissemination of World Wide Web documents. The work shows that reference patterns observed at a Web server can be used as an effective source of information to drive prefetching, and reaches similar results as [PM99]. Cunha and Jaccoud use [CJ97] a collection of Web client traces and study how effectively a user's future Web accesses can be predicted from his or her past Web accesses. They show that a number of models work well and can be used in prefetching. Crovella and Barford [CB98] analyzed the network effects of prefetching and show that prefetching can reduce access latency at the cost of increasing network traffic and increasing network traffic burstiness (and thereby increasing network delays). They proposed a rate-controlled prefetching scheme to minimize the negative network effect by minimizing the transmission rate of prefetched documents. However, these early studies do not consider or model caching proxies and hence fail to answer the question about the performance of prefetching completely.

Between proxies and Web servers

After proxies were deployed to improve Web access, research interest has shifted to investigating prefetching techniques between proxies and Web servers. Gwertzman

and Seltzer [GS94] discuss a technique called Geographical Push-Caching where a Web server selectively sends it documents to the caches that are closest to its clients. The focus of the study is on deriving reasonably accurate network topology information and using the information to select caches. Kroeger et al. [KLM97] investigate the performance limits of prefetching between Web servers and proxies, and show that combining perfect caching and perfect prefetching at the proxies can at least reduce the client latency by 60% for high bandwidth clients. Markatos and Chronaki [MC98] propose that Web servers regularly push their most popular documents to Web proxies, which then push those documents to the clients. They evaluate the performance of the strategy using several Web server traces and find that this technique can anticipate more than 40% of a client's request. The technique requires cooperation from the Web servers. The study does not evaluate client latency reduction from the technique. Cohen et al. [CKR98b] also investigate similar techniques. Wcol [CY97] is a proxy that prefetches documents, links, and embedded images. The proxy, however, does not push the documents to the client.

Between browser clients and proxies

Prefetching can also be done between browser clients and proxies. Loon and Bharghavan [LB97] proposed a design and an implementation of a proxy system that performs prefetching, image filtering, and hoarding for mobile clients. Fan et al. [FCLJ99] proposed an approach to reduce latency by prefetching between caching proxies and browsers. The approach relies on the proxy to predict which cached documents a user might reference next (based on the PPM data compressor), and takes advantage of idle time between user requests to either push or pull

the documents to the user. Simulation results show that prefetching combined with large browser cache and delta-compression can reduce client latency up to 25%.

Discussion

Although Web performance is improved by caching documents at proxies, the benefit from this technique is limited [DFKM97] [KLM97]. Previous research shows that the maximum cache hit ratio that can be achieved by any caching algorithm is usually no more than 50%, that is, regardless of the caching scheme in use, one out of two documents cannot be found in the cache [ASA+95]. One way to further increase the cache hit ratio is to anticipate future document requests and preload or prefetch these documents in a local cache.

2.2.5 Cache placement/replacement

A key aspect of the effectiveness of proxy caches is a document placement and replacement algorithms that can yield a high hit rate. While cache placement has not been well studied, a number of cache replacement algorithms have been proposed in recent studies, which attempt to minimize various cost metrics, such as hit ratio, byte hit ratio, average latency, and total cost. The approaches can be classified into the following three categories [AWY99].

1. Traditional replacement policies and its direct extensions:

    - *Least Recently Used* (LRU) evicts the object which was requested least recently.

- *Least Frequently Used* (LFU) evicts the object which is accessed least frequently.
- *Pitkow/Recker* [WAS+96] evicts objects in LRU order, except if all objects are accessed within the same day, in which case the largest one is removed.

2. Key-based replacement policies: (i.e., the replacement policies in this category evict objects based upon a primary key. Ties are broken based on secondary key, tertiary key, etc.)
   - *Size* [WAS+96] evicts the largest object.
   - *LRU-MIN* [ASA+95] biased in favor of smaller objects. If there are any objects in the cache which have size being at least S, LRU-MIN evicts the least recently used one from the cache. If there are no objects with size being at least S, then LRU-MIN starts evicting objects in LRU order of size being at least $S/2$. That is, the object who has the largest log(size) and is the least recently used object among all objects with the same log(size) will be evicted first.
   - *LRU-Threshold* [ASA+95] is the same as LRU, but objects larger than a certain threshold size are never cached.
   - *Hyper-G* [WAS+96] is a refinement of LFU, breaking ties using the recency of last use and size.
   - *Lowest Latency First* [WA97] minimizes average latency by evicting the document with the lowest download latency first.
3. Cost-based replacement policies: (i.e., the replacement policies in this category employ a potential cost function derived from different factors such as

time since last access, entry time of the object in the cache, transfer time cost, object expiration time and so on.)

- *GreedyDual-Size* (GD-Size) associates a cost with each object and evicts the object with the lowest cost/size.
- *Hybrid* [WA97] associates a utility function with each object and evicts the one that has the least utility to reduce the total latency.
- *Lowest Relative Value* [LRV] evicts the object with the lowest utility value.
- *Least Normalized Cost Replacement* (LCN-R) [SSV97] employs a rational function of the access frequency, the transfer time cost and the size.
- *Bolot/Hoschka* [BH96] employs a weighted rational function of transfer time cost, the size, and the time of last access.
- *Size-Adjusted LRU* (SLRU) [AWY99] orders the object by ratio of cost to size and chooses objects with the best cost-to-size ratio.
- *Server-assisted* scheme [CKR98a] models the value of caching an object in terms of its fetching cost, size, next request time, and cache prices during the time period between requests. It evicts the object of the least value.
- *Hierarchical GreedyDual* (Hierarchical GD) [KD99] does object placement and replacement cooperatively in a hierarchy.

To summarize, great efforts have been made to maximize cache hit ratio. However, the performance of replacement policies highly depends on the traffic characteristics of WWW accesses. No known policy can outperform others for all Web

access patterns. As the cost of disk space becomes lower, cache replacement has been considered as a less serious issue.

2.2.6 Cache coherency

Caches provide decreased access latency at a cost: every cache will sometimes provide users with *stale* pages - pages which are out of date with respect to their master copies on the Web servers where the pages originated [DP96]. Every Web cache must somehow update pages in its cache so that it can give users pages which are as fresh as possible. Caching and the problem of cache coherency on the World Wide Web are similar to the problems of caching in distributed file systems. However, the Web is different than a distributed file system in its access patterns, its larger scale, and its single point of updates for Web objects [GS96]. Current cache coherency schemes provide two types of consistency, *strong cache consistency*[1], which includes client validation and server invalidation [CL97], and *weak cache consistency*[2], which includes adaptive TTL [Cat92] [CDN+96] [DP96] [GS94] [LA94] [Wes95] and piggyback validation/invalidation [CKR98b] [KW97] [KW98] [KW99].

1. Strong cache consistency

   (a) *Client validation.* This approach is also called *polling-every-time.* The proxy treats cached resources as potentially out-of-date on each access and sends an *If-Modified-Since* header with each access of the resources. This approach can lead to many 304 responses (HTTP[HTTa, HTTb]

[1]Cached data is always consistent with its original copy at the server.
[2]Cached data may be stale.

response code for "*Not Modified*") by server if the resource has not actually changed.

(b) *Server invalidation.* Upon detecting a resource change, the server sends invalidation messages to all clients that have recently accessed and potentially cached the resource [CL97]. This approach requires a server to keep track of lists of clients to use for invalidating cached copies of changed resources and can become unwieldy for a server when the number of clients is large. In addition, the lists themselves can become out-of-date causing the server to send invalidation messages to clients who are no longer caching the resource.

2. Weak cache consistency

(a) *Adaptive TTL.* The adaptive TTL (also called Alex protocol [Cat92]) handles the problem by adjusting a document's time-to-live based on observations of its lifetime. Adaptive TTL takes advantage of the fact that file lifetime distribution tends to be bimodal; if a file has not been modified for a long time, it tends to stay unchanged. Thus, the time-to-live attribute to a document is assigned to be a percentage of the document's current "age", which is the current time minus the last modified time of the document. Studies [Cat92] [GS94] have shown that adaptive TTL can keep the probability of stale documents within reasonable bounds ($< 5\%$). Most proxy servers (e.g. CERN *httpd* [LA94] [Wes95]) use this mechanism. The Harvest cache [CDN+96] mainly uses this approach to maintain cache consistency, with the percentage set to 50%. However, there are several problems with this expiration-based coher-

ence [DP96]. First, user must wait for expiration checks to occur even though they are tolerant to the staleness of the requested page. Second, if a user is not satisfied with the staleness of a returned document, they have no choice but to use a *Pragma:No-Cache* request to load the entire document from its home site. Third, the mechanism provides no strong guarantee towards document staleness. Fourth, users cannot specify the degree of staleness they are willing to tolerate. Finally, when the user aborts a document load, caches often abort a document load as well.

(b) *Piggyback validation/invalidation.* Krishnamurthy et al. propose piggyback invalidation mechanisms to improve the effectiveness of cache coherency [CKR98b] [KW97] [KW98] [KW99]. Three invalidation mechanisms are proposed. The Piggyback Cache Validation (PCV) [KW97] capitalizes on requests sent from the proxy cache to the server to improve coherency. In the simplest case, whenever a proxy cache has a reason to communicate with a server it piggybacks a list of cached, but potentially stale, resources from that server for validation. The basic idea of the Piggyback Server Invalidation (PSI) mechanism [KW98] is for servers to piggyback on a reply to a proxy, the list of resources that have changed since the last access by this proxy. The proxy invalidates cached entries on the list and can extend the lifetime of entries not on the list. They also proposed a hybrid approach which combines the PSI and the PCV techniques to achieve the best overall performance [KW99]. The choice of the mechanism depends on the time since the proxy last requested invalidation for the volume [CKR98b]. If the time is small, then the PSI mechanism is used, while for longer gaps the PCV

mechanism is used to explicitly validate cache contents. The rationale is that for short gaps, the number of invalidations sent with PSI is relatively small, but as the time grows longer the overhead for sending invalidation will be larger than the overhead for requesting validations.

2.2.7 Connection caching and computation caching

The proxy cache has been recognized as an effective mechanism to improve Web performance. A proxy may serve in three roles: *data cache*, *connection cache*, and *computation cache*. A recent study shows that caching Web page at proxy reduces the user access latency 3% - 5%. In presence of P-HTTP[HTTb], a proxy can be used as a connection cache. By using persistent connections between clients and proxy and between proxy and Web server, the total latency improvements grow substantially (20% - 40%) [CDF+98] [FCD+99].

Computation caching can be viewed as that the Web server replicates and migrates some of its services to the proxies to alleviate the server bottleneck. One application of such computation caching is dynamic data caching. The motivation is that, in the presence of current caching schemes, the hit ratio at a proxy is at most 50%, which is limited by the fact that a significant percentage of Web pages is dynamically generated and therefore not cacheable. Computation caching can be used to improve the performance to retrieve dynamic data by caching dynamic data at proxies and migrating a small piece of computation to proxies [CID99] [CZB98] [LAISD99] to generate or maintain cached data.

2.2.8 User access pattern prediction

A proxy's policies for managing cached resources (i.e., prefetching, coherence, placement and replacement) and TCP connections rely on assumptions about client access patterns. To improve information exchanges between Web servers and proxies, a variety of techniques have been proposed to predict future requests. One approach is to group resources that are likely to be accessed together, based on the likelihood that pairs of resources are accessed together and server file system structure, etc [CKR99] [YWMW]. Others are using the Prediction by Partial Match (PPM) model to predict which pages are likely to be accessed in the near future [FCLJ99] [PM99] [PM96].

2.2.9 Load balancing

Many have experienced the hot spot phenomenon in the context of the Web. Hot spots occur any time a large number of clients wish to simultaneously access data or get some services from a single server. If the site is not provisioned to deal with all of these clients simultaneously, service may be degraded or lost. Several approaches to overcoming hot spots have been proposed. Most use some kind of replication strategy to store copies of hot pages/services throughout the Internet; this spreads the work of serving a hot page/service across several servers (i.e., proxies) [CDN+96] [HMY] [MLB95] [Rab].

2.2.10 Proxy placement

The placement of proxies is also important to achieve optimal Web performance. The desirable properties of such proxy placement policies include self-organization,

efficient routing, efficient placement, load balancing, and stable behavior. However, little study has been done to address this issue. Li et al. [LGI+99] attempts to solve it based on the assumption that the underlying network topology is a minimum spanning tree and model it as a dynamic programming problem.

2.2.11 Dynamic data caching

As mentioned before, the benefit of current Web caching schemes is limited by the fraction of accessed data that is cacheable. Non-cacheable data (i.e., personalized data, authenticated data, dynamically generated data, etc.) is a significant percentage of the total data. For example, measurement results show that 30% of user requests contain cookies [CDF+98] [FCD+99]. The results in [WM99] indicate that assumptions made in previous work about the uncacheability of responses with cookies may not be valid [CDF+98, FCD+99]. How to make more data cacheable and how to reduce the latency to access non-cacheable data have become crucial problems in order to improve Web performance. Current approaches can be classified into two categories: *active cache* and *server accelerator*.

Active cache [CZB98] supports caching of dynamic documents at Web proxies by allowing servers to supply cache applets to be attached with documents and requiring proxies to invoke cache applets upon cache hits to finish the necessary processing without contacting the server. Active cache can result in significant network bandwidth savings at the expense of CPU cost. However, due to the significant CPU overhead, the user access latencies are much larger than that without caching dynamic objects.

A Web server accelerator [LAISD99] resides in front of one or more Web servers to speed up user accesses. It provides an API which allows application programs

to explicitly add, delete, and update cached data. The API allows the accelerator to cache dynamic as well as static data. Invalidating and updating cached data is facilitated by the Data Update Propagation (DUP) algorithm which maintains data dependence information between cached data and the underlying data which affect their values in a graph [CID99].

2.2.12 Web traffic characteristics

Understanding the nature of the workloads and system demands created by users of the World Wide Web is crucial to properly designing and provisioning Web services. The effectiveness of caching schemes relies on the presence of temporal locality in Web reference streams and on the use of appropriate cache management policies appropriate for Web workloads. A number of measurements have been done to exploit the access properties at clients, proxies, and servers [AFAW] [AFJ99] [BBBC99] [BCF+99] [DFKM97] [DMF].

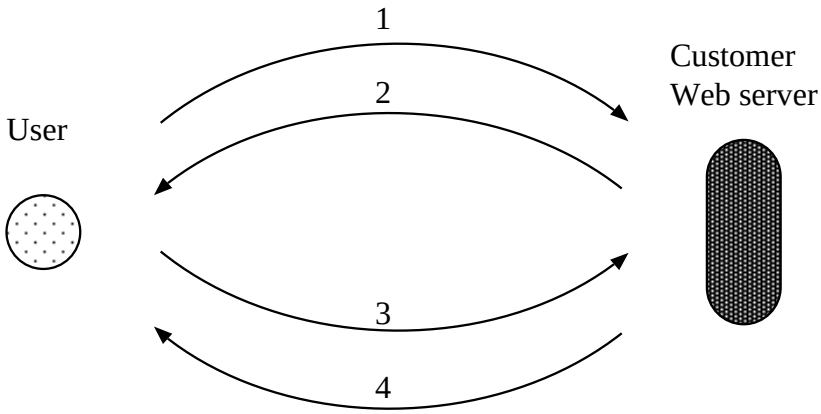
2.2.13 Summary

As the Web becomes more popular, users are increasingly suffering from network congestion and server overloading. Significant efforts have been made to improve Web performance. Web caching is recognized to be one of the effective techniques to alleviate server bottlenecks and reduce network traffic, thereby minimizing the user access latecny. In this section, we give an overview of recent Web caching schemes. In surveying previous work on Web caching, we notice that there are still some open problems in Web caching such as proxy placement, cache routing, dynamic data caching, fault tolerance, security, etc. The research frontier in Web performance improvement lies in developing an efficient, scalable, robust, adaptive,

and stable Web caching scheme that can be easily deployed.

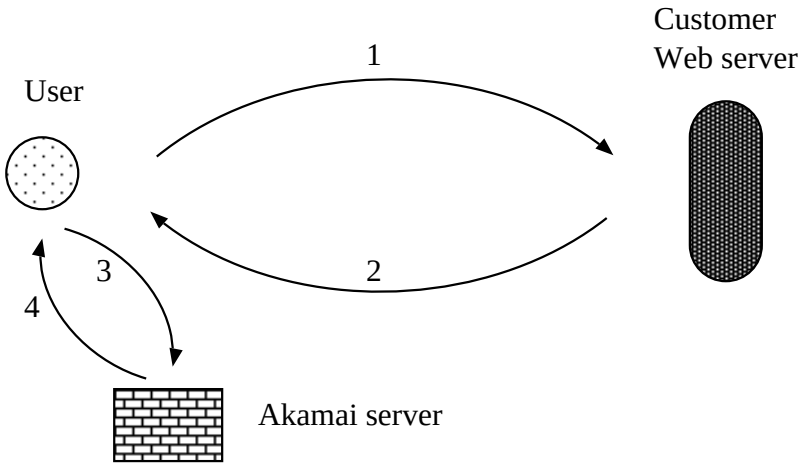
2.3 Content distribution and load balancing

Several efforts are underway at research and production levels in improving content distribution. At the DNS layer some companies (e.g., Akamai [aka], Digital Island [dig], Adero [ade]) use a client's location to serve content from a nearby site to lower load on the server and the perceived latency. For example, Akamai's *FreeFlow* [aka] delivers content over Akamai's global network. Akamai servers are distributed throughout the world at the edges of the Internet, so they are geographically close to end-users. FreeFlow continuously monitors Internet conditions and determines the best delivery route and optimal "edge server" for each Web site request. FreeFlow Akamaizer automatically tags the specified content for delivery from the Akamai network. When an Akamaized Web site is requested, the site serves the Web page while FreeFlow delivers objects within the page from an Akamai server (Figures 2.3 and 2.4). Akamai could use the clustering methodology to automatically monitor users' demand and to install Akamai servers on its network. It may also help to perform load balancing among Akamai servers.



1. User enters URL
2. Web server returns HTML with embedded URLs
3. User's browser requests embedded objects
4. Rich content served

Figure 2.3: Internet content delivery without Akamai's Freeflow.



1. User enters standard URL
2. Web server returns HTML with embedded URLs pointing to Akamai Network
3. User's browser requests embedded objects
4. Rich content served locally

Figure 2.4: Internet content delivery with Akamai's Freeflow.

Chapter 3

A Network-aware approach to client clustering

We present a novel method to identify client clusters by using the prefixes and netmasks information extracted from the BGP (Border Gateway Protocol [Hal97, LR90, Moy98]) routing and forwarding table snapshots. In this chapter, we first give a overview of BGP and the rationale behind the network-aware approach to clustering. Next, we present the network-aware approach and show some experimental results.

3.1 BGP

The Internet consists of a large collection of hosts connected by networks of links and routers. It is divided into thousands of distinct regions of administrative control, referred to as *Autonomous Systems* (AS), ranging from college campuses and corporate networks to large Internet Service Providers (ISPs). Within the boundaries of each AS, Interior Gateway Protocols (IGPs), such as RIP and OSPF,

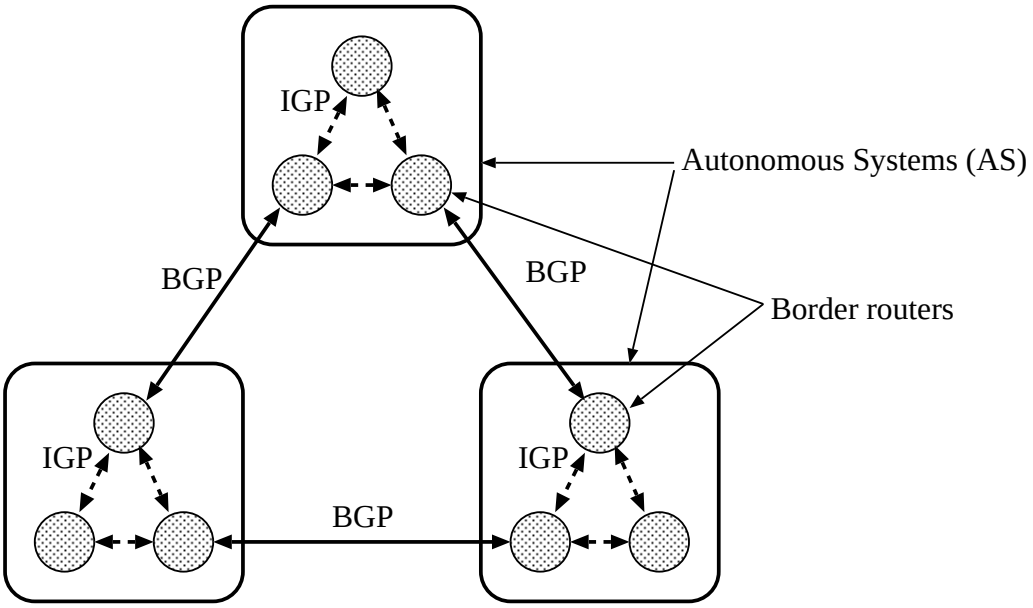


Figure 3.1: General illustration of AS relationships.

are used to route datagrams. Then different ASs are interconnected via an Exterior Gateway Protocol (EGP) to form the global internetworking. Currently the de facto EGP in the Internet is the Border Gateway Protocol version 4 (BGP4) (see Figure 3.1).

EGPs were introduced because IGPs are not suitable for global internetworking. First, IGPs do not scale well for large networks. Their routing table structure and control information exchanging mechanism make IGPs unusable in large-scale networks. Secondly, IGPs do not provide the hook for logical separation between organizations. Since individual organizations need to function with enough technical independence, IGPs are inappropriate for operating in multi-organization networks.

EGPs were aimed at what IGPs are not capable of. They segregate routing domains into separate ASs and offer routing service between ASs. EGPs reduce the routing table expansion problem, because the internals of the organizational

networks can be omitted by EGPs. By organizing the Internet into ASs, EGPs allow each AS to have its own routing policies independent of what is used in other ASs. Thus, an individual administration has total control over the routing activities within the organization's network.

From the routing point of view, each AS consists of the set of routers having a single routing policy. At the same time, from an administrative point of view, each AS consists of a set of routers under a single technical administration. To the rest of the world, the whole AS can be viewed as a single network entity.

BGP4's initial deployment started in 1993. Different from any distance vector or link state algorithm, BGP is a path vector protocol. BGP routing information consists of a sequence of AS numbers. This sequence describes the path a data packet will traverse and hence the term "path vector".

BGP uses negotiation between neighboring ASs to establish relationships between BGP peers. The negotiation includes the successful completion of a TCP connection between BGP peers, a successful reception and processing of an OPEN message, and periodical exchange of KEEPALIVE messages. Using TCP connections, BGP routing control messages are exchanged reliably. As soon as the TCP connection is up, a BGP endpoint sends an OPEN message to its peer and waits for an acknowledgement, which could be either in the format of a NOTIFICATION message (failure) or a KEEPALIVE message (success). If a KEEPALIVE message is received, the neighboring relationship between BGP peers is established. From this point on, neighboring BGP peers exchange KEEPALIVE messages periodically to maintain their relationship and monitor each other's well being. If something goes wrong, a NOTIFICATION is sent to the peer BGP entity and the neighbor negotiation is restarted from the TCP connection step.

After the neighboring relationship is confirmed, BGP routers exchange their route information using UPDATE messages. The exchange of UPDATE messages gives BGP enough knowledge to construct loop-free routes. Each UPDATE message contains three blocks: network layer reachability information (NLRI), path attribute, and unreachable routes. The NLRI block indicates the network whose route is advertised in this UPDATE message. The path attribute block is used to keep track of route specific information such as the path information, degree of preference of a route, next hop value of a route, and aggregation information. The last block lists routes that have become unreachable.

3.2 Why BGP?

The rationale behind the network-aware approach is that the prefixes and corresponding netmasks identify the routes in the routing table which are used by core BGP routers to forward packets to a given destination. By examining routing tables from a collection of points in the network, we are likely to see entries that approximately correspond to our notion of a cluster (described in Chapter 1). The most specific entries are most useful to us, since they represent groups of clients that are topologically very close together. Less specific entries (resulting from significant route aggregation) are likely to produce clusters that are too large. By examining multiple routing tables, we hope to get a large number of specific entries that include most of the IP addresses of interest (e.g., IP addresses extracted from Web logs). Our approach makes use of snapshots of routing tables. Although, observed from Table 1.1, these tables will change over time, we find our network-aware approach performs well.

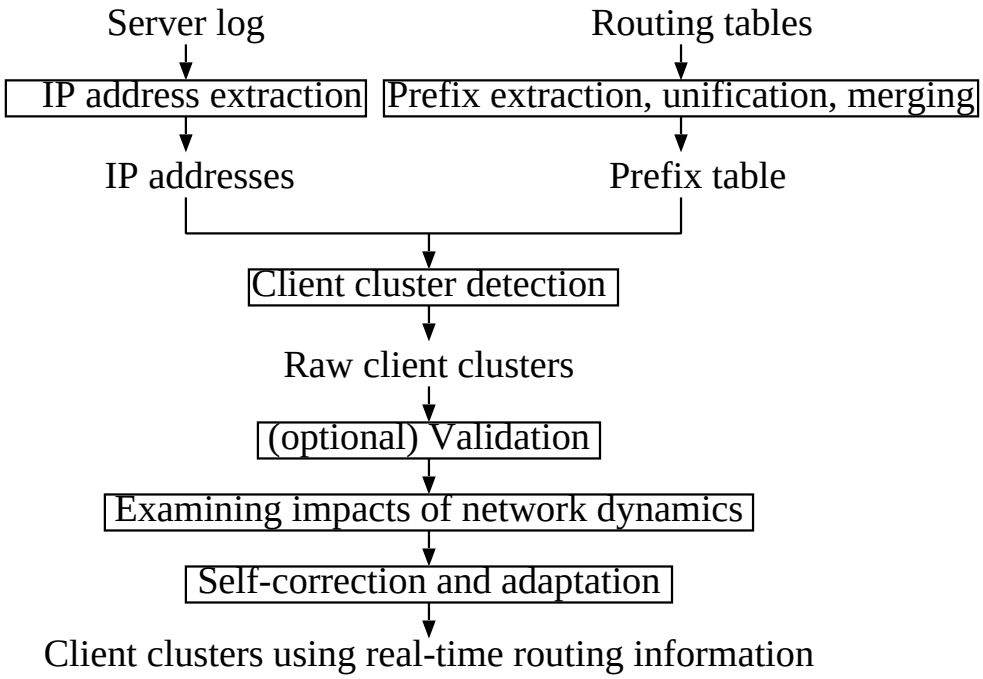


Figure 3.2: The entire automated process of client clustering.

3.3 Overview

Our network-aware approach to client clustering is an entirely automated process starting with the sources of IP addresses such as Web server logs and BGP routing table snapshots. Figure 3.2 provides a graphical view of the five steps.

1. network prefix/netmask extraction and extraction of IP addresses from Web server logs;
2. client cluster identification;
3. (optional) validation of our method by sampling the resulting client clusters;
4. examining the effect of network dynamics on client cluster identification;
5. self-correction and adaptation.

The validation step is optional for most applications; we include it here to provide a quantitative measurement of the accuracy of our approach. We discuss each of these steps in this chapter and also look at other issues affecting client cluster identification.

3.4 Network prefix extraction

The first step of the network-aware approach is to generate a combined prefix/netmask entry table. We do this in three steps.

1. Extract prefix/netmask entries from the routing table snapshots
2. Unify prefix/netmask entries into a standard format
3. Insert all the entries from different routing tables into a single, large table. Taking such a union gives us a more complete picture of the network topology.

3.4.1 Prefix entry extraction

The BGP routing tables (Table 3.1) are collected automatically via a simple script from AADS, MAE-EAST, MAE-WEST, PACBELL, PAIX (all from [merle]), ARIN [ari99], AT&T-Forw and AT&T-BGP [att99], CANET [can99], CERFNET [cer99], NLANR [nla97], OREGON [ore97], SINGAREN [sines], and VBNS [vbnml]. We do so by one of two ways:

- Downloading them from well-known Web sites (e.g., AADS)
- *Telnet*ing to a particular host to run a script to dump routing tables (e.g., OREGON).

Table 3.1: BGP routing tables used in our experiments.

Name	Date	Size	Comments
AADS	12/7/1999	17K	BGP routing table snapshots updated every 2 hours
ARIN	10/1999	300K	IP network dump
AT&T-BGP	12/15/1999	74K	BGP routing table snapshots
AT&T-Forw	4/28/1999	65K	BGP forwarding table snapshots
CANET	12/1/1999	1.7K	Real-time BGP routing table snapshots
CERFNET	9/29/1999	50K	Real-time BGP routing table snapshots
MAE-EAST	12/7/1999	46K	BGP routing table snapshots taken every 2 hours
MAE-WEST	12/7/1999	30K	BGP routing table snapshots taken every 2 hours
NLANR	11/1997	200K	IP network dump
OREGON	12/7/1999	70K	Real-time BGP routing table snapshots
PACBELL	12/7/1999	25K	BGP routing table snapshots updated every 2 hours
PAIX	12/7/1999	10K	BGP routing table snapshots updated every 2 hours
SINGAREN	12/7/1999	68K	Real-time BGP routing table snapshots
VBNS	12/7/1999	1.8K	BGP routing table snapshots updated every 30 minutes

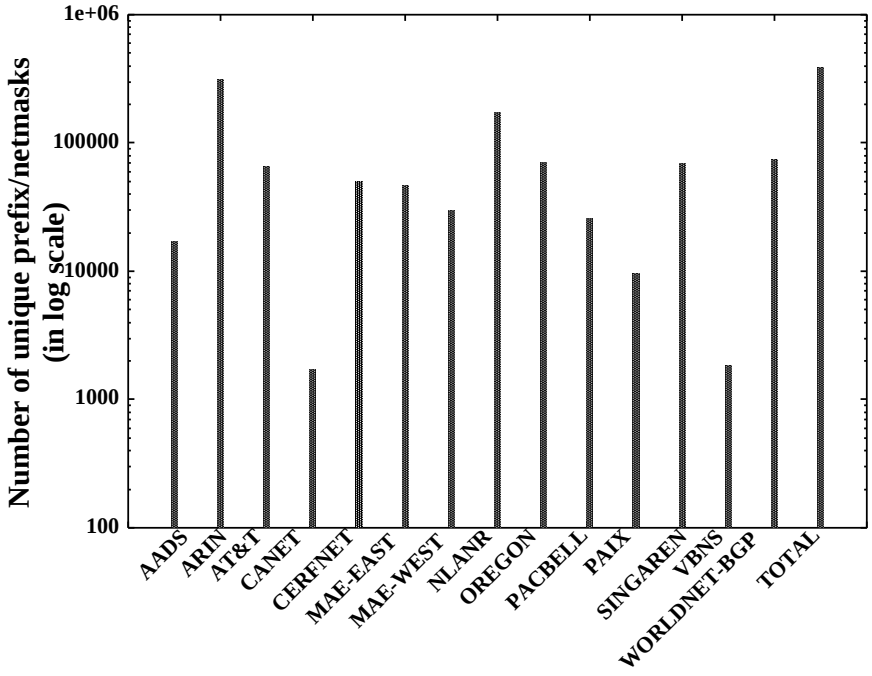


Figure 3.3: Number of unique prefix/netmask entries extracted from routing tables.

The size of the table depends on the location of the BGP router and the AS it belongs to. The number of prefix/netmask entries extracted from each routing table are shown in Table 3.1 and Figure 3.3. In our experiments, we assembled a total of 391,497 unique prefix/netmask entries [1] by December 1999.

- AADS, MAE-EAST, MAE-WEST, PACBELL, PAIX routing tables are near-real-time snapshots (taken every 2 hours) of the complete, "default-free"[2] Internet routing table as seen by the route servers at AADS, MAE-EAST, MAE-WEST, PACBELL, and PAIX NAPs, and contain 17K, 46K, 30K, 25K, and 10K destination networks (prefix entries), respectively.

- ARIN is the October 1999 version of the ARIN `ip_network_dump` file, which

[1] These include prefix entries collected from both BGP routing tables and IP network dumps as explained later in this section.

[2] A default-free table does not contain the default route (e.g. 0.0.0.0/0.0.0.0) in the snapshots.

gives a numerical representation of the networks registered (IP allocation) at the American Registry for Internet Numbers (ARIN). The ARIN tables are the largest collection of network prefixes/netmasks (to the best of our knowledge). There are over 300K unique networks registered at ARIN at the time the dump file was taken.

- AT&-BGP contains a dump of BGP tables from a collection of AT&T WorldNet routers, collected on December 15, 1999. There are over 74K destination networks.

- AT&T-Forw is a set of forwarding table entries collected from AT&T WorldNet on April 28, 1999. It contains 65K destination networks.

- CANET contains CA∗Net II routing tables, which have around 1.7K destination networks. The snapshot was taken on December 1, 1999.

- CERFNET is the real-time dump of BGP routing table at CERFnet, which contains approximately 50K unique destination networks (prefix/netmask). The snapshot was taken on September 29, 1999.

- NLANR is the November 1997 version[3] of the InterNIC's netinfo/ip_network_dump file, which gives a numerical representation of the networks registered (IP allocation) at InterNIC. There were around 200K unique networks registered at InterNIC at the time the dump file was taken.

[3]The most recent version of NLANR's IP network dump is from 1997, but as we only use it as a secondary source of network prefixes in our experiments, its lack of recency has only a minor impact on client cluster results. Only $< 0.1\%$ clients are clustered using network prefixes extracted from the NLANR network dump.

- OREGON provides real-time access to BGP routing dumps. The current participants are ANS (Cleveland), ATT (Chicago), BBNPlanet (Palo Alto), CERFnet (San Diego), DIGEX (MAE-EAST), EBONE (EU), ESnet (GA), RIPE NCC (Amsterdam), IAGnet (Chicago), IIJ (Japan), JINX (Johannesburg), LINX (London), MCI (San Francisco), PIPEX (London), Sprint (Stockton), vBNS (Hayward), Verio (MAE-WEST) and blackrose.org (Ann Arbor). It contains 70K destination networks and is considered to be the largest BGP routing table dump. The snapshots was taken on December 7, 1999.

- SINGAREN contains routing tables as viewed at SingAREN BGP peers: *1net, abilene, apan, esnet, ihpc, imcb, korea, krdl, nanyang, napnet, ncb, nordunet, ntu, nus, startap, stc, surfnet, tanet, temasek, tgbackup*, and *vbns*, which have around 68K destination networks. The snapshot was taken on December 7, 1999.

- VBNS contains all vBNS routes snapshots (about 1.8K destination networks), updated every 30 minutes from `cs.res.vbns.net`. The snapshot we used was taken on December 7, 1999.

An example BGP routing table snapshot is shown in Table 3.2. Though the routing table also contains interdomain information such as the next hop IP address, AS number, and AS path, we have only used the prefix/netmask information in the network-aware approach. The AS number and path information can also provide hints on the geographical location of clients.

Although ARIN and NLANR are the two largest tables, they are not directly generated from BGP information, but from a dump of the networks registered at

Table 3.2: An example snapshot of a BGP routing table (VBNS).

Prefix	Prefix description	Next hop	AS path	Peer AS description
6.0.0.0/8	Army Information Systems Center	cs.ny-nap.vbns.net	7170 1455 (IGP)	AT&T Government Markets
12.0.48.0/20	Harvard University	cs.cht.vbns.net	1742 (IGP)	Harvard University
12.6.208.0/20	AT&T ITS	cs.cht.vbns.net	1742 (IGP)	Harvard University
18.0.0.0/8	Massachusetts Institute of Technology	cs.cht.vbns.net	3 (IGP)	Massachusetts Institute of Technology

these two sites. The difference between a network dump file and a BGP routing table dump file is that the former usually has a much larger collection of networks. An IP address registered/allocated at these two sites may not necessarily exist and be a routable host on the Internet. However, since our client IP addresses are gathered from real Web server logs, this does not affect us. A network entry in a network dump file is usually larger than one in a BGP routing table (i.e., it has a shorter network prefix length) despite the fact that routes may be aggregated in a BGP routing table. This is because an AS that requests IP addresses from ARIN and NLANR sites may further partition and allocate these IP addresses to smaller network entities without the knowledge of ARIN and NLANR. This might cause an error in the network-aware approach if the network prefix is taken from those IP network dumps. However, less than 1% of clients are clustered by network prefixes taken from IP network dump files. We use BGP dump files as the primary source of network prefixes in identifying client clusters and use IP network dump files as a secondary source because each BGP routing table will only have a limited view

of the entire network topology. In our experiments, this improves the proportion of the clustered clients from 99% to 99.9%.

3.4.2 Network prefix format unification

The network prefix/netmask entries extracted from the various routing tables are in one of three different formats as shown in Table 3.3.

1. $x1.x2.x3.x4/k1.k2.k3.k4$: This format is used in routing tables at AADS, MAE-EAST, MAE-WEST, PACBELL, and PAIX, where $x1.x2.x3.x4$ and $k1.k2.k3.k4$ are the network prefix and the netmask with zeroes dropped at the tail. One such example is 193.1/255.255, which corresponds to 193.1.0.0 /255.255.0.0, where 193.1.0.0 and 255.255.0.0 are the network prefix and netmask, respectively.

2. $x1.x2.x3.x4/l$: This format is used in routing tables at ARIN, AT&T, CANET, CERFNET, NLANR, OREGON, SINGAREN, VBNS, and WORLDNET-BGP, where $x1.x2.x3.x4$ is the prefix and l is the netmask length. For example, 128.148.0.0/16 stands for 128.148.0.0/255.255.0.0, where 128.148.0.0 and 255.255.0.0 are network prefix and netmask.

3. $x1.x2.x3.0$: This format, which can be found in CANET, CERFNET, OREGON, SINGAREN, and WORLDNET-BGP is an abbreviated representation of $x1.x2.x3.0/k1.k2.k3.0$ (i.e., $x1.x2.x3.0$ is a block of standard Class A, Class B, or Class C addresses and the network prefix is 8, 16, or 24, respectively). For example, 130.15.0.0 is abbreviation of 130.15.0.0/255.255.0.0.

We unify the different formats, arbitrarily choosing the first format as our standard format and converting all prefix/netmask entries to this format. For instance,

Table 3.3: The formats of network prefix and netmask in routing table snapshots.

Formats	$x1.x2.x3.x4/k1.k2.k3.k4$	$x1.x2.x3.x4/l$	$x1.x2.x3.0$
Routing tables	AADS MAE-EAST MAE-WEST PACBELL PAIX	ARIN AT&T CANET CERFNET NLANR OREGON SINGAREN VBNS WORLDNET-BGP	CANET CERFNET OREGON SINGAREN WORLDNET-BGP
Examples	193.1/255.255 193.0.128/255.255.192	128.148.0.0/16	130.15.0.0 192.75.72.0
Unification	193.1/255.255 193.0.128/255.255.192	128.148/255.255	130.15/255.255 192.75.72/255.255.255

we convert prefix/netmask examples 128.148.0.0/16, 130.15.0.0, and 192.75.72.0 to 128.148/255.255, 130.15/255.255, and 192.75.72/255.255.255, respectively.

3.4.3 Merging network prefix entries

After unifying network prefix entry formats, we measure the routing table coverage by counting the number of unique prefix/netmask entries extracted from each routing table. As Figure 3.3 and Table 3.1 show, some routing tables have a better view of network routes than others (i.e., they contain more prefix/netmask entries) and none of them contain complete information of all the prefixes and netmasks (not all routes are visible to each router). We merge them into a single prefix/netmask table for clustering clients in server logs.

3.5 Client cluster identification

After extracting IP addresses from Web server logs and creating the merged prefix table from routing table snapshots, we identify client clusters in the logs. We now describe our methodology of identifying client clusters and then our experiments on various Web server logs to demonstrate the applicability and generality of our approach.

3.5.1 Methodology

Clustering of clients involves three steps:

1. Extracting client IP addresses from the server log;
2. Performing longest prefix matching (similar to what IP routers do) on each client IP address using the constructed prefix/netmask table;
3. Classifying all the client IP addresses that have the same longest matched prefix into one client cluster, which is identified by the shared prefix.

Suppose we want to cluster the IP addresses 12.65.147.94, 12.65.147.149, 12.65.146.207, 12.65.144.247, 24.48.3.87, and 24.48.2.166. The longest matched prefixes are respectively 12.65.128.0/19, 12.65.128.0/19, 12.65.128.0/19, 12.65.128.0/19, 24.48.2.0/23, and 24.48.2.0/23. We then classify the first four clients into a client cluster identified by prefix/netmask 12.65.128.0/19 and the last two clients into another client cluster identified by prefix/netmask 24.48.2.0/23. Metrics of client clusters, such as the distributions of number of clients in client clusters, number of requests issued from within a client cluster, and number of unique URLs accessed from within a client cluster, can be generated for various applications (described in Chapter 4).

Table 3.4: Overview of some Web server logs used in the experiments.

Log	Dates	Number of requests	Number of clients	Number of unique URLs
Apache	10/1/99 - 11/18/99	3,461,361	274,844	51,536
EW3-a1	7/1/99 - 1/31/99	1,199,276	21,519	683
EW3-a2	6/27/99 - 7/26/99	1,357,623	33,675	11,475
	7/27/99 - 8/26/99	1,645,975	35,249	10,462
	8/27/99 - 9/26/99	2,180,029	44,831	8,569
	10/1/99 - 10/31/99	2,877,528	54,882	10,005
	11/6/99 - 11/30/99	2,223,590	46,579	10,111
	12/1/99 - 12/20/99	1,847,913	38,913	10,556
EW3-b	7/1/99 - 7/31/99	1,908,761	40,156	7,772
EW3-c1	7/1/99 - 7/31/99	1,496,408	160,020	618
EW3-c2	7/1/99 - 7/31/99	148,859	41,456	769
EW3-w	6/27/99 - 7/26/99	575,451	3,659	1,705
	7/27/99 - 8/26/99	3,753,571	16,202	19,210
	8/27/99 - 9/15/99	2,405,524	10,587	8,569
	10/1/99 - 10/31/99	2,297,309	20,788	14,982
	11/6/99 - 11/30/99	4,506,717	20,165	2,069
	12/1/99 - 12/20/99	2,573,251	13,452	1,526
Nagano	2/13/98	11,665,713	59,582	33,875
Sun	9/30/97 - 10/9/97	13,871,352	219,528	116,274
Unav	12/99	6,593,672	28,672	24,150

3.5.2 Experiments

We conducted the experiments of client cluster detection on a wide range of Web server logs ranging in number of requests (148K to 11 Million), number of clients (40K to 60K), number of unique URLS (340 to 116K), duration (24 hours to 94 days), location (various sites in North and South America), organization (governmental, non-profit, commercial), and nature (transient event logs and popular site logs), in order to demonstrate the applicability and generality of our approach. Table 3.4 provides an overview of some Web server logs we used in the experiments.

Table 3.5: Experimental results of client cluster detection on the Nagano server log.

Total number of requests	11,665,713
Total number of unique URLs	33,875
Total number of unique clients	59,582
Number of undetected clients	0
Total number of client clusters	9,853
Largest client cluster	1,343 clients (134,963 requests)
Smallest client cluster	1 client (1 request)
Client cluster URL access	1 URL (3.0e-05% of total) - 8095 URLs (0.24% of total)

Applicability

We first use the Nagano server log to demonstrate the applicability of our method. The Nagano server log is a 1 day (February 13, 1998) extract from the 1998 Winter Olympic Games server log[4]. It has a total of 11,665,713 requests issued by 59,582 clients accessing 33,875 unique URLs. We extract the IP addresses[5] of these clients, run the cluster identification algorithm and group the clients into 9,853 *client clusters*. The size of client clusters varies from 1 to 1,343 clients, the number of requests issued from within each client cluster varies from 1 to 339,632, and clusters access anywhere from 1 to 8,095 unique URLs.

In order to get more insight on the access pattern of client clusters, we plot the cumulative distribution of clients and requests in a client cluster in Figure 3.4.

[4]In our approach, we use the network prefix and netmask information in BGP routing tables as an approximation of the collection of IP addresses belonging to each network—this rarely changes and most of the changes are incremental. The effect of age difference between the BGP routing table dumps (used in identifying client clusters) and the server logs is fixed in the self-correction and adaptation stage of our approach (discussed later in this chapter).

[5]Requests issued from IP address 0.0.0.0, an arbitrary address typically used as source address in protocols such as BOOTP when the client does not know its own IP address were ignored and client 0.0.0.0 was excluded from our experiments.

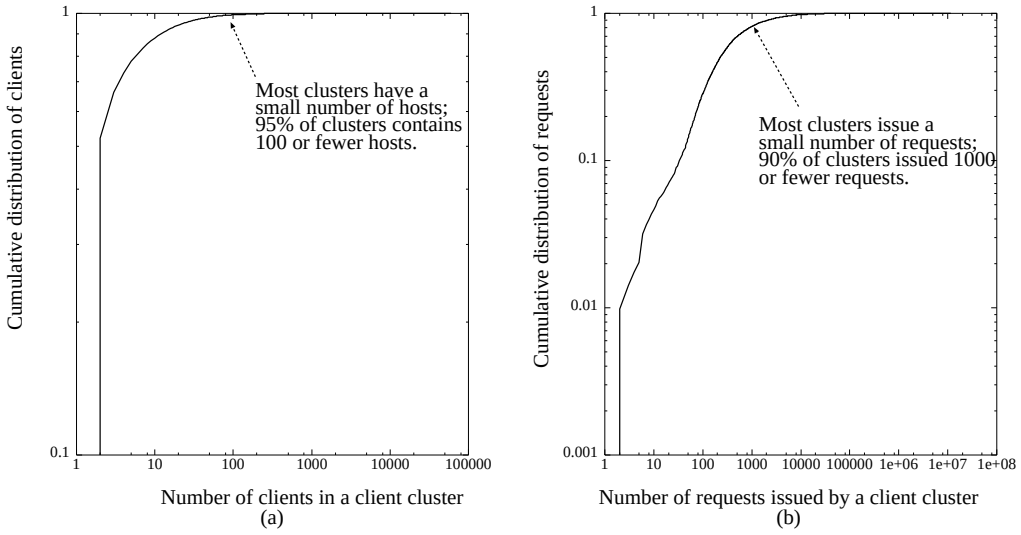


Figure 3.4: The cumulative distribution of clients and requests in a client cluster for the Nagano server log (y axis is in log scale): (a) is the cumulative distribution of the number of clients in client clusters; (b) is the cumulative distribution of the number of requests issued from within client clusters.

Figure 3.4(a) shows the cumulative distribution of the number of clients in a client cluster. We observe that, although the largest client cluster contains 1,343 clients, more than 95% of client clusters contain fewer than 100 clients. The cumulative distribution of requests issued by a client cluster is shown in Figure 3.4(b), which is more heavy-tailed than that of the number of clients in a client cluster as shown in Figure 3.4(a), implying the possible existence of proxies and/or spiders. Around 90% of the client clusters issued fewer than 1,000 requests. Only a few client clusters are very busy, issuing up to a maximum of 339,632 requests. We should note that such Zipf-like distributions are common in a variety of Web measurements [BCF$^+$99].

Figures 3.5(a), (b), and (c) show the distributions of number of clients in client clusters, number of requests issued from within client clusters, and number of URLs accessed from within client clusters, respectively. They are plotted in reverse order of the number of clients in a cluster (i.e., larger client clusters are on the left). From

Figures 3.5(b) and (c), we see that, while larger client clusters usually issue more requests and access more URLs than smaller client clusters, there are a number of relatively small client clusters which issue a significant number of requests ($\sim 1\%$ of the total) and/or access a large fraction of URLs ($\sim 20\%$ of the total) at the server. Locating such unusual clusters is useful in identifying suspected spiders and proxies.

We re-plot the same set of data shown in Figure 3.5 with different sorting of clusters (on x axis)—in reverse order of number of requests. Figures 3.6(a), (b), and (c) show the distributions of the number of requests issued from within client clusters, number of clients in client clusters and number of URLs accessed from within client clusters, respectively. Comparing Figure 3.6(a) with Figure 3.5(a), the results further show that the distribution of the number of requests issued from within client clusters is more heavy-tailed than that of number of clients in client clusters. Figures 3.6(b) and (c) show that busy client clusters usually have a large number of clients and access a large fraction of URLs at the server. We observe that some busy clusters actually have a small number of clients and may access very few URLs—these are also useful measures in identifying spiders and proxies.

Note that Figures 3.5(a), (b), and (c) are plotted in such a way that the points at the same position on the x axis correspond to the same client cluster. For example, cluster 10 (i.e., $x = 10$) in Figures 3.5(a), (b), and (c) refer to the same client cluster. The same is true for Figure 3.6.

Generality

In order to examine the generality of our approach, we conducted our experiments on a very wide range of Web server logs which include Apache, EW3, Sun, and

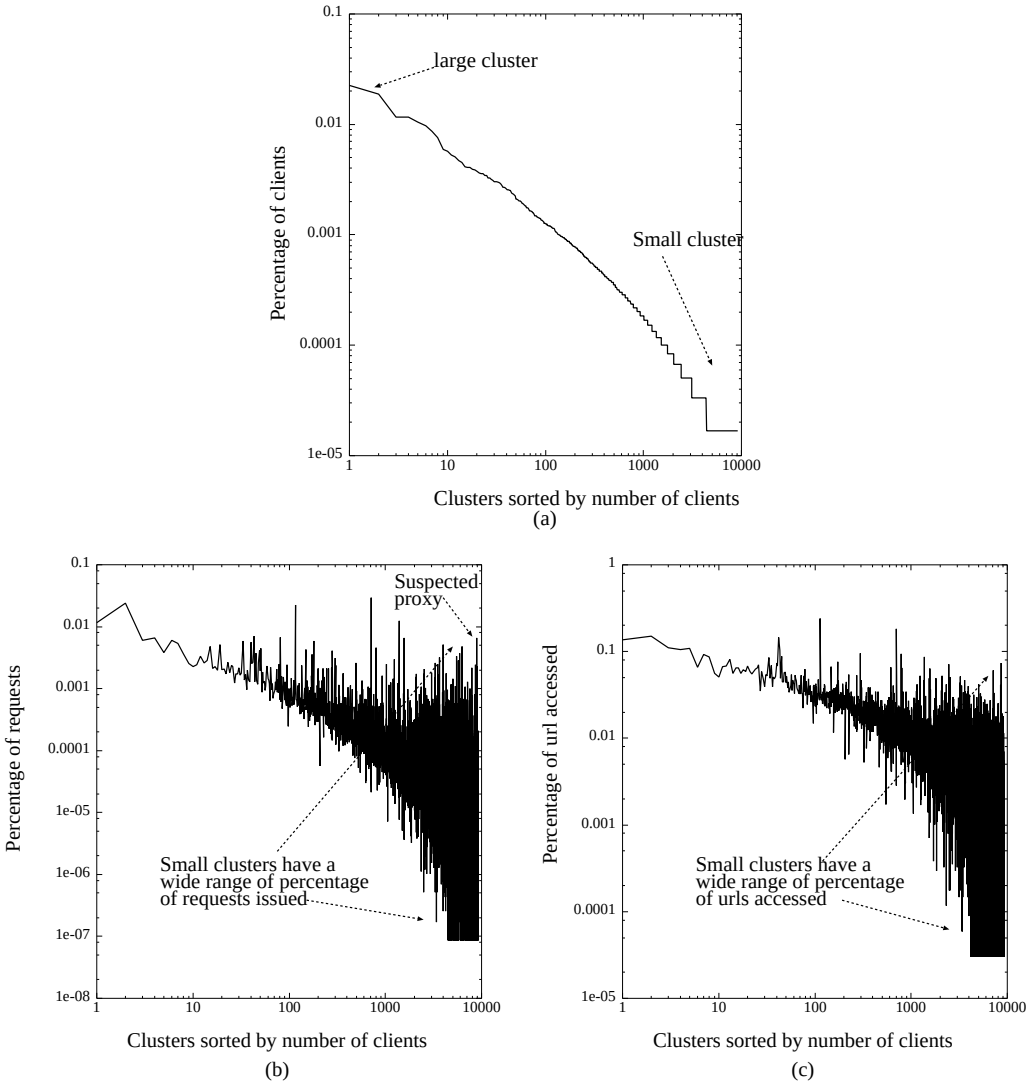


Figure 3.5: The client cluster distribution for the Nagano server log plotted in reverse order of the number of clients in a cluster (both x axis and y axis are in log scale): (a) is the distribution of the number of clients in client clusters; (b) is the distribution of the number of requests issued from within client clusters; and (c) is the distribution of the number of URLs accessed from within client clusters.

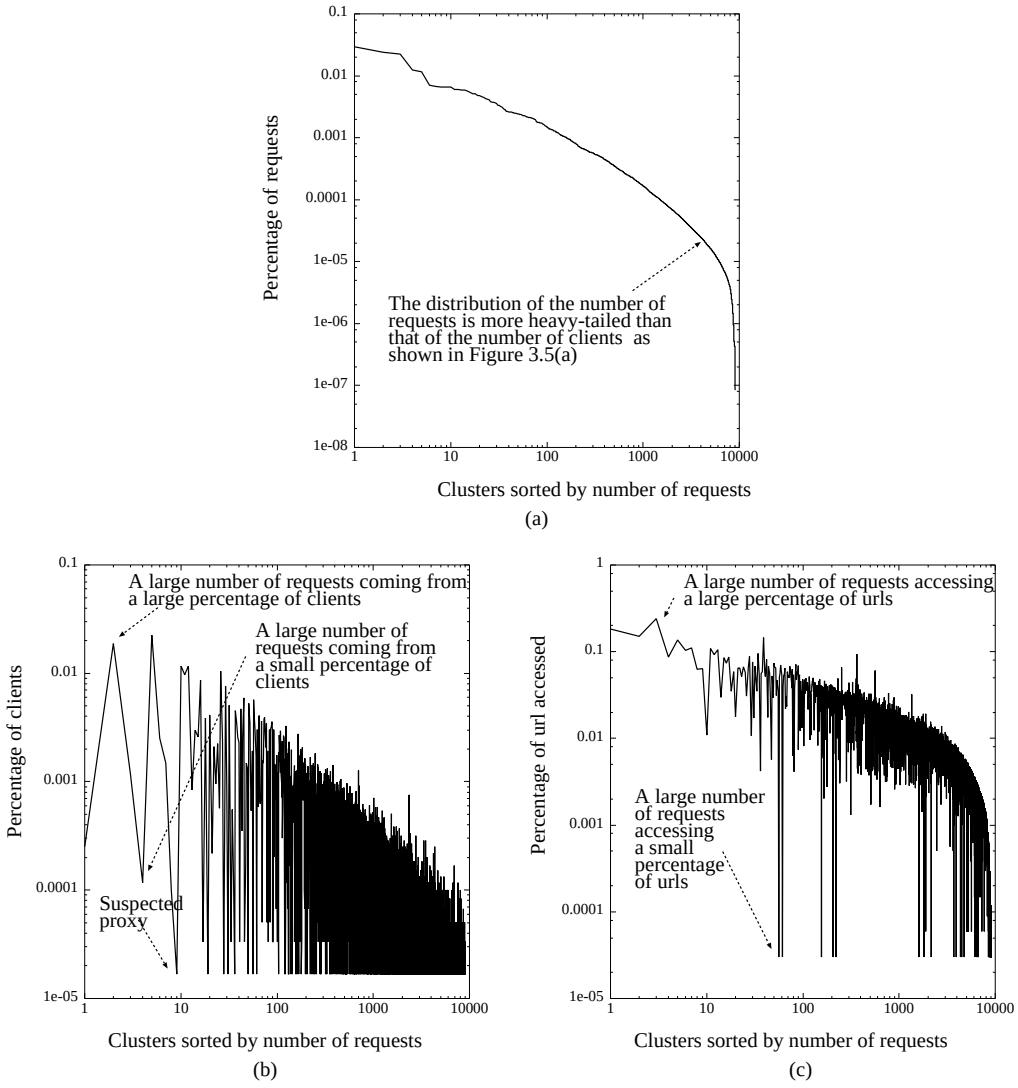


Figure 3.6: The client cluster distribution for the Nagano server log plotted in reverse order of the number of requests (both x axis and y axis are in log scale): (a) is the distribution of the number of requests issued from within client clusters (re-plot of Figure 3.5(b)); (b) is the distribution of the number of clients in client clusters (re-plot of Figure 3.5(a)); and (c) is the distribution of the number of URLs accessed from within client clusters (re-plot of Figure 3.5(c)).

Unav server logs. Instead of including the detailed results client cluster results of these server logs, we provide a glimpse of them in Figure 3.7. The detailed results are provided in Appendix A. Figures 3.7 (a) and (b) are the distributions of the number of clients and number of requests in client clusters in reverse order of the number of clients in the Apache, EW3, Nagano, and Sun server logs, respectively. Figures 3.7 (c) and (d) are the distributions of the number of requests and number of clients in client clusters in reverse order of the number of requests, respectively. All observations on client clustering made on the Nagano server log also apply to every one of the various other server logs we experimented with. For example, we observe suspected proxies or spiders in other logs (Figures 3.7(b) and (d)).

In summary, our experimental results show that we can group more than 99.9% of the clients into clusters, with very few clients not clusterable (i.e., no network prefixes in our prefix table matches the client IP addresses) due to the lack of proper prefix/netmask information in the routing table snapshots. This is fixed in the self-correction and adaptation stage of our approach (discussed in Section 3.8).

3.6 Validation

A client cluster may be mis-identified by being either too large (i.e., containing clients that are not topologically close or are under different administrative control) or too small (i.e., other clusters contain clients that belong to the cluster in question). The simple approach as discussed in Section 1.1 generally errs by producing clusters that are too small. Our approach can produce clusters that are too big, but much less frequently, as we will see below.

Validation of clusters is fundamentally a difficult problem, since the definition

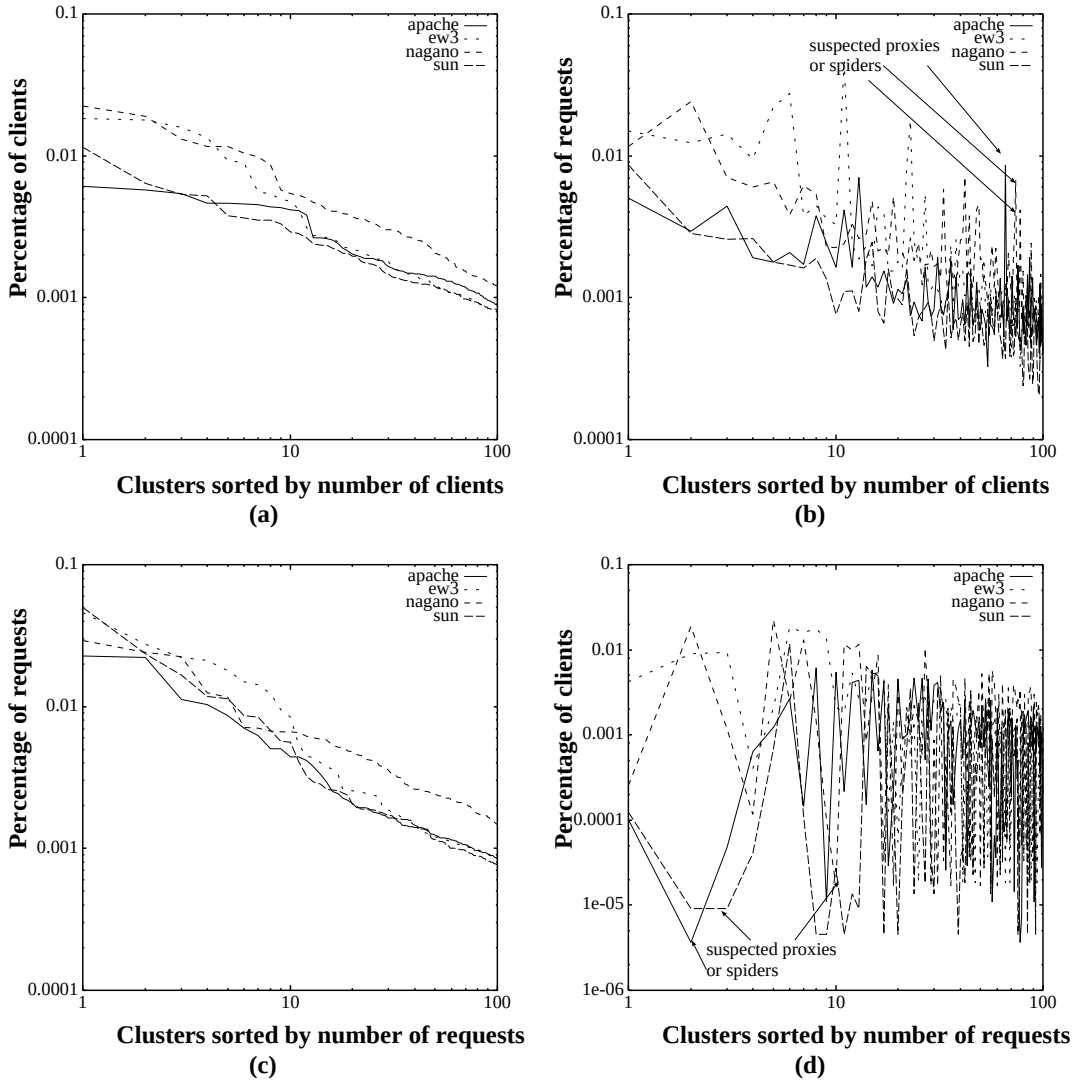


Figure 3.7: Comparison of client cluster distributions of Apache, EW3, Nagano, and Sun server logs (both x axis and y axis are in log scale): (a) and (b) are the distributions of the number of clients and number of requests in client clusters in reverse order of the number of clients, respectively; (c) and (d) are the distributions of the number of requests and number of clients in client clusters in reverse order of the number of requests, respectively.

of a cluster relies on the fuzzy notion of common administrative control. We use two approaches to validation, an *nslookup*-based approach and a *traceroute*-based approach. In each case, we test sampled clusters to determine whether they pass the tests that the approaches apply.

3.6.1 *Nslookup*-based test

Nslookup queries Internet domain name servers for information about hosts:

```
% nslookup 209.68.10.19
Server: www.whitehouse.gov
Address: 198.137.240.91
Name: tyrant.com
Address: 209.68.10.19
```

The *nslookup*-based validation method is based on the observation that clients in the same cluster often share a non-trivial suffix in their fully-qualified domain names. We say that two clients share a non-trivial suffix if the last n components of their names are the same, where a component of a name is a string separated by "." (e.g., *foo.dummy.com* has 3 components: *foo*, *dummy*, and *com*). Suppose there are m components in the client name, then we use $n = 3$ if $m \geq 4$, else we use $n = 2$. For example, *macbeth.cs.wits.ac.za* and *macabre.cs.wits.ac.za* are in the same cluster and their names share the same suffix *cs.wits.ac.za*. We therefore propose a validation method that does an *nslookup* on each client IP address in an identified cluster and performs suffix matching on the names of clients. To pass the *nslookup* test, all the clients in a cluster must have matching suffixes. We label a client cluster to be *incorrect* if there is *even one* client that does not share

Table 3.6: Client cluster validation of the Apache, Nagano, and Sun logs using DNS *nslookup* and optimized *traceroute*.

Server log	Apache	Nagano	Sun
Total number of client clusters	35,563	9,853	33,468
Number of sampled client clusters	382	111	365
Number of sampled clients	2,437	307	2,217
Prefix length range	8 - 29	8 - 28	8 - 29
Number of client clusters of prefix length 24	191	57	186
DNS *nslookup* validation			
Number of *nslookup* reachable clients	1,470	172	1,116
Number of mis-identified client clusters	34	5	22
Number of mis-identified non-US client clusters	21	3	15
Optimized *traceroute* validation			
Number of *traceroute* reachable clients	2,437	307	2,217
Number of mis-identified client clusters	35	12	33
Number of mis-identified non-US client clusters	20	7	28

the same suffix with others. We take samples (1%) from client clusters of various Web server logs. Validation results of Apache, Nagano, and Sun server logs in Table 3.6 (see [KW00, KW00TM] for similar results on other server logs) show that more than 90% of the client clusters pass the validation test using nslookup suffix matching within a cluster. Note, however, that we are only able to obtain names for about 50% of the addresses. For the simple approach, only about 50% of the client clusters pass this validation test. For instance, among the 111 sampled client clusters in the Nagano server log, 5 client clusters fail this validation test by our method. Around 95.4% client clusters pass this validation test. However, only 57 of the total 111 client clusters have prefix length of 24, that is, only 48.6% client clusters pass this validation test using the simple approach.

Since *nslookup* is only done within a cluster, it is possible for clients with similar suffixes to be present in other clusters. That is, we do not check whether the identified clusters are too small. We discuss this issue in Chapter 5.

One reason for mis-identification in our approach is the existence of suspected national gateways/routers on the Internet (e.g., Croatia, France, Japan), which are recognizable by the client name. In such cases, additional information about the clients/networks behind the national gateways/routers are not available in the routing table. Also, route aggregation in the routing table may cause mis-identification in our approach, and may account for approximately 50% of the mis-identifications as observed from Table 3.6. It is hard to further quantify its actual effect on client clustering.

The *nslookup*-based validation has some limitations. As Table 3.6 shows, around 50% of clients are not locatable via *nslookup* in our experiments. This is true for all the logs and varies only very slightly from time to time with repeated experiments over a period of time. There are several possible reasons.

- If an organization operates a firewall on its network, then the DNS server may not give out the name of the machines behind the firewall.
- If the machines on the local network acquire dynamic IP addresses via a DHCP server, the IP addresses do not have a one-to-one mapping to a machine on the network. In such cases, the DNS server will not have the registration record for the dynamic addresses.
- It is also possible that an ISP may not register any names for their customers. In such cases, *traceroute* can be used to get further information.

We use an optimized *traceroute* to resolve the clients which are not locatable by *nslookup* in our experiments. We describe the optimized *traceroute* next.

3.6.2 *Traceroute*-based test

The second approach to validating client cluster identification results is *traceroute*-based. This test first attempts to resolve the address to a name and if successful, tests for suffix match as before. If the client name is not resolvable, it tests to see whether the clients in the same cluster share the same routing path suffix. *Traceroute* attempts to trace the route an IP packet follows to an Internet host by launching UDP probe packets with a small maximum time-to-live (Max_ttl variable), and then listening for an ICMP TIME_EXCEEDED response from gateways along the way. Probes are started with a Max_ttl value of one hop, which is increased one hop at a time until an ICMP PORT_UNREACHABLE message is returned. The ICMP PORT_UNREACHABLE message indicates either that the host has been located or the maximum hop count has been reached. By running *traceroute* on each client, a path towards the designated client is discovered. If two clients sit in the same network, then it is very likely that packets routed to them will traverse paths sharing the same suffix, and vice versa. *Traceroute* imposes more traffic load on the Internet and takes more time to discover routes, but yields more information on clients (such as RTT and hop count). We are more interested in either the name of the client or the last few hops (two in our experiments) on the path towards the client. For faster path resolution and to reduce the traffic load imposed on the Internet, we implemented an optimized *traceroute* (in Linux Red Hat 6.0) with two improvements:

- Instead of having a fixed number (q) of probes sent for each time-to-live (ttl) value independent of ICMP replies returned, we send only one probe for each ttl value. If the first ICMP reply does not provide proper information, we send out the second probe, continuing until we get the proper information or

the number of probes reaches q. Thus we send out $\leq q$ probes for each ttl.

- The initial value of ttl, does not necessarily have to be the default of 1. In fact, we can set the initial value of $ttl = Max\_ttl$ (we set $Max\_ttl = 30$). As such, we only send one probe with a ttl of Max_ttl. If the destination is less than max_ttl hops away, *traceroute* returns the destination IP address, name (if available), and round trip time (RTT). Interestingly enough, around 50% of clients can be resolved in this way, which is consistent with our observation on *nslookup* results. This is because *traceroute* does not get a ICMP reply packet from the designated router which is either unreachable or unwilling to give out the required information due to the presence of a firewall.

Pseudo-code of the optimized *traceroute* is shown in Table 3.7. We define a client to be unresolved if the ICMP reply packet does not contain proper information. In our experiments, we set the constant $c = 30$ and $q = 3$. We send probes in serial[6]. The basic idea is that we do a binary search to find the optimal Max_ttl (i.e., the distance to the designated client) instead of the sequential search in the original *traceroute*. We now give an estimation on how much time and bandwidth we saved on optimized *traceroute*. Assume the average path length is 15. The original *traceroute* will send out $15 \times 3 \approx 45$ probes and the waiting time (measured by number of hops the probes traverse) is $3 \times (1 + 2 + ... + 15) \approx 360$ hops. We observed that 50% clients can be resolved by sending one probe with $ttl = 30$. We also found that almost all clients are fewer than 30 hops away. For the purpose of

[6]Although we can send probes in parallel, which makes the resolving process faster than serial probing does, it will impose much more traffic on the network. Because half of the clients can be resolved by sending one probe, we do not gain a significant saving on resolving time compared to the amount of extra traffic we impose on the network.

performance estimation, we assume that all clients are fewer than 30 hops alway and we only need to do binary search for $ttl < 30$. We further assume that, during binary search, the path length towards clients are randomly distributed with an average of 15 (i.e., average *ttl* is 15) and the average number of probes that need to be sent for each *ttl* is 1.5. Therefore, the average number of probes sent during this binary search is $log_2 30 \approx 5$. The number of probes we need to send for a client is $50\% \times 1 + 50\% \times 5 \times 1.5 \approx 4.25$. The waiting time using the optimized *traceroute* would be $50\% \times 30 + 50\% \times 5 \times 1.5 \times 15 \approx 71.25$ hops. Thus, we save 90% of the probes and 80% of the waiting time.

The advantage of using the optimized *traceroute*[7] on validation is that we are able to resolve the name and, if that fails, the path towards the designated client. The resolvability (either name or path) on clients is improved from 50% to 100% in our applications. In addition, the extra traffic imposed on the network and time spent on resolving clients has been significantly reduced by our optimized *traceroute*. The time consumed by sending one probe in the optimized *traceroute* is about the same as that of a DNS *nslookup*.

We run our optimized *traceroute* on 1% of the sampled client clusters. After resolving all the sampled clients, we apply suffix matching on either the name of the client or on the path towards the client if the client name is not available. (As mentioned earlier, we use either the client's name if it was resolved or the last few hops on the path towards the client for validating a cluster.) We show validation results of Apache, Nagano, and Sun server logs in Table 3.6 (the results on other server logs are in [KW00, KW00TM]). Our results show that 90% of

[7]Note that our optimized *traceroute* is not the same as *ping*. *Ping* does not provide the information we need to validate client clustering results (e.g., the name of clients).

Table 3.7: The pseudo-code of the optimized *traceroute*.

```
Max_ttl = c;
for each client IP address {
    ttl = Max_ttl;
    send one probe packet;
    save ICMP reply;
    while (unresolved) {
        if (Max_ttl is too short) {
            Max_ttl *= 2;
            ttl = Max_ttl;
            n = 0;
            while (no reply and n < q) {
                send one probe packet;
                n += 1;
            }
            save ICMP reply;
        }
        if (Max_ttl is too long) {
            Max_ttl /= 2;
            ttl = Max_ttl;
            n = 0;
            while (no reply and n < q) {
                send one probe packet;
                n += 1;
            }
            save ICMP reply;
        }
    }
    hop_count = ttl;
    if (path needed) {
        for each missing time-to-live value t in [1..hop_count] {
            Max_ttl = t;
            ttl = t;
            send one probe packet;
            save ICMP reply;
        }
    }
}
```

the client clusters pass the traceroute-based test. Moreover, we use the validation information to improve the applicability and accuracy of the cluster identification results. We will discuss this in detail in Section 3.8.

3.6.3 Discussion

Many real applications will be tolerant to a certain degree of inaccuracy. An alternative way to validation is to set a threshold (say 5%) and selectively sample clients. For example, if 95% of the clients inside a cluster are correctly identified, we could consider this cluster to be correct. This selective sampling can be performed in either a client-based or a request-based manner depending on the application's criteria.

3.6.4 Comparison of the simple approach and the network-aware approach

Given that the simple approach and our approach pass the validation tests in 50% and 90% of the samples cases, respectively, we compare the client cluster distributions of the Nagano server log obtained by the two approaches to illustrate the effect of cluster mis-identification. The number of client clusters identified by our approach is 9,853 as compared to 23,523 of the simple approach. The largest client cluster identified by our approach contains 1,343 hosts (issuing 134,963 requests or 1.15%). However, there are only 63 hosts in the largest client cluster identified by the simple approach (issuing 9,662 requests or 0.08%). We further plot client cluster distributions of the Nagano server log obtained by our approach (as dotted curves) and by the simple approach (as solid curves) in Figure 3.8. Figures 3.8(a) and (b) show the distributions of the number of clients in client clusters in reverse

order of the number of clients and in reverse order of the number of requests, respectively. We observe that the number of client clusters identified by the simple approach is much larger than that of our approach. Note that the maximum number of clients in a client cluster is 256 because of its assumption of prefix length of 24. The average size of client clusters identified by the simple approach is smaller than that of our approach, as is the variance of the size of client clusters. Figures 3.8(c) and (d) show the distribution of number of requests issued from within client clusters in reverse order of the number of requests and in reverse order of the number of clients. The average number of requests issued from within client clusters identified by the simple approach is smaller than that of our approach. The significant differences in the client cluster distributions obtained by our approach and the simple approach indicates that the simulation results based on different client cluster identification methods may vary a lot.

3.7 Effect of BGP dynamics on client cluster identification

An update of a BGP routing table is triggered by changes in network reachability and topology, as well as by policy changes. BGP dynamics is a well-known phenomenon which affects the performance of Internet applications [LMJ97]. In this section, we examine the impact of BGP dynamics on client cluster identification results. The experiments were conducted on a timescale of days. We download routing table snapshots daily. In our measurements, we define the *dynamic prefix set* to be the set of prefixes that changed during the entire testing period, that is, the set of prefixes which are not in the intersection of prefixes of all the routing

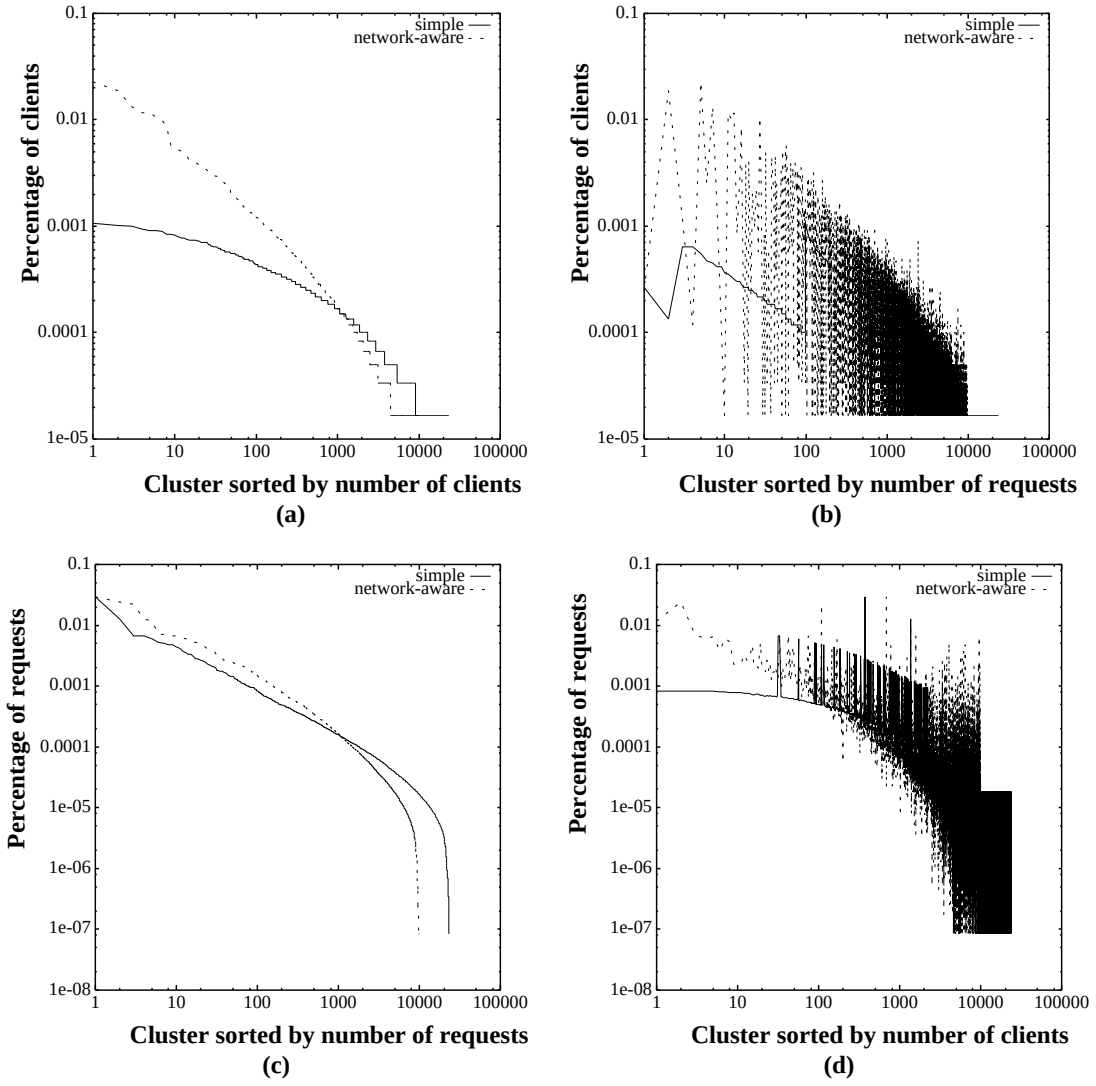


Figure 3.8: Comparison of the client cluster distributions of the Nagano server log obtained by our approach and the simple approach (both x axis and y axis are in log scale): (a) and (b) are the distributions of the number of clients in client clusters in reverse order of the number of clients and number of requests, respectively; (c) and (d) are the distributions of the number of requests issued from within client clusters in reverse order of the number of clients and number of requests, respectively.

tables we obtained over the entire testing period. We further define the *maximum effect* to be the size of the dynamic prefix set. The changes from day to day will typically be much smaller than the maximum effect.

We use AADS, PACBELL, SINGAREN, and VBNS routing tables as examples to illustrate the effect of BGP dynamics (Tables 3.8– 3.11). We show measurement results for 1, 4, 7, and 14 day periods on Apache, EW3, Nagano, and Sun server logs. For example, the AADS routing table entries vary between 16,595 and 17,288 on those days, among which the number of prefixes (*maximum effect*) that are not in the intersection of the routing tables on those days vary between 711 and 1,404. The number of prefixes used to identify client clusters in the Apache server logs on those days vary from 3,932 to 4,035, and the corresponding maximum effects vary from 124 to 227. There are 2,869 client clusters issuing a large number of requests, among which between 605 and 614 client clusters are identified using the network prefixes in AADS routing table. The maximum effects for these busy clusters vary from 22 to 31. We observe that overall BGP dynamics affects fewer than 3% of client clusters. This result holds across all the routing tables and all the Web server logs. Therefore, we believe that, although the BGP routing table changes dynamically according to the network environment, its impact on client cluster identification is minor, and fixed in the self-correction and adaptation process (discussed next).

3.8 Self-correction and adaptation

Besides validating the client cluster results, we use *traceroute* for two other purposes. In Section 3.5.2, we showed that more than 99.9% clients in the Web server

Table 3.8: The effect of AADS dynamics on client cluster identification.

Period (days)	0	1	4	7	14
AADS prefix	16,595	16,669	16,704	16,792	17,288
Maximum effect	711	785	820	908	1,404
Apache prefix (total 35,563)	3,932	3,935	3,929	3,950	4,035
Maximum effect	124	127	121	142	227
Apache busy clusters (total 2,869)	605	603	603	605	614
Maximum effect	22	20	20	22	31
EW3 prefix (total 24,921)	2,592	2,588	2,582	2,604	2,682
Maximum effect	75	71	65	87	165
EW3 busy clusters (total 2,062)	385	386	388	390	412
Maximum effect	4	5	7	9	31
Nagano prefix (total 9,853)	663	665	663	673	726
Maximum effect	22	24	22	32	85
Nagano busy clusters (total 717)	93	94	93	93	105
Maximum effect	2	3	2	2	14
Sun prefix (total 33,468)	3,646	3,651	3,640	3,669	3,756
Maximum effect	124	129	118	147	234
Sun busy clusters (total 2,536)	527	525	529	530	542
Maximum effect	15	13	17	18	30

logs can be clustered using our method. We have also shown in Section 3.6 that both *nslookup* and *traceroute* validation results show that our method passes validation tests in 90% of the cases. Periodic *traceroute* results on sampled clients can be used to further improve the applicability (i.e., identify the unidentified clients) and accuracy (i.e., correct the mis-identifications) of cluster identification. Periodic sampling can also be used to make client cluster identifying adaptive to network dynamics (i.e., changes of IP address allocations and network topology). For the purpose of identifying un-identified clients ($\sim 0.1\%$), we first consider each individual un-identified client to be a single client cluster. Then we merge them into bigger client clusters gradually according to *traceroute* sampling information. Self-correction and adaptation is also important to generate client clusters using real-time routing information and producing real-time client cluster identification

Table 3.9: The effect of PACBELL dynamics on client cluster detection.

Period (days)	0	1	4	7	14
PACBELL prefix	25,532	25,642	25,359	25,198	25,566
Maximum effect	1,746	1,856	1,573	1,412	1,780
Apache prefix (total 35,563)	6,435	6,485	6,416	6,389	6,557
Maximum effect	204	254	285	258	226
Apache big prefix (total 2,869)	1,091	1,099	1,092	1,092	1,119
Maximum effect	34	32	35	35	32
EW3 prefix (total 24,921)	3,895	3,907	3,874	3,857	3,985
Maximum effect	140	122	119	132	128
EW3 big prefix (total 2,062)	516	513	516	541	589
Maximum effect	14	15	16	17	14
Nagano prefix (total 9,853)	954	953	951	964	1,047
Maximum effect	37	36	34	37	35
Nagano big prefix (total 717)	132	132	129	134	144
Maximum effect	4	3	2	5	5
Sun prefix (total 33,468)	5,960	6,007	5,950	5,928	6,017
Maximum effect	227	204	217	225	204
Sun big prefix (total 2,536)	851	857	854	853	884
Maximum effect	21	27	24	23	24

Table 3.10: The effect of SINGAREN dynamics on client cluster detection.

Period (days)	0	1	4	7	14
SINGAREN prefix	67,686	67,997	67,950	68,226	68,293
Maximum effect	1,958	2,269	2,222	2,498	2,565
Apache prefix (total 35,563)	15,330	15,320	15,275	15,296	15,262
Maximum effect	217	207	162	183	149
Apache big prefix (total 2,869)	2,268	2,266	2,260	2,261	2,260
Maximum effect	21	19	13	14	13
EW3 prefix (total 24,921)	9,931	9,912	9,906	9,906	9,888
Maximum effect	109	90	84	84	66
EW3 big prefix (total 2,062)	1,309	1,309	1,309	1,309	1,308
Maximum effect	3	3	3	3	2
Nagano prefix (total 9,853)	3,730	3,724	3,718	3,718	3,712
Maximum effect	41	35	29	29	23
Nagano big prefix (total 717)	471	471	469	469	469
Maximum effect	2	2	0	0	0
Sun prefix (total 33,468)	14,412	14,397	14,341	14,374	14,333
Maximum effect	213	198	142	175	134
Sun big prefix (total 2,536)	1,956	1,956	1,951	1,950	1,949
Maximum effect	12	12	7	6	5

Table 3.11: The effect of VBNS dynamics on client cluster detecting.

Period (days)	0	1	4	7	14
VBNS prefix	1,855	1,855	1,855	1,855	1,855
Maximum effect	1	1	1	1	1
Apache prefix (total 35,563)	675	675	675	675	675
Maximum effect	0	0	0	0	0
Apache big prefix (total 2,869)	288	288	288	288	288
Maximum effect	0	0	0	0	0
EW3 prefix (total 24,921)	461	461	461	461	461
Maximum effect	0	0	0	0	0
EW3 big prefix (total 2,062)	159	159	159	159	159
Maximum effect	0	0	0	0	0
Nagano prefix (total 9,853)	164	164	164	164	164
Maximum effect	0	0	0	0	0
Nagano big prefix (total 717)	45	45	45	45	45
Maximum effect	0	0	0	0	0
Sun prefix (total 33,468)	698	698	698	698	698
Maximum effect	0	0	0	0	0
Sun big prefix (total 2,536)	285	285	285	285	285
Maximum effect	0	0	0	0	0

results. By real-time cluster identifying we mean application of cluster identification techniques to very recent server log data (within the last few minutes).

We consider two cases in the self-correction and adaptation process:

1. If there is more than one cluster which belongs to the same network, we merge them into one big cluster and the network prefix and netmask will be recomputed accordingly;

2. If there is a cluster which contains clients belonging to more than one network, we partition the cluster into several clusters based on the *traceroute* sampling results.

3.9 Summary

In this chapter, we presented a novel way to identify IP address clusters using BGP routing information. The experimental results show that our method is able to group more than 99.9% of the IP addresses captured in a wide variety of Web logs into clusters. In order to provide a quantitative measurement of the accuracy of the method, we propose two validation tests based on network tools *nslookup* and *traceroute*. Sampling validation results demonstrate that our method passes validation tests in over 90% of the cases. A self-correction and adaptation mechanism was also proposed to improve the applicability and accuracy of the initial cluster identification results. The entire cluster identification process can be done in an automated fashion—moving from a Web log to a set of interesting client clusters. In addition, our methodology is immune to BGP dynamics and independent of the range and diversity of server logs.

Chapter 4

Applications of client clustering

A network-aware clustering of IP address provides a good level of IP address aggregation which helps researchers to obtain a solid understanding of the Internet topology and traffic load pattern. The network-aware client clustering mechanism is applicable to Web caching, server replication, content distribution, server-based access prediction, and network management. The real-time client clustering information (i.e., the client clusters results identified from recent data logs using real-time routing information) gives the service provider a global view of where their customers are located and how their demands change from time to time. It enables content distribution service providers to deliver the right content to the right customer at the right time. It is crucial information for service providers to enhance their services, reduce costs, and extend global presence. As a service replication mechanism, better service can be provided by placing replicated servers (e.g., mirror sites) at hot spots to accommodate customers' demands. With the knowledge of customer's location and the traffic load origination, the service providers can do better load balancing by provisioning in advance. Client clustering provides information on client locations, which is also useful for Internet

topology discovery. In this thesis, we discuss two applications of clustering—Web caching and server replication.

4.1 Web caching

We examine the usefulness of network aware clustering in a Web caching simulation and contrast it with the simple approach discussed in Chapter 1. We present our approach to identifying spiders and proxies and then describe proper placement of proxies between clients and servers. While it is clearly more likely for site administrators to know the need for placing proxies, the general question of moving content closer to users and the positioning of proxies is an interesting one to address. Our techniques are equally likely to apply to positioning content distribution servers closer to the busy client clusters.

4.1.1 Client classification

We classify clients into *visible clients* (visible to the Web server, representing a majority), *hidden clients* (hidden behind proxies and thus not visible to the server), and *spiders*. All visible clients belonging to the same client cluster will share one set of proxies in the Web caching system. The goal of our Web caching system is to improve the Web access quality perceived by *both* visible and hidden clients. First, we identify spiders and eliminate them from server logs. Spiders blindly visit many resources on a Web site. Clients in the same cluster with a spider will not benefit from a proxy in front of the cluster (as shown in Figure 4.1(a)) since the spider will issue most of the requests and a majority of the URLs accessed by it will not be accessed repeatedly. Therefore, it is not desirable to install a new

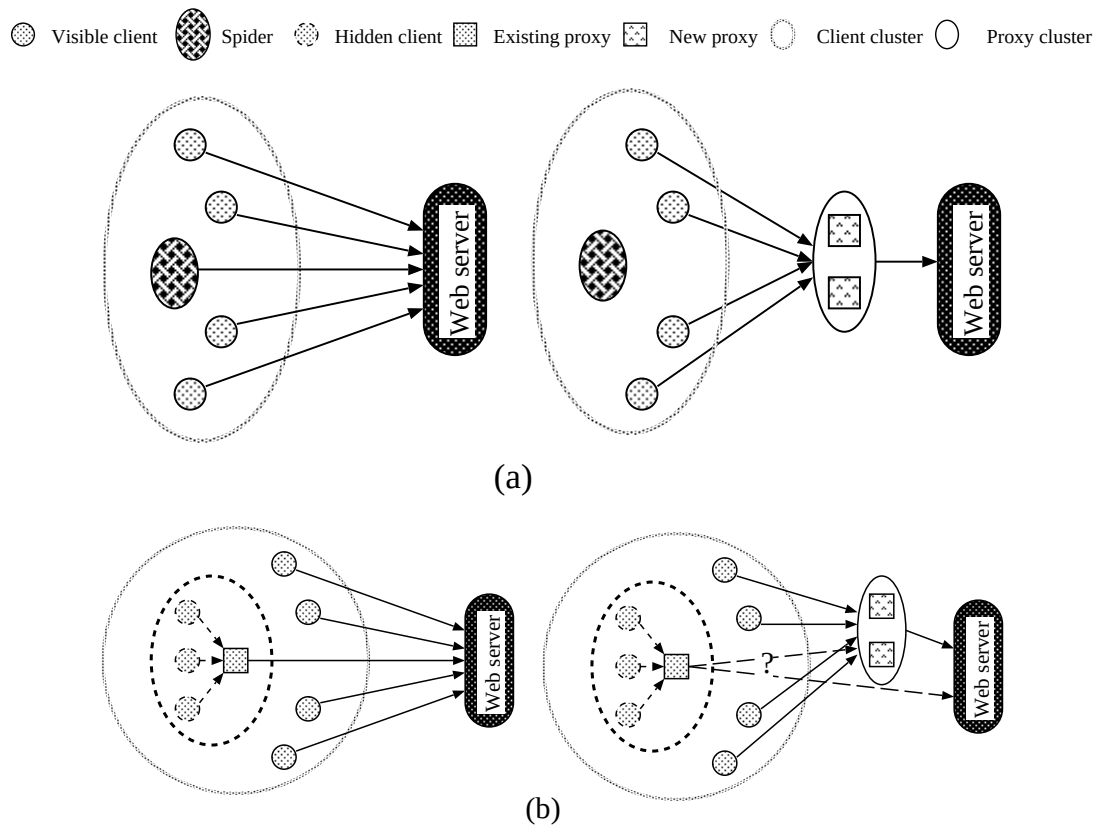


Figure 4.1: Eliminating spiders and existing proxies from the server logs: (a) spiders; (b) existing proxies.

proxies in front of a spider (as shown at the right-hand side of Figure 4.1(a)). Next, we locate existing proxies since (Figure 4.1(b)) hidden clients behind them may suffer longer latencies due to being an additional hop away from the origin server. These hidden clients will only benefit from a new proxy if most of their requested pages, which are not available at the existing proxy, are cached at the new proxy (i.e., those pages are also requested by normal clients in the same cluster as the existing proxy). We still need to examine whether or not installing new set of proxies in front of existing proxies is desirable (as shown at the right-hand side of (Figure 4.1(b)).

4.1.2 Identifying spiders/proxies

A cluster containing a spider often issues a very large number of requests within a short period. The host(s) within that client cluster responsible for a large percentage of these requests, if any, is (are) suspected to be the spider(s). The overall access pattern of clients (consisting of the number of unique URLs accessed, the arrival time of the requests, and the request distribution inside a client cluster) is shared by a proxy, but not by a spider.

The number of unique URLs accessed can identify some spiders, but may not be enough to identify all of them. The spider in the Sun server log, which is in a cluster of 27 hosts, issued 692,453 requests and accessed 4,426 out of 116,274 unique URLs. We can distinguish spiders from clients or proxies by examining the request arrival time. From Figure 4.2(b), we do not see any similarity between the access pattern of the spider and that of the entire server log shown in Figure 4.2(a). There are certain correspondences between the access pattern of the proxy shown in Figure 4.2(c) and that of the entire server log. Each spike observed in the proxy access pattern matches the daily spike in the access pattern of the entire log.

If a client cluster contains both spiders and normal clients, the requests issued by hosts within the cluster will have an uneven distribution among them and can help identify a spider. The request distribution of the cluster containing a spider is shown in Figure 4.3. Almost all the requests are issued by the spider. We thus need to combine examination of request arrival time and the distribution of requests within a cluster to identify spiders. There are no spiders in the Nagano server log—unsurprising given that it is a single day's log of a transient event site.

Unlike differentiating a spider from a proxy and a client, it is harder to identify proxies among a group of clients. A proxy will mimic the access pattern of clients

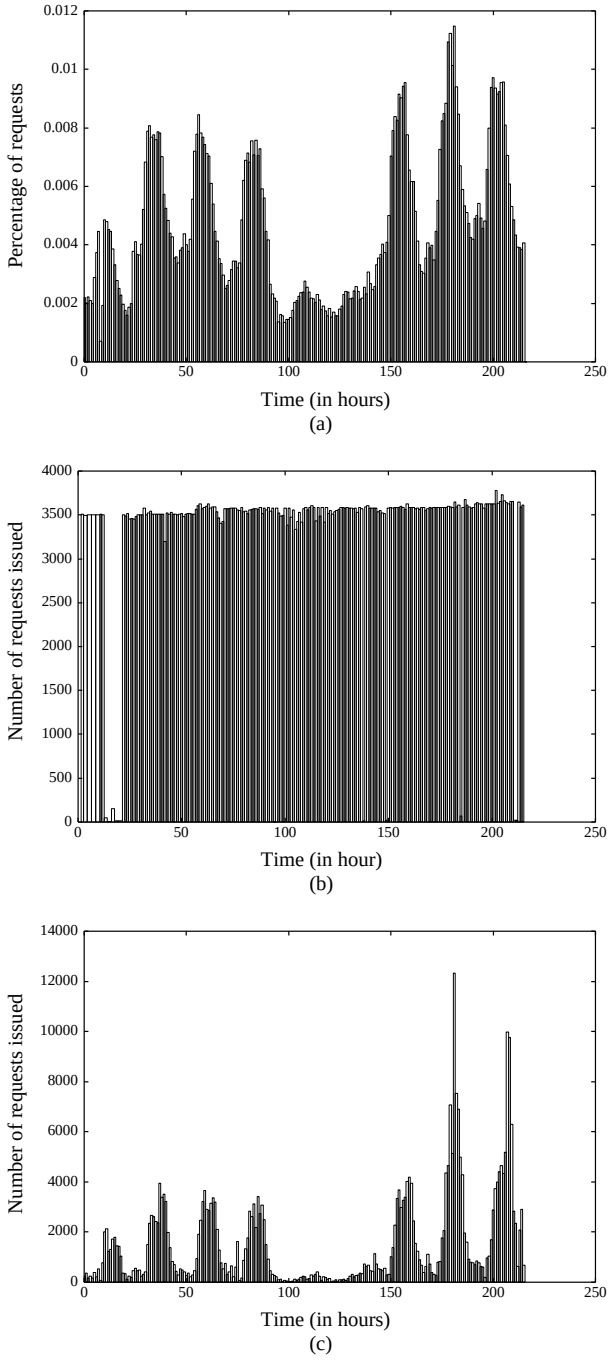


Figure 4.2: Histogram of the requests in the Sun server log: (a) the entire server log, (b) a client cluster containing a spider, (c) a client cluster containing a proxy.

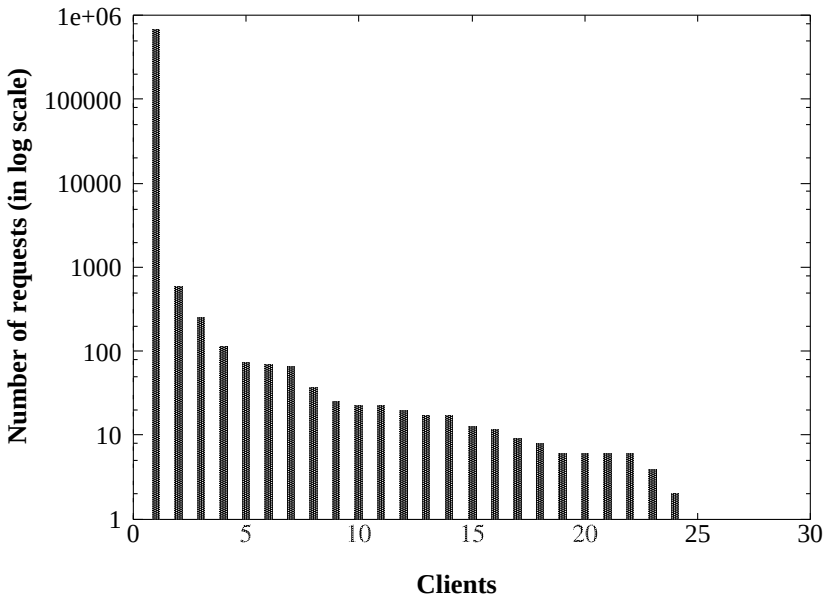


Figure 4.3: The client request distribution of a client cluster containing a spider in the Sun server log which issues 692,453 requests (99.79% of all requests in the cluster).

behind it (see Figures 4.2 and 4.4). One difference between a proxy and a client is that the proxy may issue more requests and have a shorter "think" time between requests than a client does. For example, we identified a client cluster in the Sun server log issuing 326,566 requests. It had only two clients in it issuing 2,699 and 323,867 requests, respectively. We suspect that the second client is a proxy. In the Nagano server log a client cluster containing only one client issued 77,311 requests. Proxies can also be seen in Figures 3.5(b) and 3.6(b) in Chapter 3. Our method depends on the number of clients sharing the proxy and their access pattern. We have not found a solution guaranteed to locate all proxies correctly.

Besides using access patterns, the `User-Agent` field of the HTTP request[HTTa, HTTb], if present in a server log, is also useful in differentiating proxies. The `User-Agent` field includes information about the particular browser, Operating System version and hardware used by the client initiating the request. If there

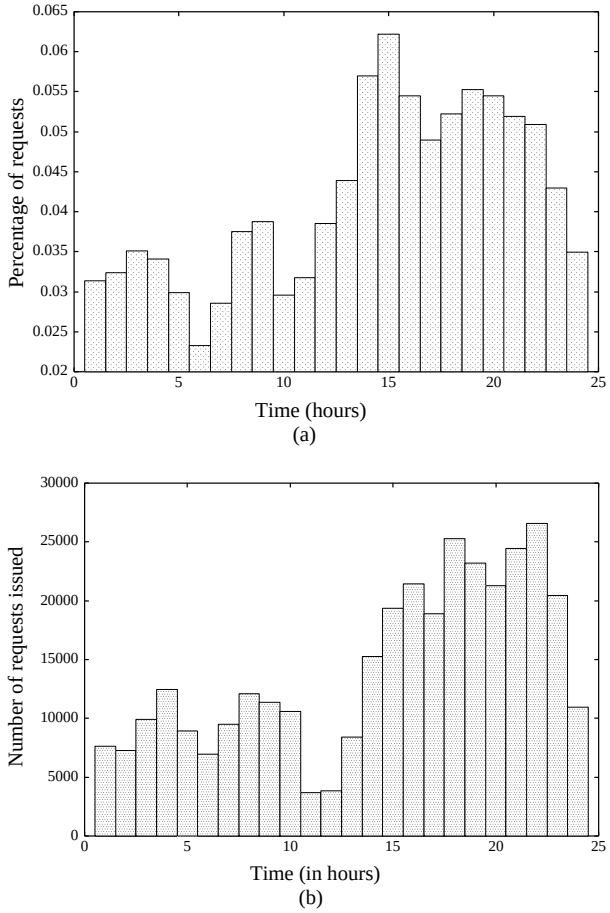


Figure 4.4: Histogram of requests in the Nagano server logs: (a) the entire server log, (b) a client cluster containing only a proxy.

are several different `User-Agent` fields from the same host responsible for a lot of requests, then it is likely that the host is a proxy.

4.1.3 Thresholding client clusters

After identifying and eliminating possible spiders and proxies from the server log, we filter out uninteresting client clusters by thresholding on the number of requests issued from within a client cluster. After reverse sorting based on the number of requests issued from within client clusters, we retain *busy* client clusters—clusters whose total requests added up to at least 70% (see Chapter 5) of the total requests

in the server log. Such a thresholding reduced the number of client clusters dramatically with the smallest client cluster included issuing at least 2,744 requests. In the Nagano log there were only 717 busy client clusters out of 9,853 (see Table 4.1). These busy client clusters have 32,691 clients and issue 8,167,590 requests. The size of busy client clusters ranges from 1 to 1,343 clients and the number of requests issued by busy client clusters ranges from 2,744 to 339,632. The size of the remaining (*less-busy*) client clusters, filtered out, ranges from 1 to 46 clients and the number of requests issued by these less-busy client clusters ranges from 1 to 2,741. The results of thresholding client clusters of other server logs are similar to the Nagano log and are provided in [KW00, KW00TM].

Similar thresholding on clusters identified using the simple approach (discussed in Chapter 1) leads to 3,242 busy client clusters (out of 23,523) in the Nagano log, with the smallest cluster issuing 696 requests. The busy clusters have 30,774 clients, issuing 8,167,335 requests (70% of the total). The size of busy client clusters ranges from 4 to 63 clients and the number of requests issued by them ranges from 696 to 339,632. The size of less-busy client clusters ranges from 1 to 4 clients and the number of requests issued by these less-busy client clusters ranges from 1 to 695. These results are very different from that of the network-aware approach, from which we conclude that the simple approach does not do a good job on clustering clients.

4.1.4 Proxy placement

One way to place proxies is to assign one or more proxies for each client cluster based on metrics such as the number of clients, the number of requests issued, the URLs accessed, or the number of bytes fetched from a server. The proxies

Table 4.1: Experimental results of thresholding client clusters on the Nagano server log.

Approach		Network-aware	Simple
Total number of client clusters		9,853	23,523
Threshold (requests per client cluster)		2,744	696
Number of busy client clusters		717	3,242
	(clients)	32,691	30,774
	(requests)	8,167,590	8,167,335
Busy client clusters	(requests)	2,744 - 339,632	696 - 339,632
	(clients)	1 - 1,343	4 - 63
Less-busy client clusters	(requests)	1 - 2,741	1 - 695
	(clients)	1 - 46	1 - 4

assigned to clients in the same client cluster form a proxy cluster and would cooperate with each other. Alternatively, we can place a proxy in front of each client cluster and further group proxies into proxy clusters according to their AS numbers and geographical locations. All proxies belonging to the same AS and located geographically nearby will be grouped together to form a proxy cluster. The first approach is easier to implement while the second one, though more practical, is complicated. We only address the first approach in this thesis.

4.1.5 Experimental results

We use the client cluster results in the trace-driven simulation to evaluate the benefit of proxy caching. The big picture of the Web caching simulation is shown in Figure 4.5. We place proxies in front of each client cluster. We implement the Piggyback Cache Validation (PCV) [KW97] scheme (also described in Section 2.2.6) with a fixed *ttl* (time-to-live) expiration period at each proxy cache. By default, a cached resource is considered stale once an hour has elapsed. When the

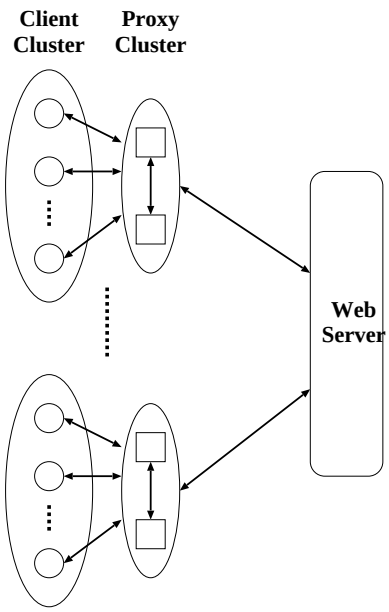


Figure 4.5: Clustering clients and proxies.

expiration time is reached for this resource, a validation check is piggybacked on a subsequent request to its server. If the resource is accessed after its expiration, but before validation, then a `GET If-Modified-Since` request is sent to the server for this resource. We use LRU as the cache replacement policy. Our purpose is not to evaluate the PCV scheme, but to illustrate the applicability of client clustering on Web caching simulation and to determine the effect of different client cluster identification approaches on simulation results.

In our simulation, we set *ttl* to be 1 hour, vary cache size at each proxy and examine two performance metrics: hit ratio and byte hit ratio. The hit ratio is defined as the fraction of the number of requests that are served by proxies. The byte hit ratio is defined as the fraction of the number of bytes that are requested by clients and are served by proxies. Varying *ttl* to 5, 10, and 15 minutes yields similar results. We compare the results of simulations on the client clusters obtained by our approach and the simple approach. Our simulations are conducted on the

Nagano logs and the EW3 logs though we only show the results on the Nagano server log[1].

Our evaluation of simulation results consists of two parts: server performance and proxy performance. To evaluate server performance, we vary the cache size at each proxy from 100 KB to 100 MB and examine the total request hit ratio and the byte hit ratio observed at the server. Figure 4.6 shows the simulation results of server performance on the Nagano server log. The curve with a '*' marker shows the results of our approach and the curve with a '+' marker shows the results of the simple approach. Figures 4.6(a) and (b) show the total request hit ratio and the byte hit ratio observed at the server (i.e., the ratio and byte ratio of requests served by local proxies), respectively. Both hit ratio and byte hit ratio increase as the proxy cache size increases. The results show that the simple approach underestimate (around 10%) both the request hit ratio and the byte hit ratio observed at the server when the local proxy cache size is large (e.g., $>$ 700KB). The hit ratio (i.e., up to $60 \sim 75\%$) obtained from our simulation is greater than typical cache hit ratios of around 40% [ABCdO96] due to the proxies in our simulation being dedicated to one typical server. The client access and resource updating pattern of the Nagano event log are different from other logs. We did not observe a high hit ratio on any of the logs.

To evaluate proxy performance, we fix the cache size at infinity and examine the request hit ratio and byte hit ratio at each proxy (Figure 4.7). The solid curves show the results of our approach and the dotted curves show the results of the simple approach. We only show the results of the top 100 client clusters in the reverse order of the number of requests. Figures 4.7(a) and (b) show the number of

[1]Requests to resources accessed by clients fewer than 10 times are ignored.

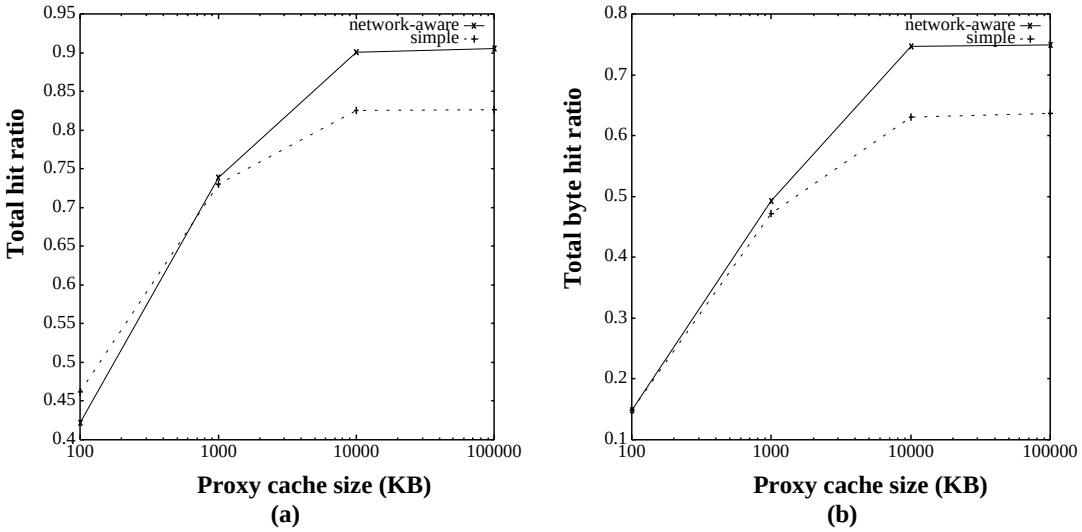


Figure 4.6: Simulation results on Web server performance vs proxy cache size of the Nagano server log (x axis is in log scale): (a) is the total hit-ratio, (b) is the total byte hit ratio.

requests and size (in KB) of requests issued in client clusters in reverse order of the number of requests, respectively. Figures 4.7(c) and (d) show the request hit ratio and the byte hit ratio observed at each proxy. The great differences between the client requests and proxy hit ratio results obtained from our approach and those obtained from the simple approach demonstrate that the simple approach fails to properly evaluate potential benefit of proxy caching.

To summarize, the significant differences shown in simulation results demonstrate that the simple approach is not able to evaluate benefits of Web caching schemes or the corresponding overhead (both at server and at proxies) well, and hence, fails to serve as a guide for solving problems such as proxy placement. The simulation results from our network-aware approach are more realistic, and is useful for designing and evaluating Web caching systems. For example, knowing the location of clients and their demands, a Web site can better provision its service. While we only address simulation of the Web caching system with one server and

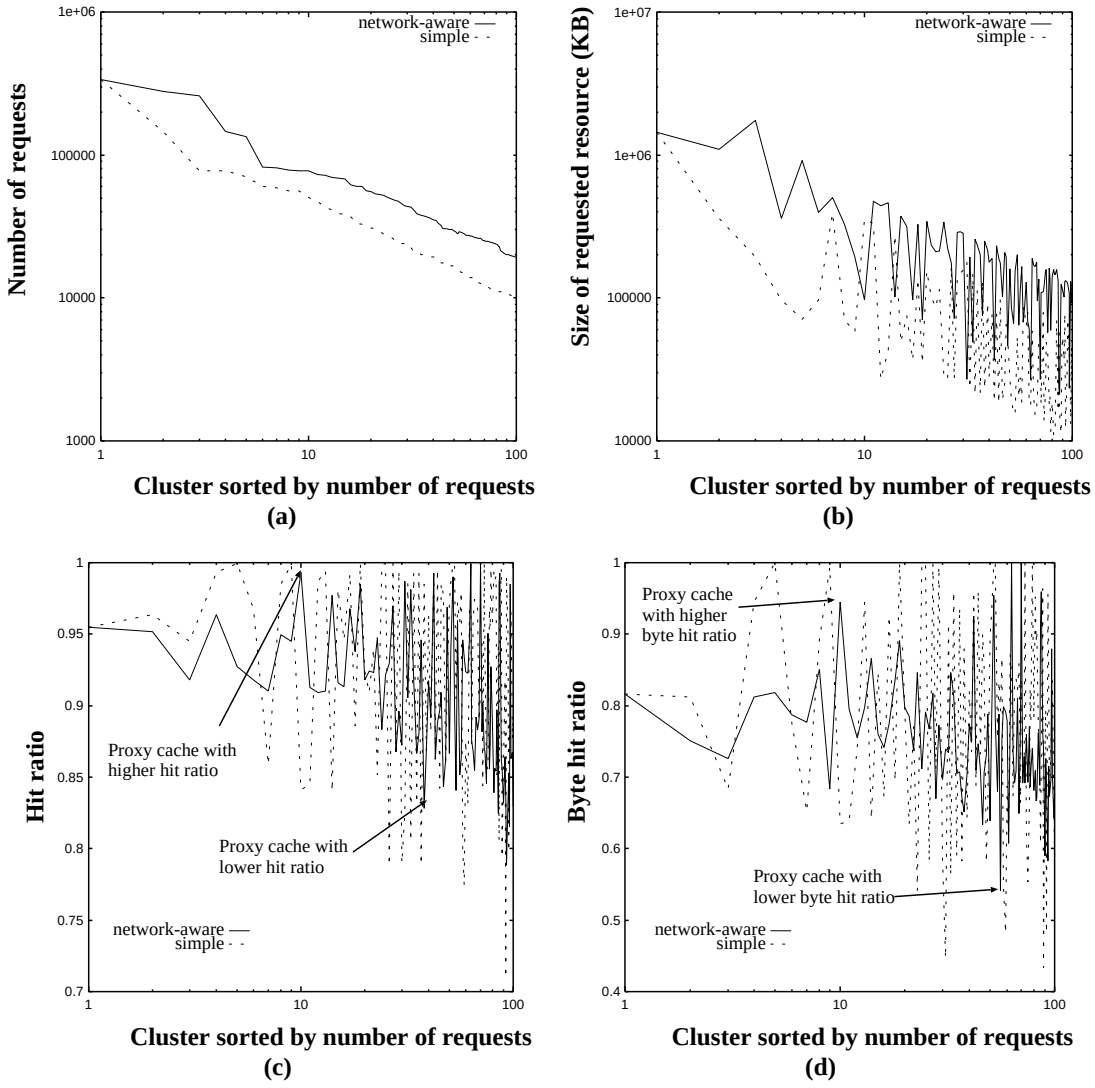


Figure 4.7: Simulation results of proxy cache performance of the top 100 client clusters in the Nagano server log (x axis is in log scale): (a) and (b) are the number of requests and size of requested resources in client clusters in reverse order of the number of requests, respectively; (c) and (d) are the proxy cache request hit ratio and byte hit ratio of client clusters in reverse order of the number of requests, respectively.

multiple proxies, we can also simulate multiple servers and multiple proxies by merging more server logs collected at the same time.

4.2 Server replication

On-line network-aware clustering is also useful in server replication. Here, we discuss a typical application of network-aware clustering in server replication—automatic user request re-direction.

To accommodate users' demands, a Web site usually replicates its content at several places (mirror servers) to lower load on the main server and to reduce user access latency. Both the main server and the mirror servers provide the same content service. In most cases, the Web site leaves the option to the users to decide which mirror server to contact. However, short of knowledge on the performance and network connectivity of the Web servers[2], the server chosen by the user could be far away from the user or have poor connectivity to the user. In such cases, it is helpful if any of the servers of a Web site, upon receiving a client request, can detect the client "location", select a good server (e.g. lower user-perceived latency), and automatically forward the user request to the selected server.

To achieve the automatic request forwarding, the server needs to perform three tasks:

1. good server set selection;

[2]One could use network tools, such as *traceroute* and *pathchar* [pat], to characterize the bandwidth and latency along the end-to-end path from the client to each server and select a good server to fetch the document. Unfortunately, the network latency (RTT) and available bandwidth could change dramatically during a short period of time. Acquiring real-time network performance information along the path from the client to the servers would incur extra latency on fetching a document.

2. automatic request re-direction;

3. dynamic client cluster thresholding.

These tasks are discussed in the following sections.

4.2.1 Good server set selection

After issuing a request to the server, the client perceived latency [3] R of fetching a document of size L is the sum of the round trip time (RTT) and the transmission time of the document (BW is the bandwidth of the network):

$$R = RTT + L/BW$$

If L is small, the client perceived latency of fetching a document is dominated by the RTT between the client and the server. Fetching small documents from a closer server would result in a lower perceived latency. If L is large, the available (bottleneck) bandwidth of the network connection between the client and the server becomes crucial. Fetching large documents from a server which has good connectivity (i.e., high bandwidth) to the client would result in a lower perceived latency.

Instead, we propose a scheme selecting good servers on the basis of client clusters. For each busy client cluster, a set of good servers is selected based on a "follow-the-herd" heuristic, that is, if a large portion of the requests issued from a client cluster are sent to a typical server in the past, then we consider this server to be a good server of the client cluster. We summarize the procedure of selecting the good server set here:

[3] We do not consider the latency of setting up a TCP connection between the client and the server, nor the latency at the server to process the request.

Table 4.2: The Digital Library server logs: 1 main server and 14 mirror servers.

Server log	Location	Number of clients	Number of requests
US-MAIN	US	58,919	1,347,859
US-MIRROR	US	429	8,160
BR	Brazil	4,167	46,699
CN	China	632	24,586
DE	Germany	4,369	149,333
ES	Spain	1,559	51,958
FR	France	3,292	110,532
IL	Israel	949	22,913
IN	India	252	7,024
IT	Italy	5,788	186,422
JP	Japan	3,948	98,365
KR	Korea	1,268	91,239
RU	Russia	1,442	73,381
TW	Taiwan	592	11,613
UK	U. K.	3,077	242,154
Total	Global	75,270	2,472,238

1. Client cluster detection;

2. Client cluster thresholding;

3. Good server set identification.

Our experiments are conducted on a set of mirror server logs of a digital library of science articles consisting of one main server and 14 mirror servers distributed globally (Table 4.2). The 13-day server logs were taken during the period of 3/19/2000 to 3/31/2000. A total of 75,270 clients visited the digital library site and issued 2,472,238 requests. The number of unique clients in each server log varies from 252 to 58,191, the number of requests in each server log varies from 7,024 to 1,347,859. Around half (55%) of the total requests were served by the main server.

The experimental results of client cluster detection of the digital library server

Table 4.3: Experimental results of client cluster detection for each server log on the digital library site.

Server log	Number of client clusters	Largest client cluster (clients)	Busiest client cluster (requests)
US-MAIN	8,545	471	11,172
US-MIRROR	232	39	1,192
BR	1,642	94	3,501
CN	181	58	6,710
DE	1,013	108	8,737
ES	286	172	7,196
FR	806	134	8,607
IL	363	226	8,343
IN	123	29	2,646
IT	1,398	276	14,355
JP	766	209	8,624
KR	379	125	3,728
RU	386	63	26,320
TW	211	31	1,497
UK	929	289	111,819

logs are shown in Table 4.3. The client cluster detection is conducted separately on each of the 15 digital library server logs. The number of client clusters detected in each server log varies from 123 to 8,545. The size of the largest client cluster in each server log varies from 29 clients to 471 clients. The number of requests issued from the busiest client cluster in each server log varies from 1,497 to 111,819.

We filter out non-busy client clusters of each server log by thresholding on the number of requests issued within a client cluster, at the 70% threshold (Table 4.4). The threshold used to filter out non-busy client clusters in each server log varies from 61 requests to 1,113 requests. The number of busy client clusters in each server log varies from 5 to 385. The number of clients in busy client clusters of each server log varies from 68 to 33,138.

After identifying busy client clusters, we take the union of all the busy client clusters in each server log to compose the *busy client cluster list* which consists of

Table 4.4: Experimental results of client cluster thresholding for each server log on the digital library site.

Server log	Threshold (requests per client cluster)	# of busy client clusters	Total # of clients in busy client clusters	Total # of requests issued from busy client clusters
US-MAIN	385	714	33,138	925,075
US-MIRROR	61	26	141	5,723
BR	94	69	1,365	29,068
CN	858	5	99	16,634
DE	589	49	1,700	102,313
ES	1,113	11	624	37,371
FR	809	30	1,272	76,967
IL	489	7	359	16,306
IN	175	8	68	5,014
IT	495	62	2,430	129,100
JP	410	41	1,627	68,697
KR	148	29	364	16,017
RU	778	16	417	51,773
TW	255	14	147	8,240
UK	5,031	6	370	170,588

a total of 868 busy client clusters. For each client cluster in the busy client cluster list, the good server set consists of all the servers that views it as a busy client cluster [4].

It is hard to validate if the good servers selected on the basis of the "follow-the-herd" heuristic will provide a lower user-perceived latency than other non-good servers. In our experiments on the digital library servers, though the server could use network tools *traceroute* and *pathchar* to measure the network latency and bandwidth along the path from the server to the clients, we do not have direct access to the digital library servers. Another way to collect the network performance measurement between the servers and the clients is to use the loose source routing option [WS94]. The source routing probes can be forced to traverse along a path passing by a typical server towards the clients. However, short of support for source routing at IP routers, all the source routing packets will be dropped at the routers which do not support source routing. The best we can do is to make use of the public *traceroute* gateways[5] to measure the network RTT between the server and its clients.

We sample 10% of client clusters in the busy client cluster list and randomly select one client from each sampled busy client cluster. For each of the servers, we use a nearby public *traceroute* gateway and run *traceroute* from the *traceroute* gateway to the sampled clients. We approximate the network latency between a server and a busy client cluster by the average network latency between the

[4]Note that a client cluster may behave as a busy client cluster in more than one server log, the good server set of a busy client cluster may contain multiple servers.

[5]One may use the public *traceroute* gateways to trace to any user-specified host from host located at various points on the Internet. A list of *traceroute* gateways is available at http://www.traceroute.org. We are not aware of any public *pathchar* gateways.

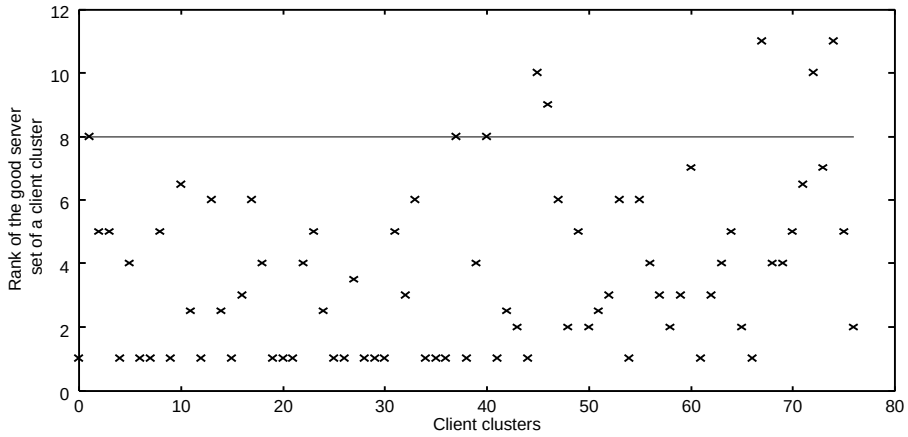


Figure 4.8: The rank of the good server set of the sampled busy client clusters.

traceroute gateway and the sampled client. The measurement is conducted between each server and each client cluster in the busy client cluster list. For each sampled busy client cluster C, after collecting the network latency (RTT) between C and each server, we rank the server in the increasing order of RTT (i.e., the server with rank 1 has the lowest RTT). The rank of a good server set is the average of the ranks of all the servers in the good server set. For example, we sampled 77 busy client clusters in the digital library server logs. The minimum rank and maximum rank of a server are 1 and 15, respectively. The ranks of the good server set of sampled busy client clusters are shown in Figure 4.8. The client clusters on the x-axis are ordered by their prefixes. The median rank of digital library servers is 8 (shown as the horizontal line in Figure 4.8). We observe that the vast majority of the sampled busy client clusters have a good server set whose rank is below 8. Though there is not a perfect correlation between the network latency (RTT) and the user-perceived latency of fetching a document, the *traceroute* results provide a justification of our "follow-the-herd" heuristic.

4.2.2 Automatic request redirection

Each server maintains a request re-direction table with one entry for each client cluster in the busy client cluster list. A set of good servers is associated with each busy client cluster. A client can send requests to an arbitrary server. Upon receiving a request, a server first detects the client cluster which the client belongs to, by applying longest prefix matching. If the client does not belong to a busy client cluster, then the server serves the request as usual. If the client belongs to a busy client cluster, the server looks up the request re-direction table to find the good server set of the client cluster. If the server itself is in the good server set, it will serve the request as usual. Otherwise, the server will randomly select a server from the good server set and re-direct the request to the selected server. Thus, no matter which server the client chooses to send requests to, the requests are always served by one of the good servers corresponding to the client.

4.2.3 Dynamic client cluster thresholding

Each server maintains the good server set for each client cluster in the busy client cluster list. As requests arrive at the server, the set of busy client clusters of the server may change. Thus we need a scheme to update the set of busy client clusters dynamically based on user access patterns.

We keep two data structures: D_1 on the full set of client clusters and D_2 on the busy client clusters. We also maintain two counters: N_1 counts the total number of requests and N_2 counts the number of requests from busy client clusters. Initially, the busy client clusters are identified as we described in Chapter 4.2.1. The total number of requests issued from the busy client clusters account for $t_1\%$ of the total requests received at the server (i.e., $N_2/N_1 \geq t_1\%$). We maintain the invariant

that, at all times, the current set of busy client clusters accounts for at least $t_2\%$ of the total requests (i.e., $N_2/N_1 \geq t_2\%$), for some $t_2 < t_1$ (we chose $t_1 = 70$ and $t_2 = 65$ in our experiments). When a request arrives at the server, the server uses D_2 to make a service decision (e.g. request re-directing) and use D_1 to construct the clients access pattern. If $N_2/N_1 < t_2\%$, we recompute busy client clusters and reconstruct D_2 accordingly. There are at least $(t_1 - t_2)N_{last}$ requests arriving at the server between two re-construction of the busy client cluster data structure D_2, where N_{last} is the total number of requests at the time of the recent re-construction of D_2.

4.2.4 Summary

We have proposed a scheme that can be deployed at Web servers to automatically re-direct client requests to one of the good servers to reduce the access latency perceived by clients. We select good servers for a client cluster based on the "follow-the-herd" heuristic. We also proposed a mechanism to reconstruct the busy cluster list dynamically . We conducted experiments on a set of server logs of a digital library site. Although, due to lack of access to the digital library servers, we cannot validate if our selection of the good servers will provide a lower user-perceived latency than other non-good servers, our *traceroute* results show that our selection of good servers usually have a smaller RTT to the client clusters than that of other servers.

4.3 Discussion

In this chapter, we discussed two applications of the network-aware clustering of IP addresses: Web caching and server replication. Our results show that the network-aware clustering of IP addresses is very helpful to obtain a good understanding of how clients' behave and the geographical pattern of the requests and to improve the performance of applications by proposing new schemes. Its applications are not limited to the ones we presented in this chapter. Other applications include content distribution networks, Internet topology discovery, traffic management, and TCP performance improvement. For example, on a content distribution network, the network-aware clustering of IP addresses can be used to monitor the traffic pattern and identify the bottleneck and hot spots if any and help the content distribution provider to manage the traffic load and determine the optimal place to install new devices.

Chapter 5

Improvements to Network-Aware Clustering

In Chapters 3 and 4, we described client cluster detection based on information obtained primarily from Web server logs. The resulting client clusters were grouped on the basis of prefixes and the "interesting" client clusters were formed based on the number of requests emanating from each client cluster. However, there are several issues need to be examined to further improve the scheme. In this chapter, we examine four potential improvements on the basic client clustering method and the applicability of server clustering.

First, in the validation tests proposed in Chapter 3, we only examined if a cluster is too big. Our current client clusters might be too fine-grained, that is, multiple client clusters might actually belong together. Our clustering does not discriminate on the basis of which AS an IP address prefix originates from. Since the originating AS information is available in the BGP tables (based on which the client cluster detection is actually done), we use this additional information to see if our clustering is indeed too fine-grained.

Second, we assemble interesting client clusters based on a thresholding scheme that includes only clusters that contribute 70% of the total number of requests. We rely on the fact that the *busy* client clusters are distinct, but it may still be possible for several *non-busy* client clusters to be grouped together and end up in the interesting set. In this chapter, we examine if such a case exists. Then we will also examine if the arbitrarilly chosen threshold – 70% – is reasonable.

Third, our client clusters are constructed based on prefixes extracted from BGP routing tables. However, some applications may be interested in an even more fine-grained partition of clients. For instance, one may want to partition a large client cluster into sub-clusters based on the clients' access patterns. We show that sub-clustering os helpful to identify spiders and proxies and to select good servers for busy client clusters.

Finally, symmetric to client clustering, we show that our method is also applicable to server clustering.

5.1 Super-cluster construction

The client clusters that are constructed on the basis of IP address prefix information extracted from BGP routing tables might be too fine-grained since we did not take into account the ASes originating the IP address prefix and route of a IP address prefix during cluster detection. Here, we first construct clusters at the AS level by grouping client clusters which belong to the same AS into a *super-cluster* (Figure 5.1 (a)). We then use a *dig*[1]-based validation test on sampled super-clusters

[1] *Dig* (domain information groper) is a flexible command line tool which can be used to gather information from the Domain Name System servers. Please refer to http://www.stopspam.org/usenet/mmf/man/dig.html for details.

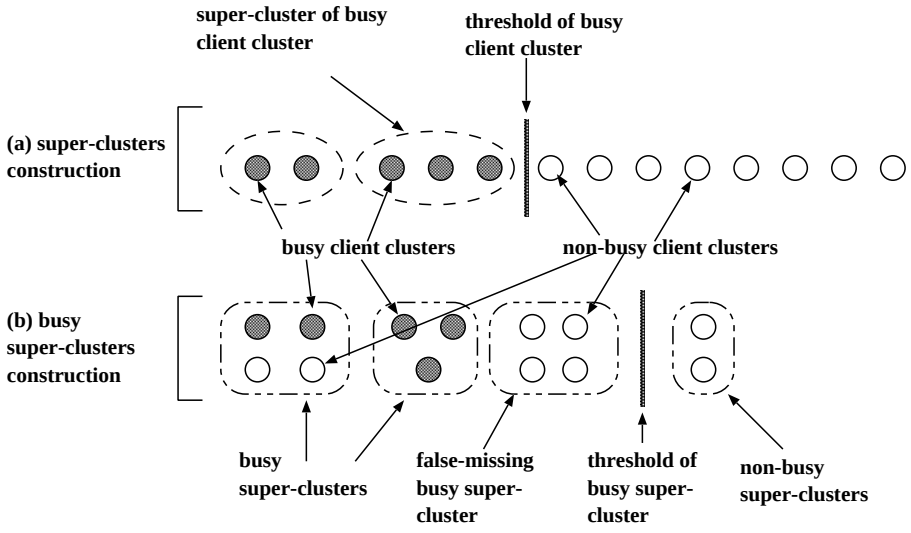


Figure 5.1: The construction of (a) super-clusters of busy client clusters and (b) busy super-clusters.

to see if AS-level super-clustering is too coarse-grained.

The super-clusters are constructed in four steps. First, clusters are detected using IP address prefix information extracted from BGP routing tables. Next, clusters contributing to a large portion of the requests to the server are labeled as *busy* client clusters. After reverse sorting based on the number of requests issued from within client clusters, clusters whose total requests contribute to 70% of the total requests in the server log are included in the busy cluster set. The first three steps are similar to client cluster detection in Chapters 3 and 4. Finally, we identify the ASes originating the IP address prefixes of the *busy* client clusters and group busy client clusters whose IP adress prefixes originating from the same AS into one super-cluster. The super-cluster is identified by the AS number. For example, both client cluster 24.48.7/255.255.255 and client cluster 63.249.128/255.255.255 originate from AS 701, and are thus grouped into super-cluster AS 701.

In the experiments, we use the Oregon BGP routing table dump taken on

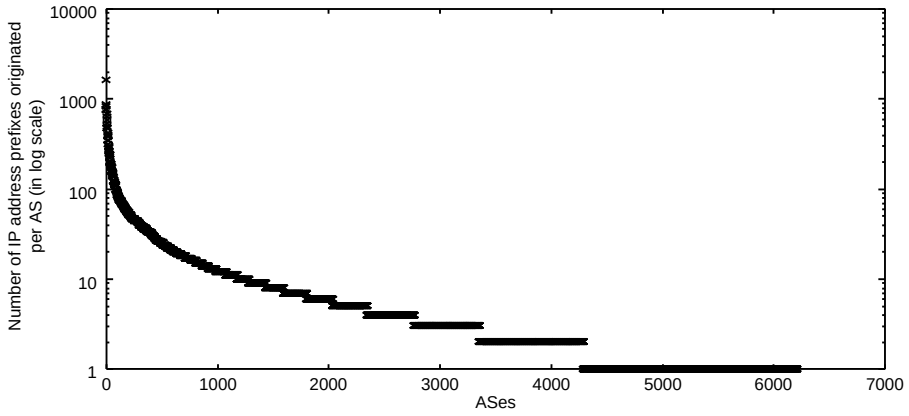


Figure 5.2: Number of IP address prefixes originate from ASes plotted in the reverse order of the number of IP address prefixes originating from an AS.

Table 5.1: Number of IP address prefixes originating from multiple ASes.

Number of ASes originating the same IP address prefix	1	2	3	4	5	7
Number of IP address prefixes originated from ASes	67,732	819	24	4	2	3

December 6, 1999, which contains 68,584 unique IP address prefixes originating from 6,213 unique ASes. Figure 5.2 shows a distribution of the number of unique IP address prefixes originating from an AS. The ASes are sorted in the reverse order of the number of unique IP address prefixes originated. The number of IP address prefixes originated from an AS ranges from 1 to 1,614 with an average of 11 (Figure 5.2). In most cases, an IP address prefix has one originating AS. An IP address prefix ($\sim 1.25\%$ of the total) may have multiple originating ASes (Table 5.1). For example, there are 819 IP address prefixes which have 2 originating ASes. If an IP address prefix is originating from multiple ASes, we consider the first AS as its primary originating AS. The super-clusters are constructed based on the primary originating AS of the IP address prefix.

Table 5.2: The server logs used in our experiments.

Server logs	Date	Number of requests	Number of clients
Apache	10/1/1999 - 11/18/1999	3,461,361	274,844
EW3	12/1/1999 - 12/20/1999	1,847,913	38,913
Nagano	2/13/1998	11,665,713	59,582
Sun	9/30/1997 - 10/9/1997	13,871,352	219,528

Table 5.3: Experimental results of client cluster detection on the Apache, EW3, Nagano, and Sun server logs.

Server logs	Total number of client clusters	Largest client cluster (clients)	Busiest client cluster (requests)
Apache	18,087	2,555	78,336
EW3	8,459	1,476	148,493
Nagano	4,607	2,279	389,861
Sun	16,901	4,299	693,955

We conducted our experiments on the Apache, EW3, Nagano, and Sun server logs (Table 5.2). Experimental results of client cluster detection using BGP routing tables on the Apache, EW3, Nagano, and Sun server logs are shown in Table 5.3. For example, in the Apache server log, all the clients are first grouped into 18,087 client clusters. The size of the client clusters varies from 1 to 2,555 clients, the number of requests issued from within each client cluster varies from 1 to 78,336.

Next, Table 5.4 shows the experimental results of thresholding client clusters of the Apache, EW3, Nagano, and Sun server logs. In the Apache server log, we set the threshold to be 418 and filter out all the client clusters that issued fewer than 418 requests. There are 1,574 busy client clusters containing 173,529 clients contributing 2,391,009 requests (i.e., 70% of the total requests).

Finally, we use the originating AS information extracted from BGP routing tables to group busy client clusters into super-clusters. In our experiments, we

Table 5.4: Experimental results of thresholding client clusters on the Apache, EW3, Nagano, and Sun server logs.

Server logs	Threshold (requests per client cluster)	# of busy client clusters	Total # of clients in busy client clusters	Total # of requests issued from busy client clusters
Apache	418	1,574	173,529	2,391,009
EW3	347	931	22,483	1,291,148
Nagano	6,318	339	35,153	8,073,614
Sun	1,836	1,355	125,681	9,510,453

Table 5.5: Busy client clusters with an IP address prefix originated from multiple ASes.

Number of ASes originating an IP address prefix	1	2	3
Apache	1,540	32	2
EW3	904	26	1
Nagano	324	15	0
Sun	1,323	31	1

only consider the primary originating AS of each IP address prefix though there is a small, but visible, portion of busy client clusters with an IP address prefix originating from multiple ASes (Table 5.5). For instance, among a total of 1,574 busy client clusters in the Apache server log, there are 32 busy client clusters with an IP address prefix originating from 2 ASes and 2 busy client clusters with an IP address prefix originating from 3 ASes.

The experimental results of super-cluster construction of busy client clusters in the Apache, EW3, Nagano, and Sun server logs are shown in Table 5.6. In the Apache server log, the busy client clusters are grouped into 761 super-clusters. The number of client clusters in each super-cluster varies from 1 to 41, the number of clients in each super-cluster varies from 1 to 7,428, and the number of requests issued from within each super-cluster varies from 1 to 90,887. The fattest super-

Table 5.6: Experimental results of super-cluster construction of busy client clusters on the Apache, EW3, Nagano, and Sun server logs.

Server logs	Number of super-clusters	Number of super-clusters containing multiple client clusters	Fattest super-cluster (client clusters)	Largest super-cluster (clients)	Busiest super-cluster (requests)
Apache	761	253	41	7,428	90,887
EW3	429	139	30	2,805	158,624
Nagano	185	55	18	6,463	1,273,140
Sun	639	212	42	7,747	707,939

cluster is the super-cluster which has the largest number of client clusters. We observe that the client clusters do not collapse dramatically. Only 253 super-clusters ($\sim 30\%$ of the total) contain multiple busy client clusters. The average number of busy client clusters in each super-cluster is 2. This observation holds across all the server logs in our experiments. Thus, we may conclude that the prefix-level clustering is not too fine-grained.

Figures 5.3(a), (b), and (c) show the number of busy client clusters in super-clusters, the number of clients in super-clusters, and the number of requests issued within super-clusters. These are plotted in the reverse order of the number of busy client clusters in a super-cluster (i.e., super-clusters consisting of a larger number of busy client clusters are on the left side). From Figure 5.3(a), we observe that around one-third of the super-clusters consist of more than one busy client cluster. Super-clusters, which consist of a larger number of busy client clusters, likely consist of a larger number of clients (Figure 5.3(b)). While we showed in Chapter 3 that larger client clusters usually issue more requests, the correlation between the number of busy client clusters in a super-cluster and the number of clients in a super-cluster is weak, which shows that the prefix-based client clusters do a better

job on capturing clients' behavior than AS-based super-clusters. Super-clusters, which consist of a larger number of busy client clusters, usually contribute to a larger number of requests. However, as we observe from Figure 5.3(c), there are some super-clusters which consist of just 1 busy client cluster and issue many requests.

We re-plot the same set of data shown in Figure 5.3 with a different sorting of super-clusters (on the x axis) – in reverse order of the number of requests issued within a super-cluster. Figures 5.4(a), (b), and (c) show the number of busy client clusters in super-clusters, the number of clients in super-clusters, and the number of requests issued within super-clusters. Figure 5.4(a) further shows that there is strong correlation between the number of busy client clusters in a super-cluster and the number of requests issued within a super-cluster. We showed in Chapter 3 that larger client clusters usually issued more requests, but the correlation between the number of requests issued within a super-cluster and the number of clients in a super-cluster is weak (Figure 5.4(b)), which further shows that the prefix-based client clusters do a better job on capturing clients' behavior than AS-based super-clusters. Figure 5.4(c) shows that more than 90% of the super-clusters issued fewer than 10,000 requests.

In fact, the AS-level super-clusters may be too coarse-grained. An AS can be very large and may contain several smaller entities which are separately administered. For example, the super-cluster AS 1221 in the Apache server log contains 8 busy client clusters. The clients in each busy client clusters share a common suffix in their name. The common client name suffix of each client cluster is listed in Table 5.7. The name suffixes of client clusters are different from each other. They are unlikely to be under the same administration and should be treated as

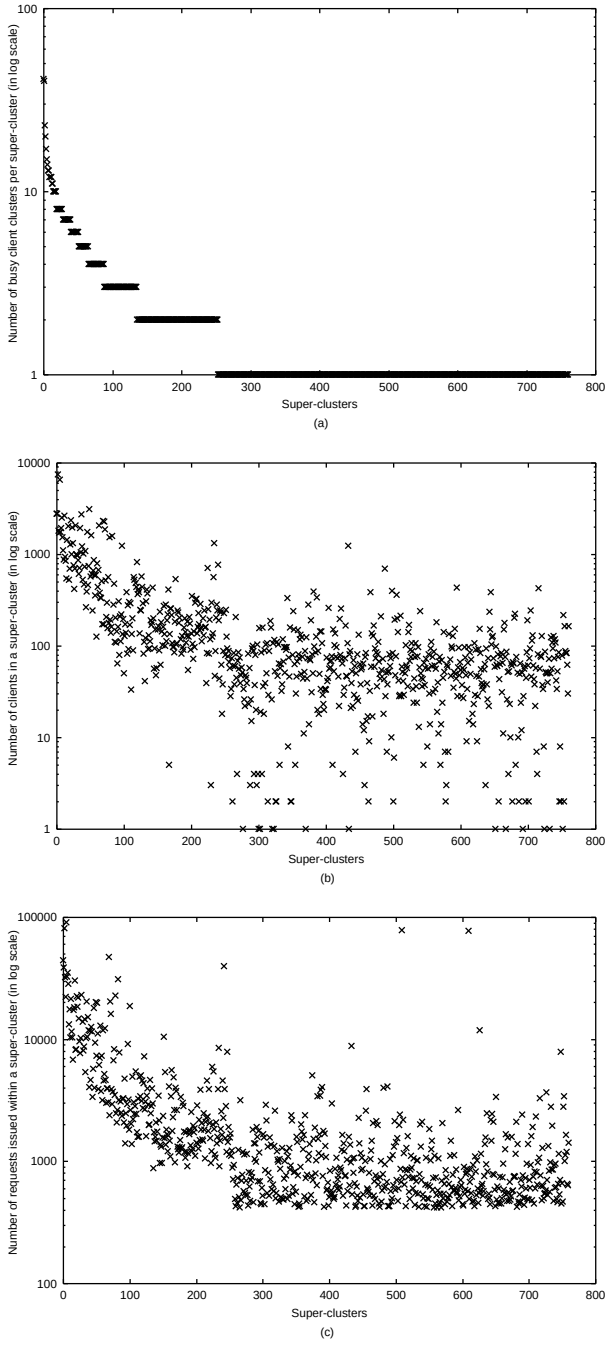


Figure 5.3: The super-cluster distribution of the Apache server log plotted in the reverse order of the number of busy client clusters in a super-cluster (y axis is in log scale): (a) number of busy client clusters in super-clusters; (b) number of clients in super-cluster; (c) number of requests issued within super-clusters.

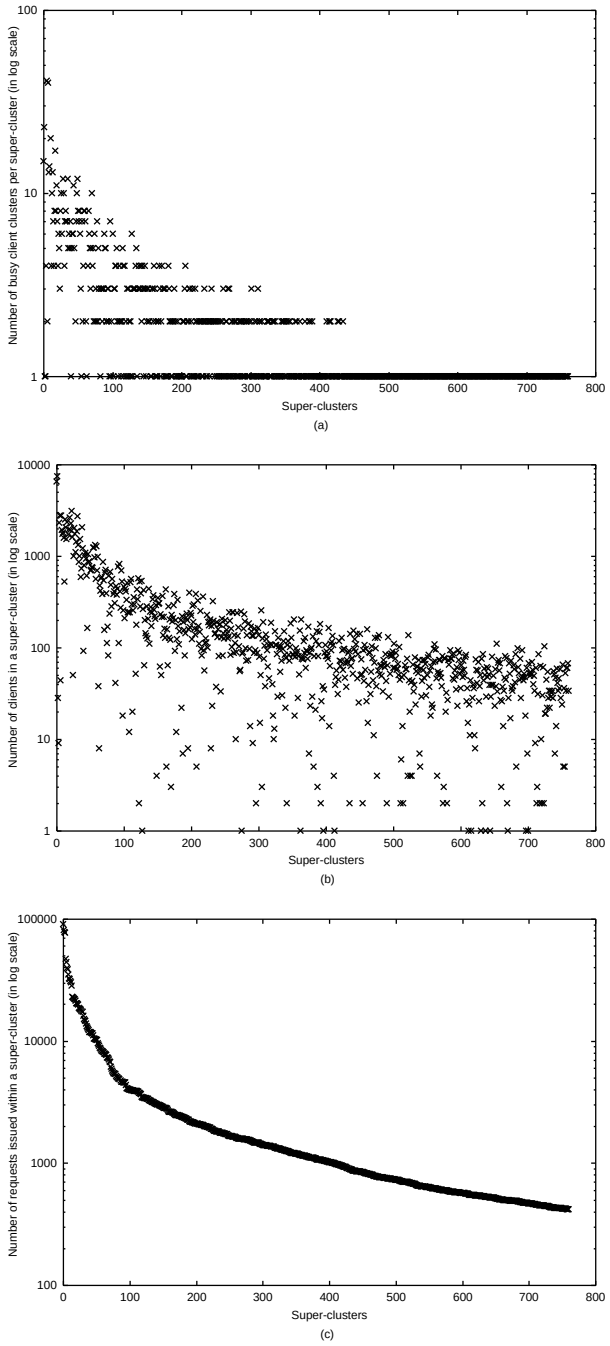


Figure 5.4: The super-cluster distribution of the Apache server log plotted in the reverse order of the number of requests issued within a super-cluster (y axis is in log scale): (a) number of busy client clusters in super-clusters (re-plot of Figure 5.3(a)); (b) number of clients in super-clusters (re-plot of Figure 5.3(b)); and (c) number of requests issued within super-clusters (re-plot of Figure 5.3(c)).

Table 5.7: The client clusters in super-cluster AS 1221 in the Apache server log.

Client cluster prefix/netmask	Common suffix of client name
139.130.0.0/16	wnise.com
139.134.0.0/16	tmns.net.au
192.148.160.0/24	telstra.com.au
203.32.0.0/14	ocs.com.au
203.36.0.0/16	trickysoft.com.au
203.38.0.0/16	panorama.net.au
203.0.0.0/10	geelong.netlink.com.au
203.0.0.0/12	iaccess.com.au

separate entities.

We conducted a *dig*-based validation test to show that the prefix-level client cluster detection is necessary. We sampled 15% of the super-clusters that had multiple busy client clusters and randomly selected one client in each busy client cluster belonging to the sampled super-clusters. We ran *dig -x* on these selected clients to resolve their names to see if the client clusters belonging to the same super-cluster are indeed under the same administration. We define a super-cluster to be *false-partitioned* if the common client name suffix of the busy client clusters belonging to it are the same. The results of this *dig*-based validation test of the Apache, EW3, Nagano, and Sun server logs are shown in Table 5.8. In the Apache server log, we sampled 45 super-clusters which together contain 165 busy client clusters. The results show that 3 super-clusters are false-partitioned, which contain a total of 7 ($< 5\%$) busy client clusters. This observation holds across all the server logs, which demonstrates the need for constructing prefix-level client clusters.

Table 5.8: Client cluster validation of the Apache, EW3, Nagano, and Sun server logs using *dig*.

Server logs	Number of sampled super-clusters	Total number of busy client clusters in sampled super-cluster	Number of false-partitioned super-clusters	Total number of busy client clusters in false-partitioned super-clusters
Apache	45	165	3	7
EW3	21	96	1	3
Nagano	7	22	0	0
Sun	35	157	2	7

5.2 Busy super-clusters

In previous chapters, our construction of interesting client clusters was based on a thresholding scheme on the number of requests issued from within a client cluster. The scheme includes client clusters that contribute 70% of the total number of requests, and relies on the fact that most of the *busy* client clusters are distinct. Even if this were always true, it is possible for several non-busy clusters to be grouped together and end up in the interesting set. Here, we examine if non-busy clusters, belonging to the same AS, can be combined into a *busy* super-cluster (Figure 5.1 (b)).

Our experiment has four steps, beginning with client cluster detection as before. Then we construct super-clusters by identifying the ASes originating the IP address prefixes of both busy and non-busy client clusters, and group client clusters which originated from the same AS into one super-cluster. Next, we threshold the super-clusters since we are interested in the *busy* super-clusters responsible for a large portion of the requests. We label the set of super-clusters of busy client clusters constructed in Section 5.1 to be *tentatively-busy*. After reverse sorting based on the number of requests issued from within super-clusters, we retain only the *busy*

Table 5.9: Experimental results of super-cluster construction of client clusters (both busy and non-busy client clusters) on the Apache, EW3, Nagano, and Sun server logs.

Server log	Number of super-clusters	Fattest super-cluster (client clusters)	Largest super-cluster (clients)	Busiest super-cluster (requests)
Apache	3,728	322	8,255	99,127
EW3	2,635	174	3,043	163,450
Nagano	1,291	186	6,919	1,396,442
Sun	3,628	380	8,971	723,714

super-clusters whose total request contribution added up to at least 70% of the total number of requests in the server log. We examine if there are any busy super-clusters consisting of non-busy client clusters and label them as *false-missing busy super-clusters*. These are the busy super-clusters that are not in the set of tentatively-busy super-clusters.

Our experiments are conducted on the Apache, EW3, Nagano, and Sun server logs. The client cluster detection is the same as we did in Section 5.1 and we do not repeat the results here. Experimental results of super-clusters construction on all client clusters is shown in Table 5.9. For instance, in the Apache server log, all 18,087 client clusters are grouped into 3,728 super-clusters. The number of client clusters in a super-cluster varies from 1 to 322, the number of clients in a super-cluster varies from 1 to 8,255, and the number of requests issued from within a super-cluster varies from 1 to 99,127.

Next, we filter out non-busy super-clusters. Table 5.10 shows the experimental results of thresholding super-clusters on the Apache, EW3, Nagano, and Sun server logs. In the Apache server log, we filter out non-busy super-clusters which issued fewer than 2,266 requests in the Apache server log. There are 263 busy super-

Table 5.10: Experimental results of thresholding super-clusters on the Apache, EW3, Nagano, and Sun server logs.

Server log	Threshold (requests per super-cluster)	# of busy super-clusters	Total # of client clusters in busy super-clusters	Total # of clients in busy super-clusters	Total # of requests issued from busy super-clusters
Apache	2,266	263	7,804	178,428	2,389,270
EW3	1,784	128	2,947	24,108	1,289,282
Nagano	25,470	76	1,523	36,496	8,059,611
Sun	11,249	196	6,619	130,417	9,502,880

clusters, containing 7,804 client clusters and 178,428 clients and issuing 2,389,270 requests ($\sim$ 70% of the total requests).

Table 5.11 shows the false-missing busy super-clusters. In the Apache server log, 5 out of 263 busy super-clusters ($<$ 2% of the busy super-clusters) are false-missing busy super-clusters, which together contain 216 client clusters (1% of the total 18,087 client clusters) and 1,439 clients (0.5% of the total 274,844 clients) issuing 14,172 requests (0.4% of the total 3,461,361 requests). Thus it is unlikely that non-busy client clusters can be grouped into a busy super-cluster. This also indicates that the chosen threshold of 70% is reasonable.

5.3 Sub-clustering

All the clients belonging to the same client cluster are under the same administrative control. However, some applications may be interested in an even more fine-grained partition of clients. Our goal in this section is to see if a busy client cluster can be further partitioned into several sub-clusters based on the clients' access pattern. The four steps of the sub-clustering experiments begins with client

Table 5.11: The false-missing busy super-clusters of the Apache, EW3, Nagano, and Sun server logs.

Server log	Number of false-missing busy super-clusters	Total # of client clusters in false-missing busy super-clusters	Total # of clients in false-missing busy super-clusters	Total # of requests issued from false-missing busy super-clusters	Percentage of requests issued from false-missing busy super-clusters
Apache	5	216	1,439	14,172	0.4
EW3	2	50	73	3,891	0.2
Nagano	2	32	482	59,846	0.5
Sun	0	0	0	0	0

cluster detection of the clients extracted from the server logs. After reverse sorting based on the number of requests issued from within client clusters, we retain *busy* client clusters as before. Next, we identify the set of *common busy client clusters*—busy client clusters seen in more than one server log. Finally, we partition a common busy client cluster into sub-clusters based on the clients' access patterns. The clients that usually access the same set of servers will be in the same sub-cluster.

Our experiments are conducted on the same set of digital library server logs we used in Section 4.2. Here, we provide a summary on the server logs and the experimental results. The experiments of client cluster detection and thresholding are same as those in Section 4.2. We take the union of all the busy client clusters in each server log—there are 868 unique busy client clusters. Figure 5.5 shows the number of digital library servers a busy client cluster frequently accesses. Around one fourth of the busy client clusters are common busy client clusters. While most of the common busy client clusters usually access 2 servers, a client cluster may behave as a busy client cluster at up to 6 servers. The 196 common busy client

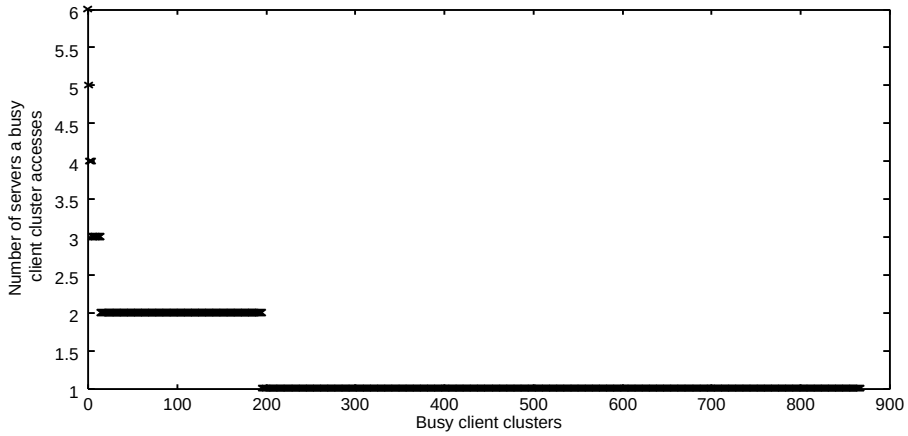


Figure 5.5: The number of digital library servers which the busy client clusters access frequently (plotted in the reverse order of the number of servers a busy client cluster frequently accesses).

clusters consist of a total of 15,433 clients and issue 909,315 requests (Table 4.3). The number of common busy client clusters in each server log varies from 3 to 191. The number of clients in the common busy client clusters in each server log varies from 64 to 10,376. The number of requests issued by the common busy client clusters in each server log varies from 2,619 to 297,012.

We denote the set of servers which is frequently accessed by a busy client cluster as its *good server set*. A busy client cluster can be partitioned into sub-clusters in the following ways:

1. A small portion of the clients in the client cluster sends many requests to each server in the good server set while the remaining clients are less active. Examples include proxies and spiders, which we can differentiate from the normal clients as we did in Chapter 4. This technique partitions the client cluster into three sub-clusters: spider, proxy, and normal-client.

2. Often, a subset of the clients send requests to a subset of servers in the good server set, while other clients send requests to the remaining servers in the

Table 5.12: The common busy client clusters in the digital library server logs.

Server log	Number of common busy client clusters	Total number of clients in the common busy client clusters	Total number of requests issued within the common busy client clusters
US-MAIN	191	10,376	297,012
US-MIRROR	14	158	2,619
BR	19	665	15,608
CN	5	122	17,123
DE	38	2,102	75,818
ES	6	632	31,734
FR	25	1,521	72,931
IL	3	130	4,557
IN	8	80	5,114
IT	49	2,634	123,233
JP	21	1,466	54,428
KR	8	225	8,914
RU	13	458	49,571
TW	5	64	4,099
UK	3	861	146,554
Total	196	15,433	909,315

good server set. The client cluster is partitioned into multiple sub-clusters based on the subset of servers the clients usually contact.

We use an example to illustrate the second case of sub-cluster construction. The common busy client cluster 130.183.0.0/255.255.0.0 behaves as a busy client cluster in both the main server in the US and the mirror server in Germany (i.e., the good server set of client cluster 130.183.0.0/255.255.0.0 consists of the main server in the US and its mirror server in Germany). It consists of 90 clients and issued a total of 7K requests to these two servers (40% of the requests are sent to the main server in the US and 60% of the requests are sent to the mirror server in Germany). Figure 5.6 shows the percentage of requests issued from a client to each server in the good server set. It is sorted first on the reverse order of the number of requests issued to the main server in the US and second on the increasing order

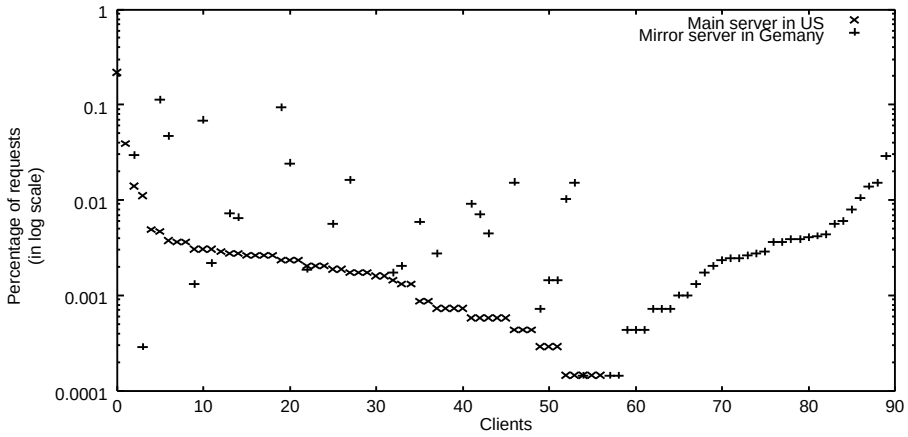


Figure 5.6: Percentage of requests issued by each client in common busy client cluster 130.183.0.0/255.255.0.0 to the main server in the US and its mirror server in Germany (y-axis is in log scale).

of the number of requests issued to the mirror server in Germany. Most clients usually access only one server. The clients can be partitioned into two sub-clusters based on the server the clients access more frequently. A client which issues more requests to the main server in the US than that to the mirror server in Germany belongs to the sub-cluster corresponding to the main server in the US (on the left side in Figure 5.6). Otherwise, it belongs to the sub-cluster corresponding to the mirror server in Germany (mainly on the right side in Figure 5.6).

In summary, sub-clustering provides us with finer-grained information on the busy clusters. Examples of potential applications of sub-clustering include Web caching and server replication.

5.4 Server clustering

Symmetric to client cluster detection, our method can be used to detect *server clusters*. Server clustering is conducted on the server addresses extracted from the proxy logs in front of a set of users. We apply server clustering to two sets of proxy

Table 5.13: The AT&T proxy logs used in our experiments.

Date	Number of requests	Number of servers
6/20/2000	995,917	1,328
6/21/2000	997,560	2,517
6/22/2000	997,098	3,159
6/23/2000	997,045	3,014
6/24/2000	991,986	1,980
6/25/2000	992,048	1,892
6/26/2000	996,527	3,545
6/27/2000	997,159	2,910
6/28/2000	997,274	2,813

Table 5.14: The Larmancom proxy logs taken from 4/9/2000 to 4/15/2000.

Proxy log	Number of requests	Number of servers
Blv	150,537,713	226,521
Bna	16,197,389	66,634
Slb	44,586,509	120,047
Stl	67,451,391	146,624

logs: AT&T and Larmancom proxy logs. The AT&T proxy logs are obtained from a 9-day packet trace at an AT&T facility between 6/20/2000 and 6/28/2000 with over 1 Million requests a day and between 1,328 and 3,545 servers contacted (Table 5.13). The Larmancom proxy logs are a set of four 7-day packet traces at a large manufacturing company that wishes to remain anonymous: Blv is a proxy in front of one set of users, Bna, Slb, and Stl are proxies associated with regional firewalls in front of three sets of users (Table 5.14).

5.4.1 Server cluster detection

Server cluster detection on the proxy logs is similar to client cluster detection in Chapter 3 on server logs except that the experiments are conducted on server IP addresses extracted from the proxy logs.

Table 5.15: Experimental results of server cluster detection on the AT&T proxy logs.

Date	Total number of server clusters	Largest server cluster (servers)	Busiest server cluster (requests)
6/20/2000	506	38	407,440
6/21/2000	907	80	174,055
6/22/2000	1,148	93	136,368
6/23/2000	1,121	85	90,483
6/24/2000	817	49	115,160
6/25/2000	680	49	77,184
6/26/2000	1,304	85	146,988
6/27/2000	1,063	84	101,704
6/28/2000	1,063	76	129,408

AT&T proxy logs

In the AT&T proxy logs, more than 99% of the servers can be grouped into server clusters. The experimental results of the AT&T proxy logs are shown in Table 5.15. The number of server clusters varies from 506 to 1,304, where the size of the largest server cluster varies from 38 servers to 90 servers. The number of requests received by the busiest server clusters varies from 77,184 to 407,440.

Figures 5.7(a), (b), and (c) show the cumulative distribution of the number of requests in a server cluster, the distribution of the number of requests in a server cluster, and the distribution of the number of servers in a server cluster. Each curve in Figure 5.7 corresponds to a 1-day proxy log. The top 100 server clusters are plotted in the reverse order of the number of requests received by a server cluster (i.e., server clusters receiving a larger number of requests are on the left side). From Figures 5.7(a) and (b), we observe that the top 20 busiest server clusters received 50% of the requests. This implies that the proxy can do a better job on prefetching and caching after locating the busy server clusters. Unlike our observation on the client cluster distribution in Chapter 3, there is no perfect

Table 5.16: Experimental results of server cluster detection on the Larmancom proxy logs.

Proxy log	Total number of server clusters	Largest server cluster (servers)	Busiest server cluster (requests)
Blv	174,66	2,559	15,199,709
Bna	10,722	750	1,617,901
Slb	13,795	1,360	4,440,324
Stl	14,947	1,673	7,692,163

correlation between the number of servers in a server cluster and the number of requests received in a server cluster (Figure 5.7(c)).

We re-plot the same set of data shown in Figure 5.7 with a different sorting of server clusters (on the x axis) – in reverse order of the number of servers in a server cluster. Figures 5.8(a), (b), and (c) show the cumulative distribution of the number of servers in a server cluster, the distribution of the number of servers in a server cluster, and the distribution of the number of requests in a server cluster. Figure 5.8(a) shows that top 20 largest server clusters contain more that 99% of the total servers. Only a weak correlation exists between the number of servers in a server cluster and the number of requests received in a server cluster, as observed in Figure 5.8(c).

Larmancom proxy logs

The experimental results of the Larmancom proxy logs are shown in Table 5.16. The number of server clusters varies from 10,722 to 17,466, where the size of the largest server cluster varies from 750 servers to 2,559 servers. The number of requests received by the busiest server clusters varies from 1,617,901 to 15,199,709.

Figures 5.9 and 5.10 show the distribution of server clusters. The observations on the results of the AT&T proxy logs also hold on the Larmancom proxy logs.

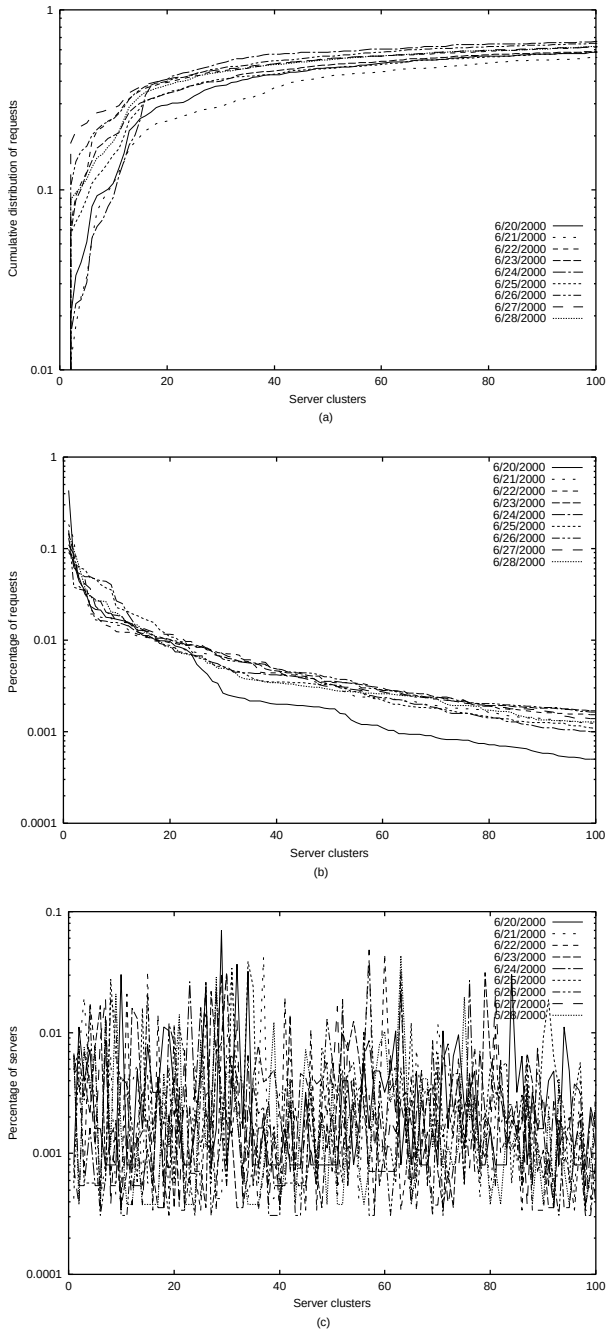


Figure 5.7: The server cluster distribution of the AT&T proxy logs plotted in the reverse order of the number of requests received by a server cluster (y axis is in log scale): (a) cumulative distribution of the number of requests in a server cluster; (b) distribution of the number of requests in a server cluster; and (c) distribution of the number of servers in a server cluster.

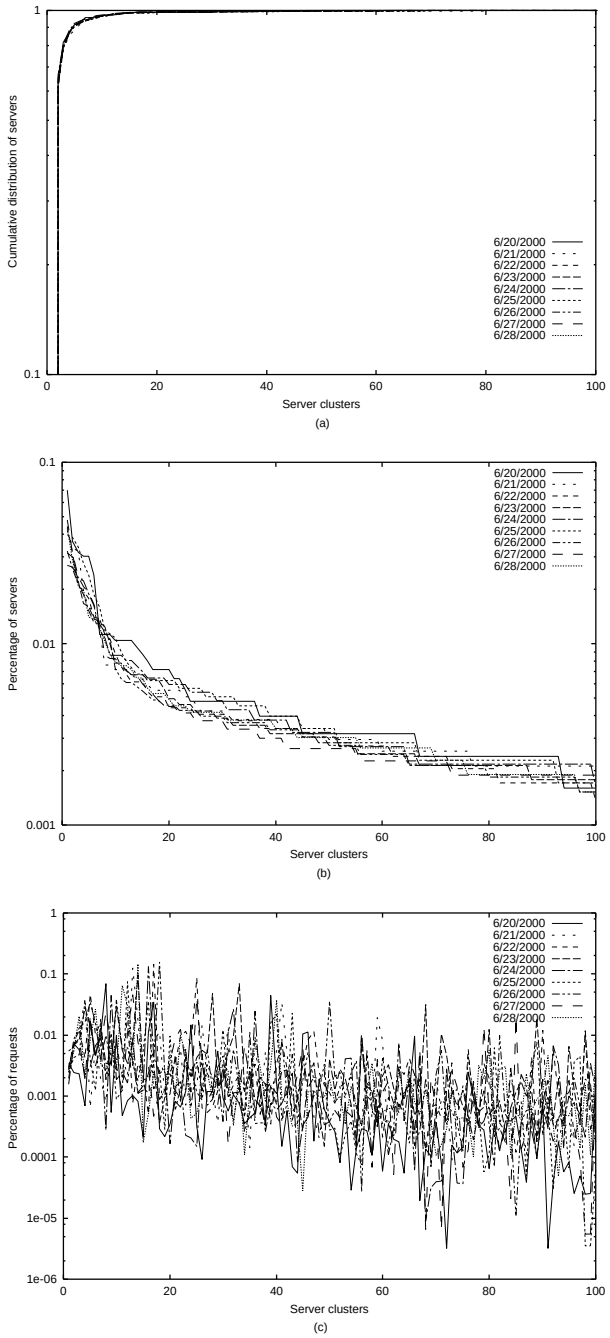


Figure 5.8: The server cluster distribution of the AT&T proxy logs plotted in the reverse order of the number of servers in a server cluster (y axis is in log scale): (a) cumulative distribution of the number of servers in a server cluster; (b) distribution of the number of servers in a server cluster; (c) distribution of the number of requests in a server cluster.

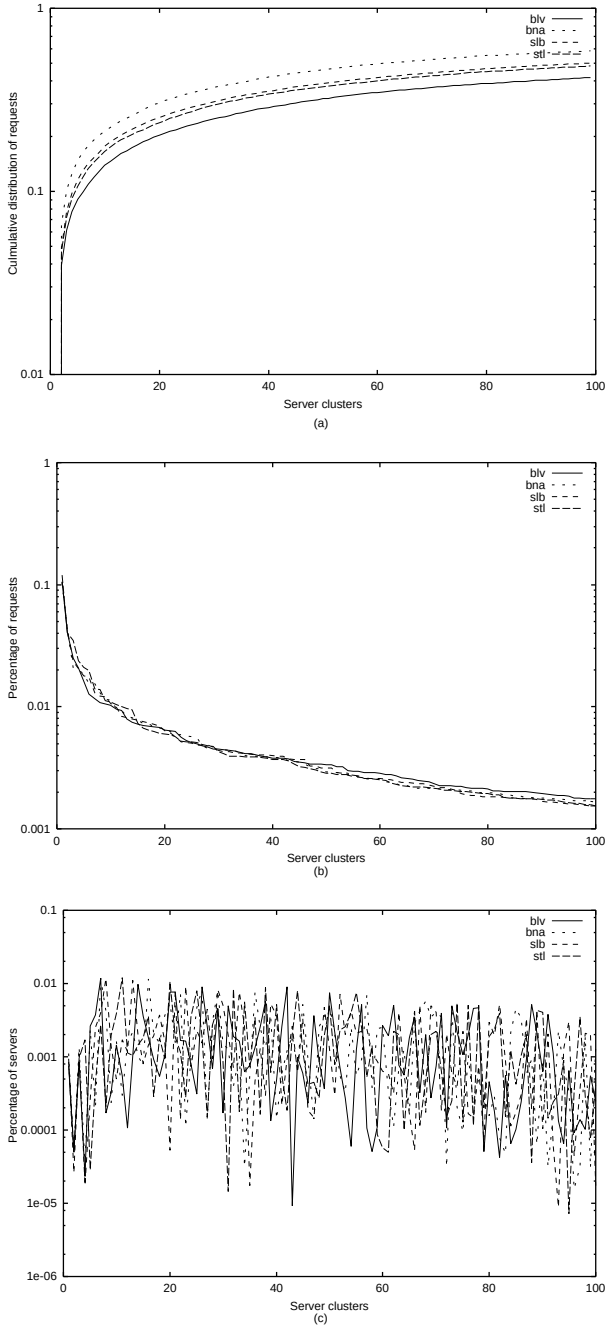


Figure 5.9: The server cluster distribution of the Larmancom proxy logs plotted in the reverse order of the number of requests received by a server cluster (y axis is in log scale): (a) cumulative distribution of the number of requests in a server cluster; (b) distribution of the number of requests in a server cluster; (c) distribution of the number of servers in a server cluster.

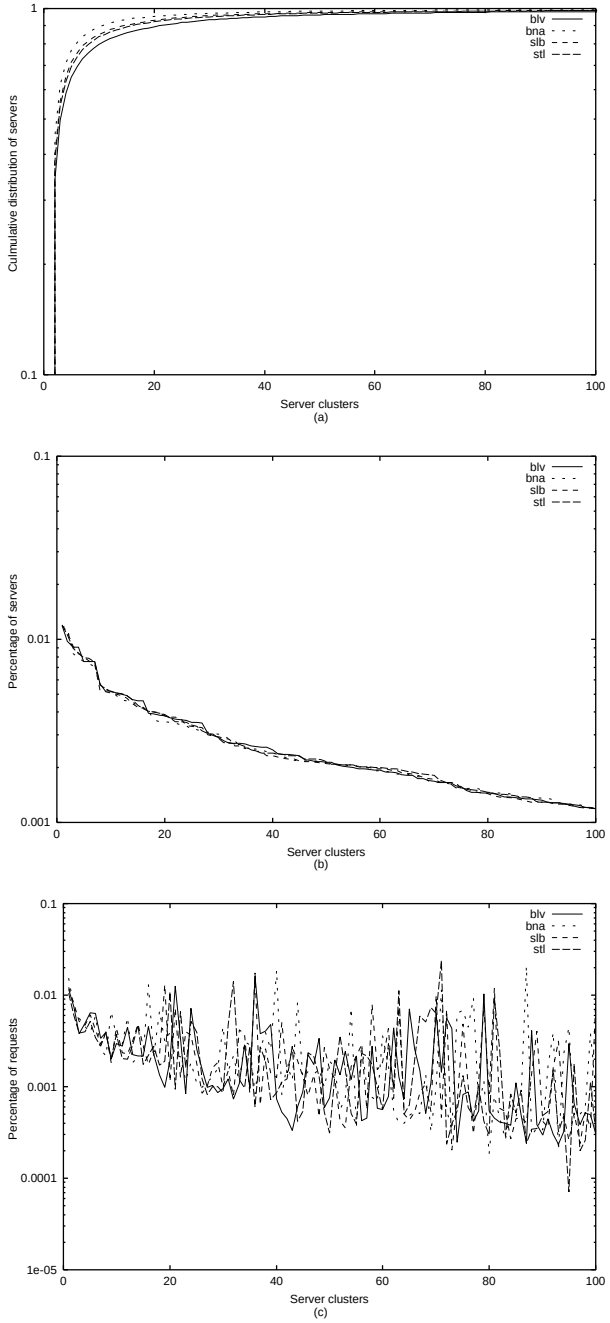


Figure 5.10: The server cluster distribution of the Larmancom proxy logs plotted in the reverse order of the number of servers in a server cluster (y axis is in log scale): (a) cumulative distribution of the number of servers in a server cluster; (b) distribution of the number of servers in a server cluster; (c) distribution of the number of requests in a server cluster.

5.4.2 Validation

Similar to client cluster validation in Chapter 3, we use both *nslookup*-based and *traceroute*-based tests to validate the results of server cluster detection, although the purpose is different here. In Chapter 3, the validation tests are based on the observation that clients in the same cluster often share a non-trivial suffix in their fully-qualified domain names. This is not true for servers since servers in the same cluster can have different names (e.g., due to the service provider). Our validation test on server cluster detection is based on the combination of the name of the server and path towards the server. We first perform the *nslookup*-based test on the server cluster. If the server cluster passed the *nslookup*-based test, then we say the server cluster is detected correctly. Failing to pass the *nslookup*-based validation test does not necessarily mean that the server cluster is mis-detected. We then perform the *traceroute*-based test on the server clusters that failed to pass the *nslookup*-based test. If a server cluster passes the *traceroute*-based test, we also say the server cluster is detected correctly. Otherwise, we say the server cluster is mis-detected.

We take 10% samples from the server clusters detected in each of the AT&T proxy logs. The validation results are shown in Table 5.17. The number of sampled server clusters in a proxy log varies from 44 to 120 and the number of servers in the clusters from 77 to 263. After performing the validation tests, the percentage of sampled server clusters in a proxy log that failed in the *nslookup*-based test varies from 4% to 17% and the percentage of sampled server clusters in a proxy log that failed in the *traceroute*-based test varies from 0% to 10%. For example, we sampled 72 server clusters in the proxy log taken on 6/24/2000. The total number of servers in sampled server clusters is 127. There are 9 server clusters that failed

Table 5.17: Server cluster validation on the AT&T proxy logs.

Date	Total number of sampled server clusters	Number of sampled servers	Number of server clusters that failed the *nslookup* test	Number of server clusters that failed the *traceroute* test
6/20/2000	44	77	2	0
6/21/2000	79	111	10	7
6/22/2000	106	209	19	10
6/23/2000	106	257	15	11
6/24/2000	72	127	9	5
6/25/2000	60	153	7	4
6/26/2000	120	263	18	10
6/27/2000	100	192	13	10
6/28/2000	100	192	13	8

the *nslookup*-based test, among which 5 server clusters (4% of the sampled server clusters) failed the *traceroute*-based test (i.e., 5 server clusters are mis-detected). Thus, the accuracy of server cluster detection algorithm is above 90%.

We also conducted the validation test on the Larmancom proxy logs. We sampled 1%[2] of server clusters in each proxy log. The validation results are shown in Table 5.18. For example, we sampled 190 server clusters in the Blv proxy log. The total number of servers in the sampled server clusters is 1374. There are 63 server clusters that failed the *nslookup*-based test, among which 17 server clusters failed the *traceroute*-based test. Thus, the accuracy of the server cluster detection algorithm is again above 90%.

5.4.3 Summary

Experimental results show that more that 99% of the servers in the proxy logs we examined can be grouped into server clusters. Around 90% of the server clusters are

[2]Instead of sampling 10% of server clusters, we sampled 1% of the server clusters in the Larmancom proxy logs due to the large size of the Larmancom proxy logs.

Table 5.18: Server cluster validation on the Larmancom proxy logs.

Proxy log	Total number of sampled server clusters	Number of sampled servers	Number of server clusters that failed the *nslookup* test	Number of server clusters that failed the *traceroute* test
Blv	190	1,374	63	17
Bna	117	738	38	11
Slb	152	1,437	53	16
Stl	167	2,115	57	15

detected correctly (i.e., passed our validation tests). These are consistent with the observations of the applicability and accuracy of client cluster detection in Chapter 3. Detecting client and server clusters can aid in engineering and shaping traffic in a content distribution application.

5.5 Discussion and Other issues on client clustering

In this chapter, we improve our basic clustering approach in four distinct ways. We first examine if busy clusters can be combined into a super-cluster based on the AS that originates the cluster's IP address prefix. Experiments show that prefix-level clustering is not too fine grained and additional information on the originating AS will not help. Second, we examine if non-busy client clusters can be grouped into busy super-clusters. We group all the client clusters into super-clusters. After identifying busy super-clusters, we show that small (non-busy) clusters cannot be combined into a busy super-cluster if their IP address prefixes originate from the same AS. Third, we show that sub-clustering is helpful to identify proxies and spiders and to select good servers for busy client clusters. This indicates that large

clusters can be partitioned into sub-clusters based on the client access patterns. Finally, we examine server clustering and show that server clustering is also possible and interesting. Experimental results show that more that 99% of the servers in the proxy logs we examined can be grouped into server clusters. Around 90% of the server clusters are detected correctly (i.e., passed the validation tests). These are consistent with the observations of the applicability and accuracy of client cluster detection.

Besides the improvements described above, we examine how requests issued and unique URLs accessed from within each client cluster vary during different time periods. We partition the Nagano server log into four 6-hour sessions [KW00, KW00TM]. Although the first two sessions are less busy than the last two ones, all of them show similar patterns in terms of both the number of requests issued and number of unique URLs accessed by each client cluster. The observations on client cluster distributions obtained from the entire server log still hold for each session indicating that simulations on a sample of server logs might suffice. Time partitioning result details are reported in [KW00].

Chapter 6

Conclusion

A natural partition of IP addresses exists on the Internet. This thesis has proposed a novel way to identify IP address clusters using BGP routing information. The experiments are conducted on a wide range of Web logs which are a good source of IP addresses. The results show that our method is able to group more than 99.9% of IP addresses captured in a wide variety of Web logs into clusters. In order to provide a quantitative validation of the resulting clusters, we proposed two tests based on the network tools *nslookup* and *traceroute*. Validation results demonstrate that our method passes validation tests in over 90% of the sampled cases. We also presented a self-correction and adaptation mechanism that improves the applicability and accuracy of the initial cluster identification results. The entire cluster identification process can be done in an automated fashion. Our methodology is insensitive to BGP dynamics and independent of the range and diversity of server logs.

Network-aware IP address clustering provides a tool which allows a natural aggregation of IP addresses, where the location of the IP addresses has been considered. It helps researchers to obtain a solid understanding of the Internet topology

and traffic load patterns. The cluster information is useful in many applications such as content distribution, Web caching, server replication, traffic engineering and load balancing, network management, and Internet map discovery. Two typical applications – Web caching and server replication – are discussed in this thesis as examples to illustrate the use of client clustering. In the Web caching application, we examine the usefulness of network aware clustering in a Web caching simulation and contrast it with the typical 24-bit heuristic. The significant differences shown in simulation results demonstrate that the simple approach is not able to evaluate benefits of Web caching schemes or the corresponding overhead (both at the server and at proxies) well, and hence, fails to serve as a guide for solving problems such as proxy placement. The simulation results from our network-aware approach are more realistic, and is useful for designing and evaluating Web caching systems. For example, knowing the location of clients and their demands, a Web site can better provision its service. While we only address simulation of the Web caching system with one server and multiple proxies, we can also simulate multiple servers and multiple proxies by merging more server logs collected at the same time. In our discussion of the Web caching application, we also presented a method to identify spiders and proxies among Web clients by examining the access patterns of corresponding client clusters.

In the server replication application, we proposed a scheme that can be deployed at Web servers to automatically re-direct client requests reducing the access latency. We selected good servers for a client cluster based on a "follow-the-herd" heuristic. We also proposed a mechanism to reconstruct the busy cluster list dynamically. We conducted experiments on a set of server logs of a digital library site. Although, due to lack of direct access to the digital library servers, we can-

not validate if our selection of the good servers will provide a lower user-perceived latency than other non-good servers, our *traceroute* results show that our selection of good servers usually have a smaller RTT to the client clusters than that of other servers.

Besides the basic clustering scheme and applications, for the sake of completeness, this thesis also discussed several improvements to the basic clustering scheme. First, in the validation tests proposed in Chapter 3, we only examined if a cluster is too big. We used the prefix originating AS information to group busy client clusters into super-clusters and show that prefix-level clustering is not too fine grained, and additional information on the originating AS will not help.

Second, we assembled interesting client clusters based on a thresholding scheme that includes only clusters that contribute 70% of the total number of requests. We relied on the fact that the *busy* client clusters are distinct but it might still have been possible for several *non-busy* client clusters to be grouped together and end up in the interesting set. In our experiments, we grouped all the client clusters into super-clusters. After identifying busy super-clusters, we found that a number of non-busy clusters cannot be combined into a busy super-cluster if their IP address prefixes originate from the same AS.

Third, our client clusters are constructed based on prefixes extracted from BGP routing tables. However, some applications may be interested in an even more fine-grained partition of clients. For instance, one may want to partition a large client cluster into sub-clusters based on the clients' access patterns. We showed that sub-clustering is helpful to identify proxies and spiders and to select good servers for busy client clusters. This indicates that large clusters can be partitioned into sub-clusters based on the client access patterns.

Finally, symmetric to client clustering, we examined server clustering and showed that server clustering is also possible and interesting. Experimental results show that more that 99% of the servers in the proxy logs we examined can be grouped into server clusters. Around 90% of the server clusters are detected correctly (i.e., passed the validation tests). These are consistent with the observations of the applicability and accuracy of client cluster detection. Detecting client and server clusters can aid in engineering and shaping traffic in a content distribution application.

To summarize, this thesis proposed a network-aware approach to clustering IP addresses and discussed its applications and potential improvements. Ongoing work includes fast prefix matching, cluster-based access pattern analysis, better validation tools, and exploring a wider range of applications of clustering.

Appendix A

Experimental results of client clustering on other Web logs

In order to examine the generality of our approach to client clustering, we conducted our experiments on a wide range of Web server logs which include Apache, EW3, Sun, and Unav server logs.

A.1 Apache server log

The Apache log is a 49-day Web server log taken from October 1, 1997 to November 18, 1997. A summary of the client cluster detection results is provided in Table A.1. There are a total of 3,461,364 requests in the server log issued by 274,844 clients accessing 51,536 unique URLs at the server. We detect all these clients as 35,563 client clusters. The size of client clusters varies from 1 to 1,680 clients, the number of requests issued from within each client cluster varies from 1 to 78,336, and clusters access anywhere from 1 to 21,207 unique URLs.

We plot the client cluster distributions in Figures A.1 - A.4. Figure A.1(a)

Table A.1: Experimental results of client cluster detection on the Apache server log.

Total number of requests	3,461,361
Total number of unique URLs	51,536
Total number of unique clients	274,844
Number of undetected clients	0
Total number of client clusters	35,563
Largest client cluster	1,680 clients (17,529 requests)
Smallest client cluster	1 client (1 request)
Busiest client cluster	78,336 requests
Least busy client cluster	1 request
Client cluster URL access	1 URL (0.000019% of total) - 21207 URLs (0.41% of total)

shows the cumulative distribution of the number of clients in client clusters. We observe that, although the largest client cluster contains 1,690 clients, more than 95% of the client clusters contain fewer than 100 clients. The cumulative distribution of requests issued in client clusters is shown in Figure A.1(b), which is more heavy-tailed than that of the number of clients in client clusters as shown in Figure A.1(a). Around 95% of the client clusters issued fewer than 1,000 requests. Only a few client clusters are very *busy*, issuing up to a maximum of 78,336 requests. Figures A.2(a) and A.3(a) show the distributions of the number of clients in client clusters (in the reverse order of the number of clients) and the number of requests issued from within client clusters (in reverse order of the number of requests issued), respectively, again demonstrating that the distribution of the number of requests issued from within client clusters in more heavy-tailed than the number of clients in client clusters.

We also plot the distributions of the number of requests issued from within client clusters and the number of unique URLs accessed from within client clusters (in the reverse order of the number of clients) in Figure A.2(b) and (c), respectively.

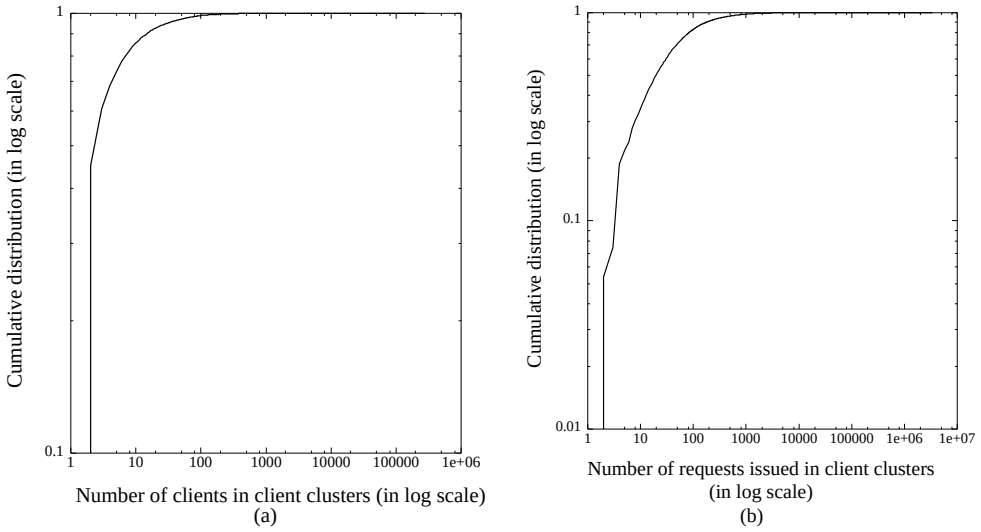


Figure A.1: The cumulative distribution of clients and requests in a client cluster for the Apache server log (both x axis and y axis are in log scale): (a) is the cumulative distribution of the number of clients in client clusters; (b) is the cumulative distribution of the number of requests issued from within client clusters.

Again larger client clusters issue more requests and access more URLs than smaller client clusters. Also, there are a number of relatively small client clusters which issue a significant amount of requests ($\sim$ 2% of the total) and/or access a big fraction of URLs ($\sim$ 40% of the total) at the server. Again, we plot the distributions of the number of clients in client clusters and the number of URLs accessed from within client clusters (in the reverse order of the number of requests issued) in Figures A.3(b) and (c), respectively.

A.2 EW3 server logs

We study a number of EW3 server logs, four one-month logs of July 1999 (EW3-a1, EW3-b, EW3-c1, and EW3-c2) and two six-month logs from July 1999 to December 1999 (EW3-a2 and EW3-w).

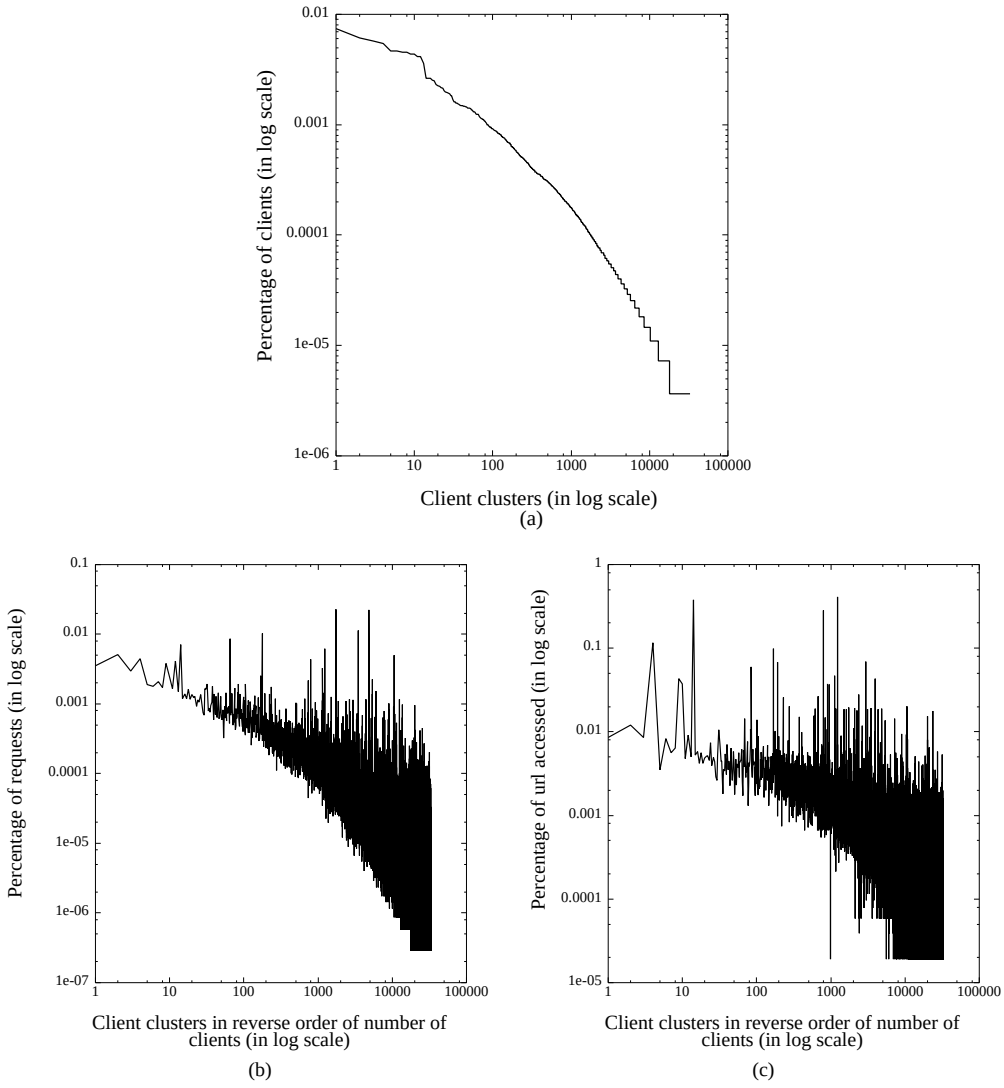


Figure A.2: The client cluster distribution for the Apache server log plotted in the reverse order of the number of clients in a cluster (both x axis and y axis are in log scale): (a) is the distributions of the number of clients in client clusters; (b) is the number of requests issued from within client clusters; (c) is the number of URLs accessed from within client clusters.

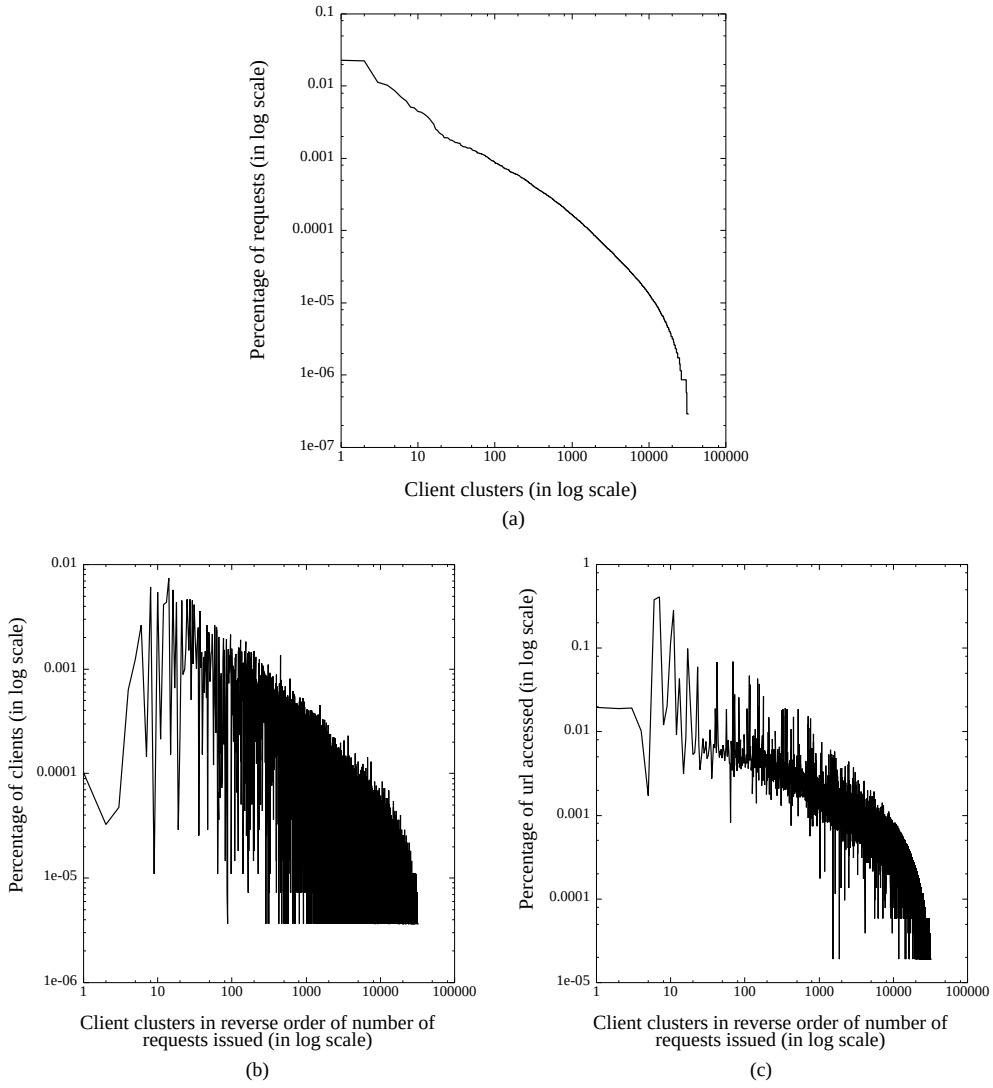


Figure A.3: The client cluster distribution for the Apache server log plotted in the reverse order of the number of requests (both x axis and y axis are in log scale): (a) is the distributions of the number of requests issued from within client clusters (re-plot of Figure A.2(b)); (b) is the number of clients in client clusters (re-plot of Figure A.2(a)); (c) is the number of URLs accessed from within client clusters (re-plot of Figure A.2(c)).

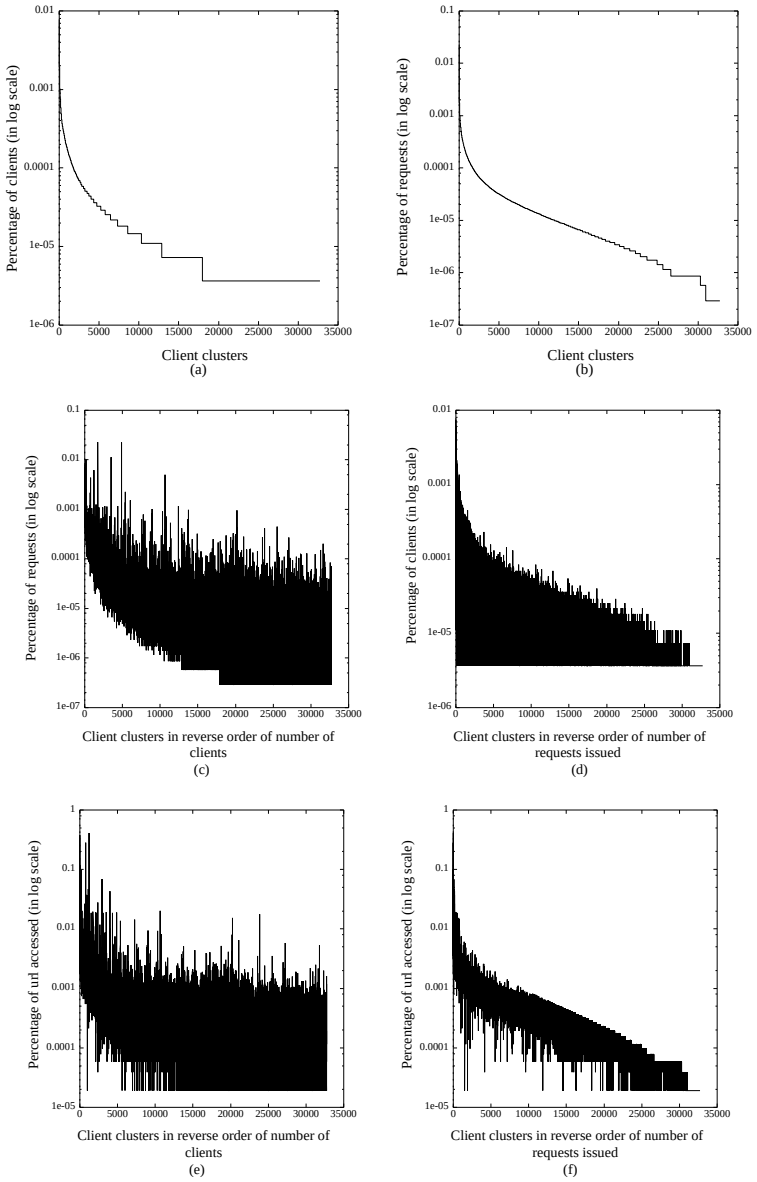


Figure A.4: The client cluster distribution for the Apache server log (y axis is in log scale): (a) - (f) are the re-plotted figures of Figures A.2 (a) - (c) and A.3 (a) - (c), respectively, to clarify how client cluster distributions vary along the number of clients in client clusters and the number of requests issued from within client clusters.

- **EW3-a1.** The EW3-a1 log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 1,199,276 requests in the server log, issued by 21,519 clients accessing 683 unique URLs at the server. We detect all of these clients as 7,754 client clusters. The size of client clusters varies from 1 to 359 clients, while the number of requests issued from within each client cluster varies from 1 to 21,949 (Table A.2).

- **EW3-a2.** The EW3-a2 log is a 6-month Web server log taken from June 26, 1999 to December 20, 1999. We divide it into 6 sub-logs, one for each month. The detailed information is provided in Table A.3.

- **EW3-b.** The EW3-b log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 1,908,761 requests in the server log, issued by 40,156 clients accessing 7,772 unique URLs at the server. We detect all of these clients as 14,895 client clusters. The size of client clusters varies from 1 to 704 clients, while the number of requests issued from within each client cluster varies from 1 to 76,218 (Table A.4).

- **EW3-c1.** The EW3-c1 log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 1,496,408 requests in the server log, issued by 160,020 clients accessing 618 unique URLs at the server. We detect all of these clients as 18,571 client clusters. The size of client clusters varies from 1 to 6,989 clients, while the number of requests issued from within each client cluster varies from 1 to 53,217 (Table A.5).

- **EW3-c2.** The EW3-c2 log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 148,859 requests in the server log, issued by 41,456 clients accessing 769 unique URLs at the server. We detect

Table A.2: Experimental results of client cluster detection on the EW3-a1 server log.

Total number of requests	1,199,276
Total number of unique URLs	683
Total number of unique clients	21,519
Number of undetected clients	0
Total number of client clusters	7,754
Largest client cluster	359 clients (19,409 requests)
Smallest client cluster	1 client (1 request)
Busiest client cluster	21,949 requests
Least busy client cluster	1 request

all of these clients as 16,286 client clusters. The size of client clusters varies from 1 to 824 clients, while the number of requests issued from within each client cluster varies from 1 to 2,410 (Table A.6).

- **EW3-w.** The EW3-w log is a 6-month Web server log taken from June 26, 1999 to December 20, 1999. We divide it into 6 sub-logs, one for each month. The detailed information is provided in Table A.7.

Our experimental results show that all clients in the EW3 server logs are able to be detected as client clusters using our method.

A.3 Sun server log

The Sun log is a 9-day Web server log taken from September 30, 1997 to October 9, 1997. A summary of the client cluster detection result is provided in Table A.8. There are a total of 13,871,352 requests in the server log, issued by 219,528 clients and access 116,274 unique URLs at the server. We detect all of these clients as 33,468 client clusters. The size of client clusters varies from 1 to 2,529 clients, the

Table A.3: Experimental results of client cluster detection on the EW3-a2 server log over 6 months.

Date (1999)	Jun 27 - Jul 26	Jul 27 - Aug 26	Aug 27 - Sep 26	Oct 1 - Oct 31	Nov 6 - Nov 30	Dec 1 - Dec 20
Total # of requests	1,357,623	1,645,975	2,180,029	2,877,528	2,223,590	1,847,913
Total # of URLs	11,475	10,462	8,569	10,005	10,111	10,556
Total # of clients	33,675	35,249	44,831	54,882	46,579	38,913
# of undetected clients	0	0	0	38	82	110
Total # of client clusters	12,465	12,880	14,949	17,826	15,206	13,620
Largest client cluster						
(clients)	634	932	760	1001	859	717
(requests)	16,513	26,979	25,648	42,989	27,635	24,784
Smallest client cluster						
(clients)	1	1	1	1	1	1
(requests)	1	1	1	1	1	1
Busiest client cluster (requests)	100,140	145,970	116,806	132,009	81,694	86,585
Least busy client cluster (request)	1	1	1	1	1	1

Table A.4: Experimental results of client cluster detection on the EW3-b server log.

Total number of requests	1,908,761
Total number of unique URLs	7,772
Total number of unique clients	40,156
Number of undetected clients	0
Total number of client clusters	14,895
Largest client cluster	704 clients (25,328 requests)
Smallest client cluster	1 client (1 request)
Busiest client cluster	76,218 requests
Least busy client cluster	1 request

Table A.5: Experimental results of client cluster detection on the EW3-c1 server log.

Total number of requests	1,496,408
Total number of unique URLs	618
Total number of unique clients	160,020
Number of undetected clients	0
Total number of client clusters	18,571
Largest client cluster	6,989 clients (53,217 requests)
Smallest client cluster	1 client (1 request)
Busiest client cluster	53,217 requests
Least busy client cluster	1 request

Table A.6: Experimental results of client cluster detection on the EW3-c2 server log.

Total number of requests	148,859
Total number of unique URLs	769
Total number of unique clients	41,456
Number of undetected clients	0
Total number of client clusters	16,286
Largest client cluster	824 clients (2,410 requests)
Smallest client cluster	1 client (1 request)
Busiest client cluster	2,410 requests
Least busy client cluster	1 request

Table A.7: Experimental results of client cluster detection on the EW3-w server log over 6 months.

Date (1999)	Jun 27 - Jul 26	Jul 27 - Aug 26	Aug 27 - Sep 15	Oct 1 - Oct 31	Nov 6 - Nov 30	Dec 1 - Dec 20
Total # of requests	575,451	3,753,571	2,405,524	2,297,309	4,506,717	2,573,251
Total # of URLs	1,705	19,210	8,569	14,982	2,069	1,526
Total # of clients	3,659	16,202	10,587	20,788	20,165	13,452
# of undetected clients	0	0	0	10	27	36
Total # of client clusters	1,517	6,187	4,391	7,321	6,863	5,111
Largest client cluster						
(clients)	389	523	520	574	593	573
(requests)	39,021	145,010	103,531	45,810	114,206	82,788
Smallest client cluster						
(clients)	1	1	1	1	1	1
(requests)	1	1	1	1	1	1
Busiest client cluster						
(requests)	39,021	151,229	104,183	107,791	167,807	99,176
Least busy client cluster						
(request)	1	1	1	1	1	1

Table A.8: Experimental results of client cluster detection on the Sun server log.

Total number of requests	13,871,352
Total number of unique URLs	116,274
Total number of unique clients	219,528
Number of undetected clients	0
Total number of client clusters	33,468
Largest client cluster	2,529 clients (119,231 requests)
Smallest client cluster	1 client (1 request)
Busiest client cluster	693,955 requests
Least busy client cluster	1 request
Client cluster URL access	1 URL (0.000009% of total) - 33594 URLs (0.29% of total)

number of requests issued from within each client cluster varies from 1 to 693,955, and clusters access anywhere from 1 to 33,594 unique URLs.

We plot the client cluster distributions in Figures A.5 - A.8. Figure A.5(a) shows the cumulative distribution of the number of clients in client clusters. We observe that, although the largest client cluster contains 2,536 clients, more than 95% of the client clusters contain fewer than 100 clients. The cumulative distribution of requests issued in client clusters is shown in Figure A.5(b), which is more heavy-tailed than that of the number of clients in client clusters as shown in Figure A.5(a). Around 90% of the client clusters issued fewer than 1,000 requests. Only few client clusters are very *busy*, issuing up to a maximum of 693,955 requests. Figures A.6(a) and A.7(a) show the distributions of the number of clients in client clusters (in the reverse order of the number of clients) and the number of requests issued from within client clusters (in reverse order of the number of requests issued), respectively. Again the distribution of the number of requests issued from within client clusters is more heavy-tailed than that of the number of clients in client clusters.

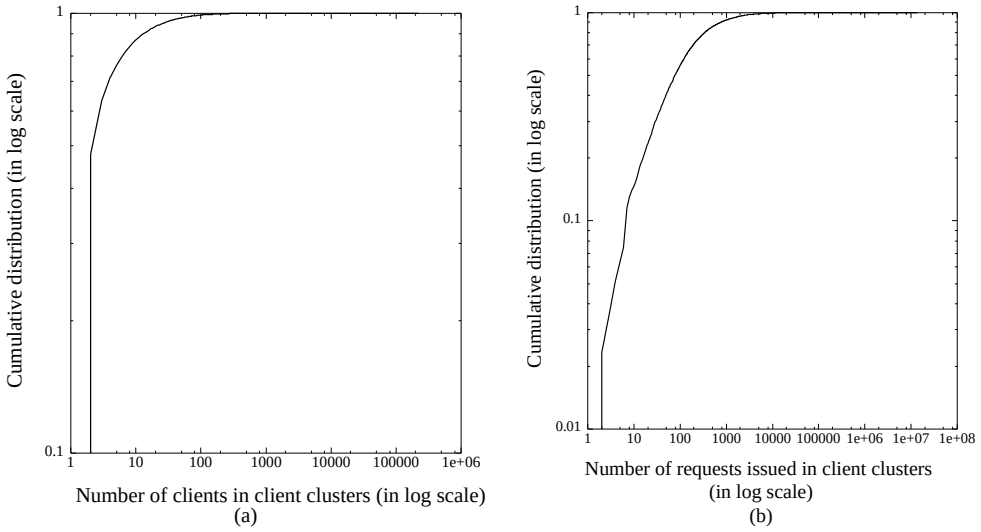


Figure A.5: The cumulative distribution of clients and requests in a client cluster for the Sun server log (both x axis and y axis are in log scale): (a) is the cumulative distribution of the number of clients in client clusters; (b) is the cumulative distribution of the number of requests issued from within client clusters.

We plot the distributions of the number requests issued from within client clusters and the number of unique URLs accessed from within client clusters (in the reverse order of the number of clients) in Figures A.6(b) and (c), respectively. Larger client clusters issue more requests and access more URLs than smaller client cluster. However, there are a number of relatively small client clusters which issue significant amount of requests ($\sim$ 5% of the total) and/or access a big portion of URLs ($\sim$ 30% of the total) at the server. Finally, we plot the distributions of the number of clients in client clusters and the number of URLs accessed from within client clusters (in the reverse order of the number of requests issued) in Figures A.7(b) and (c), respectively.

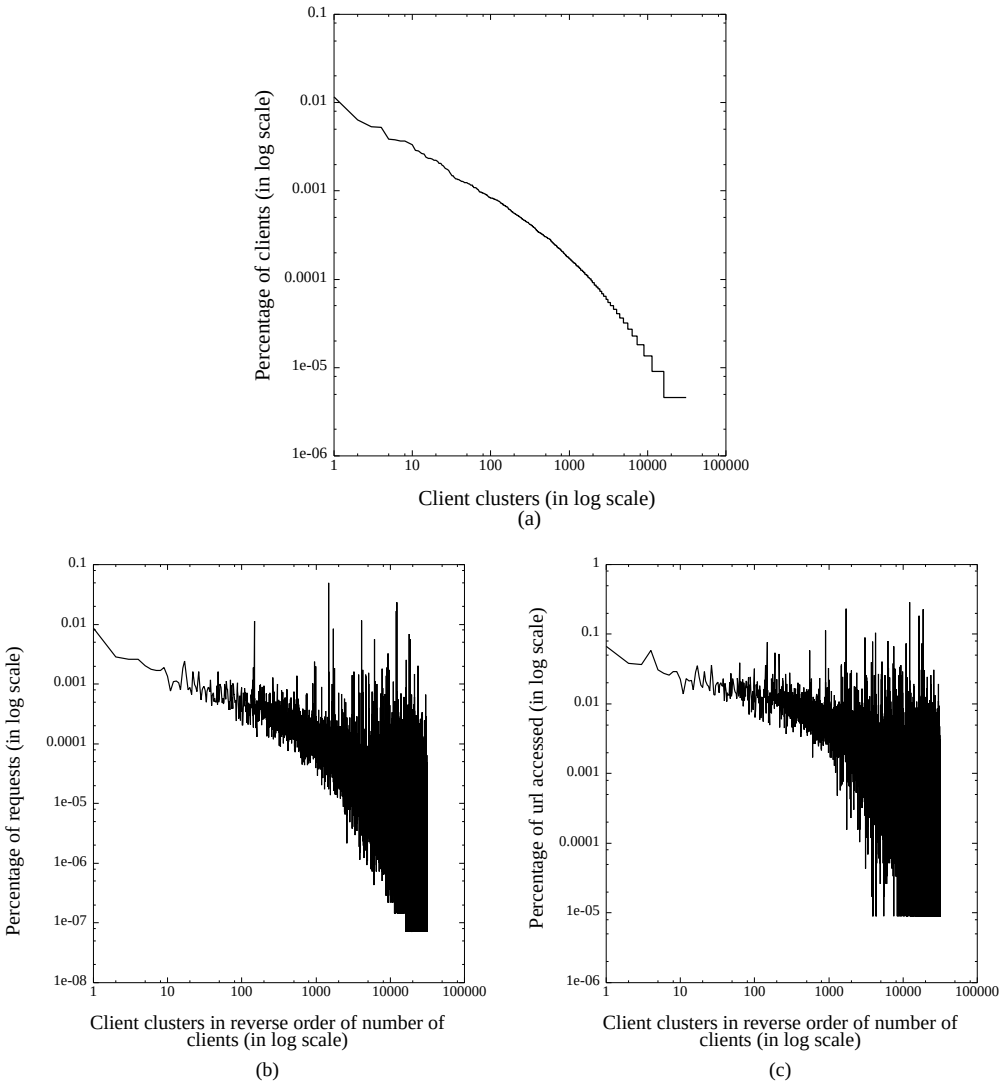


Figure A.6: The client cluster distribution for the Sun server log plotted in the reverse order of the number of clients in a cluster (both x axis and y axis are in log scale): (a) is the distributions of the number of clients in client clusters; (b) is the number of requests issued from within client clusters; (c) is the number of URLs accessed from within client clusters.

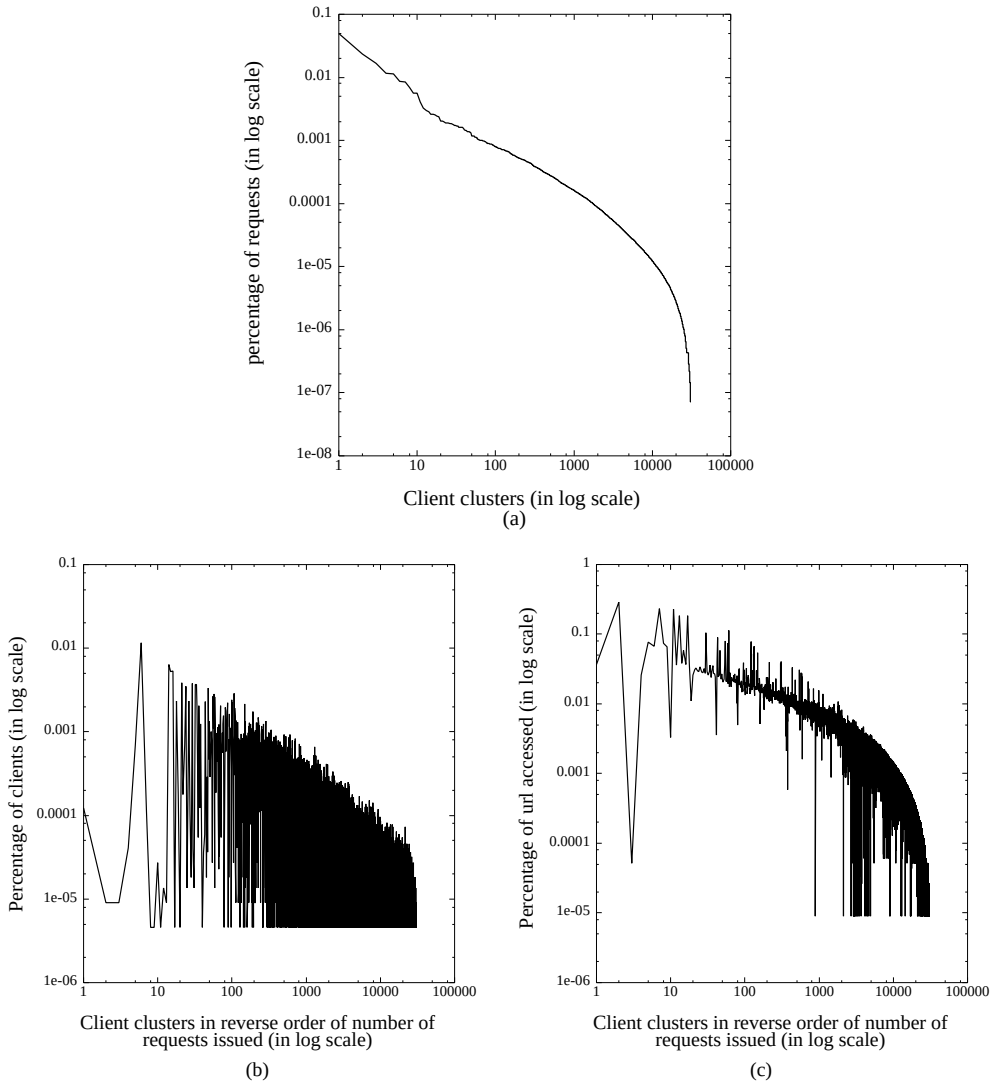


Figure A.7: The client cluster distribution for the Sun server log plotted in the reverse order of the number of requests (both x axis and y axis are in log scale): (a) is the distributions of the number of requests issued from within client clusters (re-plot of Figure A.6(b)); (b) is the number of clients in client clusters (re-plot of Figure A.6(a)); (c) is the number of URLs accessed from within client clusters (re-plot of Figure A.6(c)).

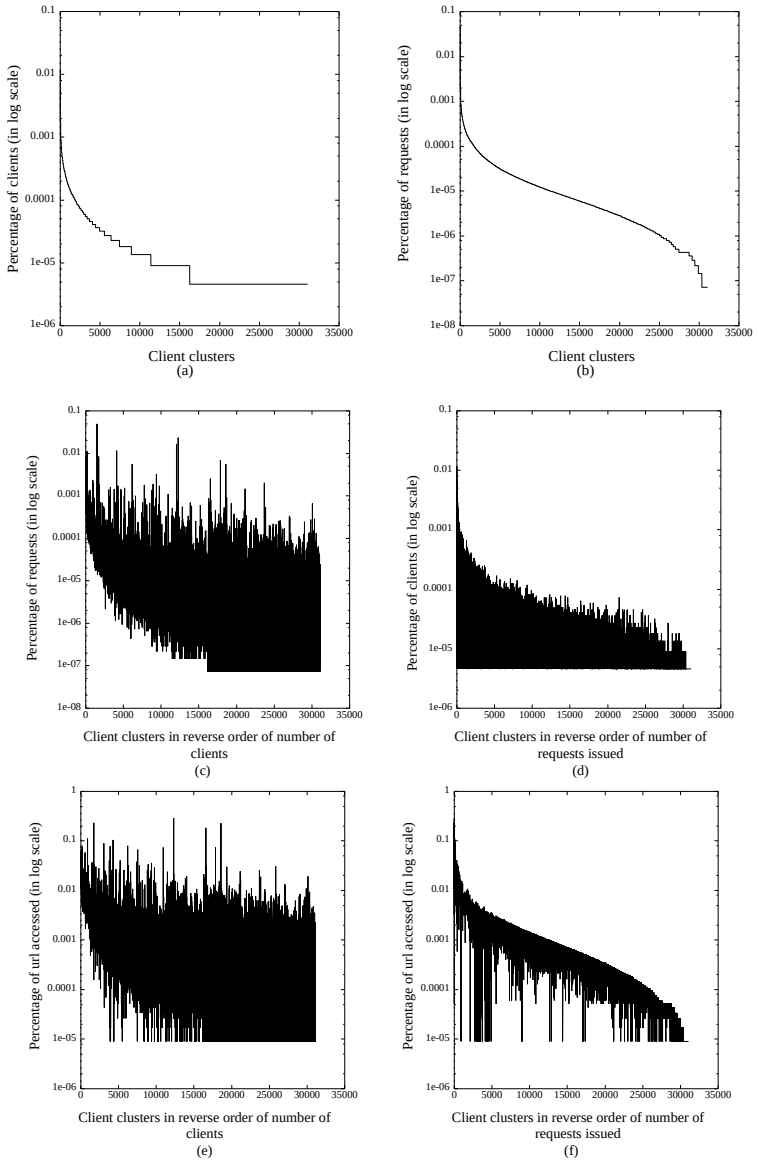


Figure A.8: The client cluster distribution for the Sun server log (y axis is in log scale): (a) - (f) are the re-plotted figures of Figures A.6 (a) - (c) and A.7 (a) - (c), respectively, to clarify how client cluster distributions vary along the number of clients in client clusters and the number of requests issued from within client clusters.

Table A.9: Experimental results of client cluster detection on the Unav server log.

Total number of requests	6,593,672
Total number of unique URLs	24,150
Total number of unique clients	28,672
Number of undetected clients	21
Total number of client clusters	2,480
Largest client cluster	7,918 clients (966,788 requests)
Smallest client cluster	1 client (1 request)
Busiest client cluster	1,507,769 requests
Least busy client cluster	1 request

A.4 Unav server log

The Unav is an event log that was obtained over a thirty-two hour period during December 1999. A summary of the client cluster detection result is provided in Table A.9. There are a total of 6,593,672 requests in the server log issued by 28,672 clients accessing 24,150 unique URLs at the server. We detect all these clients as 2,480 client clusters. The size of client clusters varies from 1 to 7,918 clients and the number of requests issued from within each client cluster varies from 1 to 1,507,769.

A.5 Summary

The experimental results on a wide range of Web logs show that we can group more than 99.9% of the clients into clusters, with very few clients not clusterable (i.e., no network prefixes in our prefix table matches the client IP addresses) due to the lack of proper prefix/netmask information in the routing table snapshots. This is fixed in the self-correction and adaptation stage of our approach (discussed in Chapter 3.8).

Bibliography

[ABCdO96] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. Technical Report TR-96-11, Boston University Computer Science Department, November 1996.

[ACF+98] V. Almeida, M. G. Cesario, R. C. Fonseca, W. Meira Jr., and C. D. Murta. The influence of geographical and cultural issues on the cache proxy server workload. In *Proceedings of the Seventh International World Wide Web Conference*, April 1998.

[ade] Adero, `http://www.adero.com`.

[AFAW] G. Abdulla, E. A. Fox, M. Abrams, and S. Williams. WWW proxy traffic characterization with application to caching. `http://csgrad.cs.vt.edu/~abdulla/proxy/proxy-char.ps`.

[AFJ99] M. Arlitt, R. Friedrich, and T. Jin. Workload characterization of a Web proxy in a cable modem environment. *ACM Performance Evaluation Review*, 27(2):25–36, August 1999. Also available as HPL Technical Report HPL-1999-48 `http://www.hpl.hp.com/techreports/1999/HPL-1999-48.html`.

[aka] Akamai, `http://www.akamai.com`.

[ari99] American Registry for Internet Numbers IP network dump, October 1999. `ftp://rs.arin.net/netinfo`.

[ASA+95] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching proxies: limitations and potentials. In *Proceedings of the 4th International WWW Conference*, December 1995.

[att99] AT&T Routing and Forwarding Table Snapshots, April 1999. Obtained from AT&T.

[AWY99] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the World Wide Web. In *IEEE Transactions on Knowledge and Data Engineering*, January/February 1999.

[KW00TM] B. Krishnamurthy and J. Wang. On network-aware clustering of Web clients. Technical Report Technical Report # HA1630000-000101-01-TM, AT&T Labs—Research, January 2000. `www.research.att.com/~bala/papers/cluster-tm.ps.gz`.

[BB] K. Bharat and A. Broder. Measuring the Web. `www.research.digital.com/SRC/whatsnew/sem.html`.

[BBBC99] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella. Changes in Web client access patterns: characteristics and caching implications. In *World Wide Web (special issue on Characterization and Performance Evaluation)*, 1999.

[BC96] A. Bestavros and C. Cunha. Server-initiated document dissemination for the WWW. In *IEEE Data Engineering Bulletin*, September 1996.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like distributions: Evidence and Implications. In *Proceedings of IEEE Infocom'99*, March 1999. `www.research.att.com/~breslau/pubs/zipf.ps.gz`.

[BH96] J. C. Bolot and P. Hoschka. Performance engineering of the World Wide Web: Application to dimensioning and cache design. In *Proceedings of the 5th International WWW Conference*, May 1996.

[Blo70] B. Bloom. Space/time trade-offs in hash coding with allowable errors. In *Communications of the ACM*, July 1970.

[can99] Canada Internet Transit Service, December 1999. `enfm.utcc.utoronto.ca/cgi-bin/c2/c2routes.pl?pop=toronto`.

[Cat92] V. Cate. Alex - a global file system. In *Proceedings of the 1992 USENIX File System Workshop*, May 1992.

[CB98] M. Crovella and P. Batford. The network effects of prefetching. In *Proceedings of Infocom*, 1998.

[CDF+98] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web proxy caching: the devil is in the details. In *ACM Performance Evaluation Review*, December 1998.

[CDN+96] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel. A hierarchical Internet object cache. In *Usenix*, January 1996.

[cer99] AT&T Cerfnet BGP Route Viewer, September 1999. Host: `route-server.cerf.net`.

[CID99] J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic Web data. In *Proceedings of Infocom*, 1999.

[CJ97] C. R. Cunha and C. F. B. Jaccoud. Determining WWW user's next access and its application to pre-fetching. In *Proceedings of ISCC'97*, July 1997.

[CKR98a] E. Cohen, B. Krishnamurthy, and J. Rexford. Evaluating server-assisted cache replacement in the Web. In *Proceedings of the European Symposium on Algorithms-98*, 1998.

[CKR98b] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the Web using server volumes and proxy filters. In *Proceedings of Sigcomm*, 1998.

[CKR99] E. Cohen, B. Krishnamurthy, and J. Rexford. Efficient algorithms for predicting requests to Web servers. In *Proceedings of Infocom*, 1999.

[CL97] P. Cao and C. Liu. Maintaining strong cache consistency in the World Wide Web. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems*, May 1997.

[CY97] K. Chinen and S. Yamaguchi. An interactive prefetching proxy server for improvement of WWW latency. In *Proceedings of INET'97*, June 1997.

[CZB98] P. Cao, J. Zhang, and K. Beach. Active cache: caching dynamic contents on the Web. In *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98)*, 1998.

[DFKM97] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, December 1997.

[dig] Digital island, `http://www.digitalisland.com`.

[DMF] B. M. Duska, D. Marwood, and M. J. Feelay. The measured access characteristics of World Wide Web client proxy caches. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*. `cs.ubc.ca/spider/feeley/wwwap/wwwap.html`.

[DP96] A. Dingle and T. Partl. Web cache coherence. In *Proceedings of the Fifth International World Wide Web Conference*, 1996.

[FCAB98] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. In *Proceedings of Sigcomm*, 1998.

[FCD+99] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *Proceedings of Infocom*, 1999.

[FCLJ99] L. Fan, P. Cao, W. Lin, and Q. Jacobson. Web prefetching between low-bandwidth clients and proxies: potential and performance. In *Proceedings of Sigmetrics*, 1999.

[GRC97] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: an approach to building large Internet caches. In *Proceedings of the HotOS'97 Workshop*, May 1997. `www.cs.duke.edu/ari/cisi/crisp-recycle/crisp-recycle.htm`.

[GS94] J. Gwetzman and M. Seltzer. The case for geographical pushing-caching. In *Proceedings of the HotOS Conference*, 1994. `ftp://das-ftp.harvard.edu/techreports/tr-34-94.ps.gz`.

[GS96] J. Gwetzman and M. Seltzer. World Wide Web cache consistency. In *Proceedings of the USENIX Technical Conference*, January 1996.

[Hal97] B. Halabi. *Internet Routing Architectures*. Cisco Press, 1997.

[HMY] A. Heddaya, S. Mirdrad, and D. Yates. Diffusion based caching along routing paths. `cs-www.bu.edu/faculty/heddaya/Pepers-NonTR/webcache-wkp.ps.Z`.

[HTTa] Hypertext Transfer Protocol – HTTP/1.0. RFC 1945.

[HTTb] Hypertext Transfer Protocol – HTTP/1.1. RFC 2068.

[KD99] M. R. Korupolu and M. Dahlin. Coordinated placement and replacement for large-scale distributed caches. In *Proceedings of the IEEE Workshop on Internet Applications*, July 1999.

[Kes97] S. Keshav. *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley, 1997.

[KLL+97] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *STOC*, 1997.

[KLM97] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, December 1997.

[KRS00] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. In *IEEE/ACM Transactions on Networking*, August 2000.

[KS98] P. Krishnan and B. Sugla. Utility of co-operating Web proxy caches. In *Computer Networks and ISDN Systems*, April 1998.

[KW97] B. Krishnamurthy and C. E. Wills. Study of piggyback cache validation for proxy caches in the World Wide Web. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[KW98] B. Krishnamurthy and C. E. Wills. Piggyback server invalidation for proxy cache coherency. In *Proceedings of the WWW-7 Conference*, 1998.

[KW99] B. Krishnamurthy and C. E. Wills. Proxy cache coherency and replacement - towards a more complete picture. In *Proceedings of ICDC99*, June 1999.

[KW00] B. Krishnamurthy and J. Wang. On network-aware clustering of Web clients. In *Proceedings of ACM SIGCOMM*, August 2000. `http://www.acm.org/sigcomm/sigcomm00/program.html`.

[LA94] A. Luotonen and K. Altis. World Wide Web proxies. In *Computer Networks and ISDN Systems, First International Conference on WWW*, April 1994.

[LAISD99] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias. Design and performance of Web server accelerator. In *Proceedings of Infocom'99*, 1999.

[LB97] T. S. Loon and V. Bharghavan. Alleviating the latency and bandwidth problems in WWW browsing. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, December 1997.

[LG98] U. Legedza and J. Guttag. Using network-level support to improve cache routing. In *Computer Networks and ISDN Systems*, November 1998.

[LGI+99] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby. On the optimal placement of Web proxies in the Internet. In *Proceedings of Infocom'99*, 1999.

[LMJ97] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. In *Proceedings of ACM SIGCOMM*, September 1997. `http://www.acm.org/sigcomm/sigcomm97/program.html`.

[LR90] K. Lougheed and Y. Rekhter. A Border Gateway Protocol. RFC 1163, IETF, June 1990. `http://www.ietf.org/rfc/rfc1163.txt`.

[LRV] P. Lorenzetti, L. Rizzo, and L. Vicisano. Replacement policies for a proxy cache. `www.iet.unipi.it/luigi/research.html`.

[MC98] E. P. Markatos and C. E. Chronaki. A top-10 approach to prefetching on the Web. In *Proceedings of INET'98*, 1998.

[merle] Merit Internet Performance Measurement and Analysis Project, 1999, `http://www.merit.edu/~ipma/routing_table`.

[MLB95] R. Malpani, J. Lorch, and D. Berger. Making World Wide Web caching servers cooperate. In *Proceedings of the 4th International WWW Conference*, December 1995. `www.w3j.com/1/lorch.059/paper/059.html`.

[MNR+98] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson. Adaptive Web caching: towards a new caching architecture. In *Computer Network and ISDN Systems*, November 1998.

[Moy98] J. T. Moy. *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.

[nla97] NLANR network analysis infrastructure, November 1997. `http://moat.nlanr.net/IPaddrocc`.

[ore97] Oregon Exchange BGP Route Viewer, December 1997. Host: `route-views.oregon-ix.net`.

[pat] Pathchar. `http://www.caida.org/tools/utilities/others/pathchar/`.

[PH97] D. Povey and J. Harrison. A distributed Internet cache. In *Proceedings of the 20th Australian Computer Science Conference*, Feburary 1997.

[PM96] V. N. Padmanabhan and J. C. Mogul. Using predictive prefetching to improve World Wide Web latency. In *Proceedings of Sigcomm'96*, 1996.

[PM99] T. Palpanas and A. Mendelzon. Web prefetching using partial match prediction. In *Proceedings of WCW'99*, 1999.

[Rab] M. Rabinovich. Issues in Web content replication.

[RCG98] M. Rabinovich, J. Chase, and S. Gadde. Not all hits are created equal: cooperative proxy caching over a wide-area network. In *Computer Networks And ISDN Systems*, November 1998.

[REL98] Relais: cooperative caches for the World Wide Web, 1998. `www-sor.inria.fr/projects/relais/`.

[RSB99] P. Rodriguez, C. Spanner, and E. W. Biersack. Web caching architectures: hierarchical and distributed caching. In *Proceedings of WCW'99*, 1999.

[RW98] A. Rousskov and D. Wessels. Cache digest. In *Proceedings of 3rd International WWW Caching Workshop*, June 1998.

[SCA99] S. Savage, N. Cardwell, and T. Anderson. The case for informed transport protocols. In *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, March 1999.

[sines] SingAREN BGP routing table, December 1999, `http://noc.singaren.net.sg/netstats/routes`.

[SSV97] P. Scheuermann, J. Shim, and R. Vingralek. A case for delay-conscious caching of Web documents. In *Proceedings of the 6th International WWW Conference*, April 1997.

[TDVK98] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Beyond hierarchies: design considerations for distributed caching on the Internet. Technical Report TR98-04, Department of Computer Science, University of Texas at Austin, February 1998.

[vbnml] VBNS route information, December 1999, `http://www.vbns.net/route/index.html`.

[VR] V. Valloppillil and K. W. Ross. Cache array routing protocol v1.0.

[WA97] R. P. Wooster and M. Abrams. Proxy caching that estimates page load delays. In *Proceedings of the 6th International WWW Conference*, April 1997. `www.cs.vt.edu/~chitra/docs/www6r/`.

[Wan97] Z. Wang. Cachemesh: a distributed cache system for the World Wide Web. In *Web Cache Workshop*, 1997.

[Wan99] J. Wang. A survey of Web caching schemes for the Internet. In *ACM Computer Communication Review*, October 1999.

[WAS+96] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox. Removal policies in network caches for World Wide Web documents. In *Proceedings of Sigcomm'96*, 1996.

[WC] D. Wessels and K. Claffy. Internet cache protocol (ICP), version 2. RFC 2186.

[Wes95] D. Wessels. Intelligent caching for World Wide Web objects. In *Proceedings of INET'95*, June 1995. `info.isoc.org/HMP/PAPER/139/archive/papers.ps.9505216`.

[WM99] C. E. Wills and M. Mikhailkov. Towards a better understanding of Web resources and server responses for improved caching. In *Proceedings of the WWW8*, May 1999. `http://www.cs.wpi.edu/~cew/papers/www8.ps.gz`.

[WS94] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated (Volumn 2)*. Addison-Wesley, 1994.

[WVS+99] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-based analysis of Web-object sharing and caching. In *Proceedings of the Second USENIX Conference on Internet Technologies and Systems (USITS)*, October 1999.

[YWMW] J. Yang, W. Wang, R. Muntz, and J. Wang. Access driven Web caching. Technical Report 990007, UCLA.

[ZQK00] Y. Zhang, L. Qiu, and S. Keshav. Speeding up short data transfers: Theory, architectural support, and simulation results. In *Proceedings of NOSSDAV2000*, June 2000.