On Network-Aware Clustering of Web Clients

Balachander Krishnamurthy
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ, USA
bala@research.att.com

Jia Wang
Cornell University
Ithaca, NY, USA
jiawang@cs.cornell.edu

ABSTRACT

Being able to identify the groups of clients that are responsible for a significant portion of a Web site's requests can be helpful to both the Web site and the clients. In a Web application, it is beneficial to move content closer to groups of clients that are responsible for large subsets of requests to an origin server. We introduce *clusters*—a grouping of clients that are close together topologically and likely to be under common administrative control. We identify clusters using a "network-aware" method, based on information available from BGP routing table snapshots.

Experimental results show that our entirely automated approach is able to identify clusters for 99.9% of the clients in a wide variety of Web server logs. Sampled validation results show that the identified clusters meet the proposed validation tests in over 90% of the cases. An efficient self-corrective mechanism increases the applicability and accuracy of our initial approach and makes it adaptive to network dynamics. In addition to being able to detect unusual access patterns made by spiders and (suspected) proxies, our proposed method is useful for content distribution and proxy positioning, and applicable to other problems such as server replication and network management.

1. INTRODUCTION

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., 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, 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. We use the term *cluster* to denote such a grouping.

A complete partition of an arbitrary set of IP addresses would require knowledge that is not available to anyone outside the administrative entities. Instead, we make use of information available in BGP routing table snapshots. This paper presents an entirely automated technique for a novel "network-aware" method of identifying client clusters starting with Web server logs and using readily available routing information.

As a primary application of our clustering, we examine Web caching. 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 caching frequently accessed resources to reduce latency for future accesses. We also need to detect hosts with unusual access patterns, such as spiders or proxies. 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 later in a search application.

A simple approach to identifying client clusters is to assume that the clients that share the same first three bytes in their IP addresses belong to the same cluster. This is one approach to cluster identification if one relied solely on client information extracted from logs. However, as we show later, this over-simplified assumption fails a validation test in over 50% of the sampled cases. Our method identifies client clusters based on routing information. We extract IP addresses from Web server logs and cluster them using BGP routing information. The experimental 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 test 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 make it adaptive to network dynamics. Our method also detects spiders and proxies, if any, in the logs.

Section 2 describes a simple approach to client clustering and gives quantitative measurements on its error ratio. Our approach to identifying client clusters is presented in Section 3. Section 4 addresses several applications of client clustering. After briefly examining related work in Section 5, we conclude with possible improvements to our work in the future.

2. SIMPLE APPROACH

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. However, it is well-known that IP addresses are not all organized in this fashion. Figures 1(a) and (b) show the distribution of prefix lengths extracted from Mae-West NAP [16] routing table snapshots taken on July 3, 1999, and the prefix length distribution over time from July 3 to July 6, 1999, respectively. While around 50% of the prefix lengths are 24 bits (see Figure 1(b)), 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 (*Classless InterDomain Routing*) addresses [1] and route aggregation[2].

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. 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 prefixes 151.198.194.16, 151.198.194.32, and 151.198.194.48, and netmask length of 28. The names of these hosts are *client-151-198-194-17.bellatlantic.net*, *mail-srv1.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. In this case, a trace-driven simulation may underestimate benefit of Web proxies due to less proxy sharing across the mis-identified clusters. In our experiments the simplified assumptions of the simple approach fail a validation test in over 50% of the sampled cases.

[1]Internet's growth in recent years led to possible exhaustion of Class B network address space and routers not being able to manage the size of routing tables. CIDR 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 [11, 17].

[2]With CIDR address allocation mechanism, the routing table can be shrunk by aggregating routing entries with adjacent IP address blocks and same routing path [11, 17].

As an alternate approach, one could identify client clusters based on Class A, Class B, and Class C networks. 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. 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 paper, we use the simple approach as a comparison.

3. OUR APPROACH

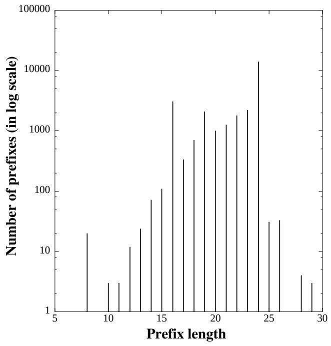
We propose a novel method to identify client clusters by using the prefixes and netmasks information extracted from the BGP (Border Gateway Protocol [15, 11, 17]) routing and forwarding table snapshots. The Internet consists of a large collection of hosts connected by networks of links and routers. It 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). An AS typically has very detailed knowledge of its internal topology and limited reachability information about other ASes. Interdomain routing protocols (such as BGP) control packet forwarding among ASes and interdomain reachability information is maintained at the routers of each AS that speak BGP.

The rationale behind our approach is that the prefixes and corresponding netmasks identify the routes in the routing table which are used by core 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. 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 Our approach makes use of snapshots of routing tables. Although these tables will change over time, we find our approach performs very well for several tests of reasonablenes.

Figure 2 gives a graphical view of the five steps in our approach: (i) network prefix/netmask extraction and extraction of IP addresses from Web server logs; (ii) client cluster identification; (iii) (optional) validation of our method by sampling the resulting client clusters; (iv) examining effect of network dynamics on client cluster identification; and (v) 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 section and also look at other issues affecting client cluster identification.

3.1 Network prefix extraction

The first step of our approach is to generate a combined prefix/netmask entry table. We do this by extracting pre-



(a)

Prefix length		15	16	17	18	19	20	21	22	23	24	26
Date	7/3/99	111	3098	333	706	2092	1009	1275	1805	2227	13937	34
	7/4/99	100	3099	331	702	2091	1003	1268	1797	2220	14029	33
	7/5/99	100	3095	331	703	2074	1006	1263	1798	2220	14013	33
	7/6/99	100	3097	332	704	2084	1006	1265	1794	2220	14018	33

(b)

Figure 1: Distribution of prefix lengths extracted from mae-West NAP routing table snapshots: (a) histogram of prefix lengths on 7/3/1999, (b) prefix length distribution from 7/3/1999 to 7/6/1999.

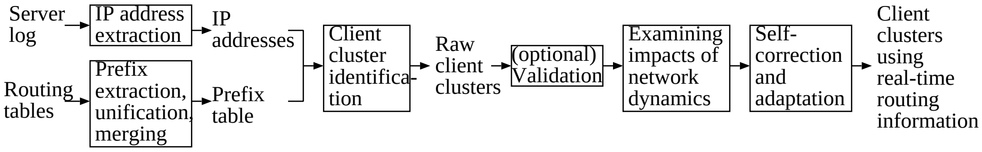


Figure 2: The entire automated process of client cluster identification.

fix/netmask entries from the routing table snapshots, unifying prefix/netmask entries into a standard format, and inserting 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.

3.1.1 *Prefix entry extraction*

The BGP routing tables (Table 1) are collected automatically via a simple script from AADS, MAE-EAST, MAE-WEST, PACBELL, PAIX (all from [16]), ARIN [3], AT&T-Forw and AT&T-BGP (from [4]), CANET [8], CERFNET [9], NLANR [18], OREGON [19], SINGAREN [20], and VBNS [21]. We do so by downloading them from well-known Web sites (e.g., AADS) or *telnet*ing to a particular host to run a script to dump routing tables (e.g., OREGON). The size of the table depends on the location of the BGP routers and the AS it belongs to. We assembled a total of 391,497 unique prefix/netmask entries[3]. An example BGP routing table snapshot is shown in Table 2. Though the routing table also contains interdomain information such as next hop IP address, AS number, and AS path, we have only used the prefix/netmask information in our experiments. The AS number and path information can also provide hints on the geographical location of clients.

Although ARIN and NLANR [4] are the two largest tables, they are not directly generated from BGP information, but from a dump of the networks registered at 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 our 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 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%.

3.1.2 *Network prefix format unification and merging entries*

The network prefix/netmask entries extracted from the various routing tables are in one of three different formats: (i) $x1.x2.x3.x4/k1.k2.k3.k4$ where $x1.x2.x3.x4$ and $k1.k2.k3.k4$ are network prefix and netmask with zeroes dropped at the tail, (ii) $x1.x2.x3.x4/l$ where l is the netmask length, and (iii) $x1.x2.x3.0$ which 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 length is 8, 16, or 24, respectively). We unify the different formats, arbitrarily choosing the first format as our standard

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

[4] 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 a very minor impact on our results (only $\sim 0.1\%$ of clients are clustered by network prefixes taken from NLANR's IP network dump).

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

Table 1: Our collection of BGP routing tables.

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

Table 2: An example snapshot of BGP routing table (VBNS).

format and converting all prefix/netmask entries to this format. As Table 1 shows, 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.

3.2 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.

3.2.1 *Methodology*

Clustering of clients involves three steps: (i) extract client IP addresses from the server log; (ii) perform the longest prefix matching (similar to what IP routers do) on each client IP address using the constructed prefix/netmask table; (iii) classify 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 Section 4).

3.2.2 *Experiments*

We conducted our experiments on a very wide range of Web server logs ranging in number of requests (148K to 46 Million), number of clients (40K to 481K), 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. Here, we show the Nagano server log, a 1 day (February 13, 1998) extract from the 1998 Winter Olympic Games server log,[5] as a primary example to illustrate our results, with an occasional glimpse at results of other logs. The results on over a dozen other logs (Apache, Easy World Wide Web, Sun, Unav) are similar across all experiments [5].

The Nagano server log has a total of 11,665,713 requests issued by 59,582 clients accessing 33,875 unique URLs. We extract the IP addresses[6] of these clients, run the cluster identification algorithm and group all of them 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.

[5] 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 section).

[6] 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 doesn't know its own IP address were ignored and client 0.0.0.0 was excluded from our experiments.

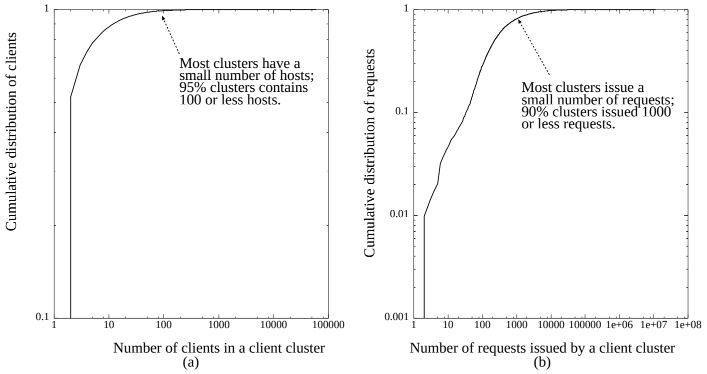


Figure 3: The cumulative distribution of clients and requests in a client cluster for the Nagano server log (y axis is in log scale): (a) is the cumulative distribution of number of clients in client clusters; (b) is the cumulative distribution of number of requests issued from within client clusters.

In order to get more insight on client clusters, we plot the cumulative distribution of clients and requests in a client cluster in Figure 3. Figure 3(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 less than 100 clients. The cumulative distribution of requests issued by a client cluster is shown in Figure 3(b), which is more heavy-tailed than that of the number of clients in a client cluster as shown in Figure 3(a), implying possible existence of proxies and/or spiders. Around 90% of the client clusters issued less 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 [7].

Figures 4(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 number of clients in a cluster (i.e., larger client clusters are on the left side). From Figures 4(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 4 with different sorting of clusters (on x axis)—in reverse order of number of requests. Figures 5(a), (b), and (c) show the distributions of 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 5(a) with Figure 4(a), the results further show that the distribution of number of requests issued from within client clusters is more heavy-tailed than that of number of clients in client clusters. Figures 5(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 very small number of clients and may access very few URLs—these are also useful measures in identifying spiders and proxies.

Note that Figures 4(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 4(a), (b), and (c) refer to the same client cluster. The same is true for Figure 5.

Although we don't include the detailed results of other server logs, we provide a glimpse of client cluster distributions of Apache, EW3, Nagano, and Sun logs in Figure 6. Figures 6 (a) and (b) are the distributions of number of clients and number of requests in client clusters in reverse order of number of clients, respectively. Figures 6 (c) and (d) are the distributions of number of requests and number of clients in client clusters in reverse order of 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 6(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 3.5).

3.3 Validation

We now validate our approach of identifying client clusters. 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 generally errs by producing clusters that are too small. Our approach can produce clusters that are too big, but in a lot fewer cases, as we will see below.

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

Nslookup queries Internet domain name servers for information about hosts. 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[7]. For example, *macbeth.cs.wits.ac.za* and

[7]We say that two clients share a non-trivial suffix if the last n components of their names are 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 $n = 2$.

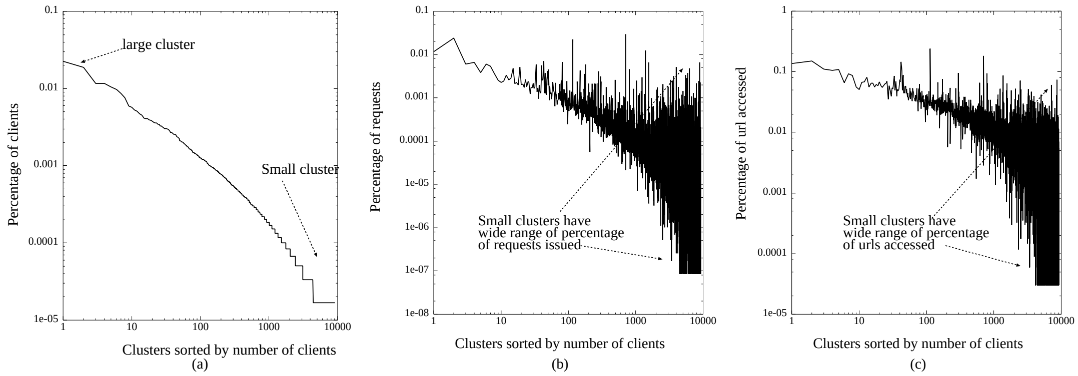


Figure 4: The client cluster distribution for the Nagano server log plotted in reverse order of number of clients in a cluster (both x axis and y axis are in log scale): (a) is the distribution of number of clients in client clusters; (b) is the distribution of the number of requests issued from within client clusters; and (c) is the distribution of the number of URLs accessed from within client clusters. Note that larger client clusters are on the left.

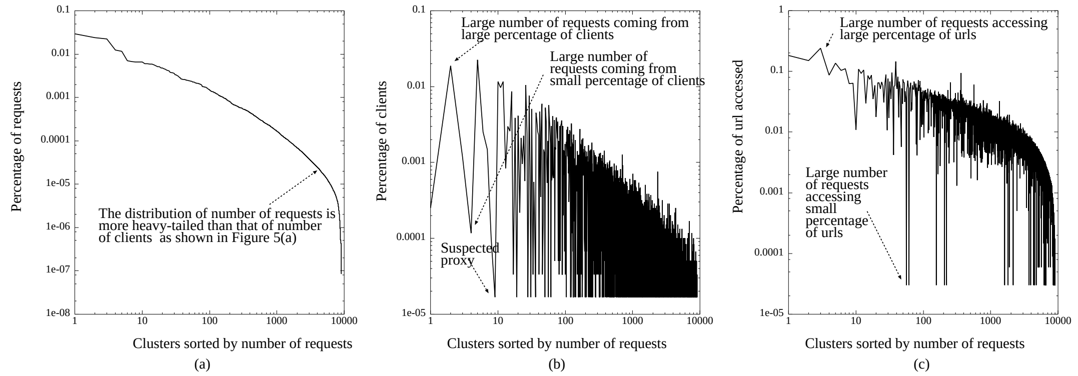


Figure 5: The client cluster distribution for the Nagano server log plotted in reverse order of number of requests (both x axis and y axis are in log scale): (a) is the distribution of number of requests issued from within client clusters (re-plot of Figures 4(b)); (b) is the distribution of number of clients in client clusters (re-plot of Figure 4(a)); and (c) is the distribution of number of URLs accessed from within client clusters (re-plot of Figure 4(c)).

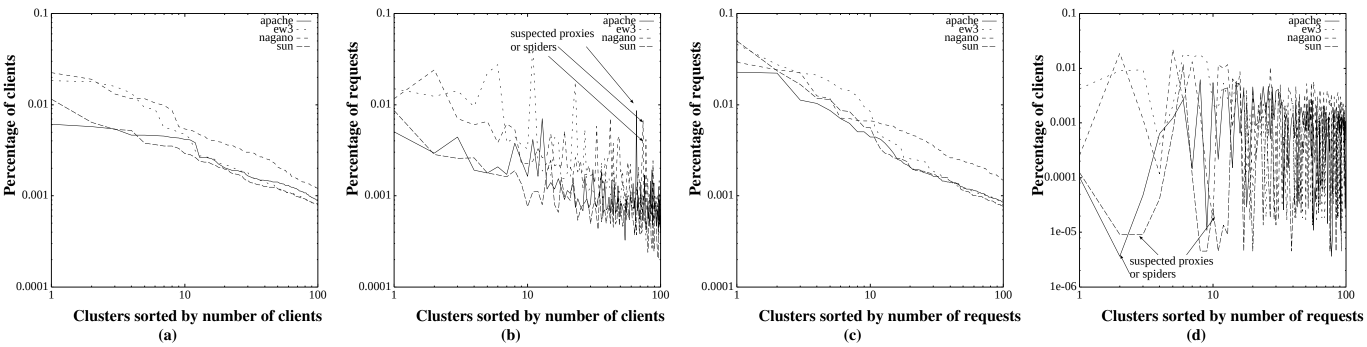


Figure 6: Comparison of client cluster distributions of Apache, EW3, Nagano, and Sun server logs (both x axis and y axis are in log scale): (a) and (b) are the distributions of number of clients and number of requests in client clusters in reverse order of number of clients, respectively; (c) and (d) are the distributions of number of requests and number of clients in client clusters in reverse order of number of requests, respectively.

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 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 (see [5] 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, i.e., 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 are looking into merging such clusters as part of ongoing work.

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. However, it is hard to further quantify its actual effect on client clustering.

The *nslookup*-based validation has some limitations. As Table 3 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. Possible reasons include DNS servers not giving out names of machines behind a firewall, machines on the local network acquiring dynamic IP addresses via a DHCP server (DNS server will not have the registration record for the dynamic addresses which do not have a one-to-one mapping to a machine), or an ISP not having registered any names for its customers. We use an optimized *traceroute* to resolve the clients which are not locatable by *nslookup* in our experiments. We describe the optimized *traceroute* next.

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 doesn't 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 *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* doesn't get a ICMP reply packet from the designated router which is either unreachable or unwilling to give out the required information due to firewalls.

The advantage of using the optimized *traceroute*[8] 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 network and time spent on resolving clients has been significantly reduced by our optimized *traceroute*. We estimate that we can save 90% of the probes and 80% of the waiting time by our modified *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

[8]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).

Server log	Apache	Nagano	Sun
Total number of client clusters	35563	9853	33468
Number of sampled client clusters	382	111	365
Number of sampled clients	2437	307	2217
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	1470	172	1116
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	2437	307	2217
Number of mis-identified client clusters	35	12	33
Number of mis-identified non-US client clusters	20	7	28

Table 3: Client cluster validation of Apache, Nagano, and Sun logs using DNS *nslookup* and optimized *traceroute*.

(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 (the results on other server logs are in [5]). Our results show that 90% of 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 3.5.

Many real applications will be tolerant to a certain degree of inaccuracy and an alternative way to validate is to set a threshold (say 5%) and selectively sample clients. For example, if 95% of the clients inside the 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. We plan to study selective sampling techniques in our future work.

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 7. Figures 7(a) and (b) show the distributions of number of clients in client clusters in reverse order of number of clients and in reverse order of number of requests. 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 7(c) and (d) show the distribution of number of requests issued from within client clusters in reverse order of number of requests and in reverse order of 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 implies that the simulation results based on different client cluster identification methods may vary a lot.

3.4 Effect of BGP dynamics on client cluster identification

An update of BGP routing table is triggered by changes in network reachability and topology, as well as policy changes. BGP dynamics is a well-known phenomenon which affects the performance of Internet applications [14]. We examine the impact of BGP dynamics on client cluster identification results. The experiments are 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 change during the entire testing period, i.e., the set of prefixes which are not in the intersection of prefixes of all the routing 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 routing table (Table 4) as an example to illustrate the effect of BGP dynamics (results for other routing tables are in [5]). We show measurement results for 1, 4, 7, and 14 day periods on Apache, EW3, Nagano, and Sun server logs. 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 less 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

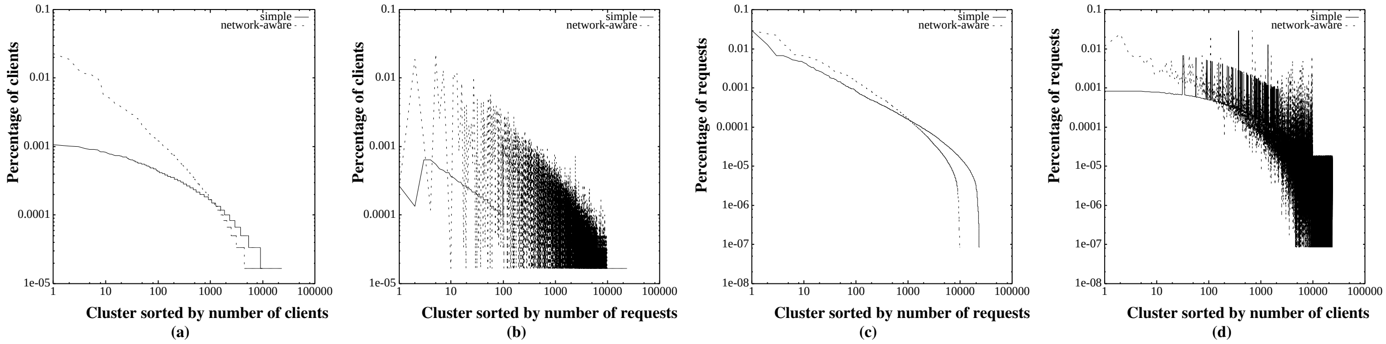


Figure 7: Comparison of the client cluster distributions of the Nagano server log obtained by our approach and the simple approach (both x axis and y axis are in log scale): (a) and (b) are the distributions of number of clients in client clusters in reverse order of number of clients and number of requests, respectively; (c) and (d) are the distributions of number of requests issued from within client clusters in reverse order of number of clients and number of requests, respectively.

changes dynamically according to the network environment, its impact on client cluster identification is very minor, and fixed in the self-correction and adaptation process (discussed next).

3.5 Self-correction and adaptation

Besides validating the client cluster results, we use *traceroute* for two other purposes. In Section 3.2.2, we showed that more than 99.9% clients in the Web server logs can be clustered using our method. We have also shown in Section 3.3 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 very important to generate client clusters using real-time routing information and producing real-time client cluster identification results. By real-time cluster identifying we mean application of cluster identifying techniques to very recent server log data (within the last few minutes).

We consider two cases in the self-correction and adaptation process: (i) 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; (ii) 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.

3.6 Other issues

We examine three related issues: partitioning a session into time periods, identifying server clusters in client and proxy logs, and clustering client clusters themselves.

To 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. 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 in [5].

Our method can also be used to identify *server* clusters in proxy logs. In a set of servers accessed over a 11-day period in a large ISP's client trace we found 69,192 unique server IP addresses, with only 153 ($\sim$ 0.2% of the total) that were not clusterable (due to lack of proper network prefix/netmask information). Roughly 4% (729 out of 17,192) of the server clusters received 70% of the 12.4 Million requests. Identifying client *and* server clusters can aid in engineering and shaping traffic in a content distribution application.

After identifying client clusters based on the BGP routing table information, we can further cluster nearby client clusters into *network clusters*. We use *traceroute* to do the higher level clustering. Typically, we run *traceroute* on a number of ($r \geq 1$) randomly selected clients in each cluster and do suffix matching on the path towards each destination network. The second level clustering is useful in applications such as selective content distribution, proxy placement, and load balancing.

4. APPLICATIONS OF CLIENT CLUSTERING

The client cluster identification 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. This is crucial information for service providers to enhance their services, reduce costs,

Period (days)	0	1	4	7	14
AADS prefix	16595	16669	16704	16792	17288
Maximum effect	711	785	820	908	1404
Apache prefix (total 35563)	3932	3935	3929	3950	4035
Maximum effect	124	127	121	142	227
Apache busy clusters (total 2869)	605	603	603	605	614
Maximum effect	22	20	20	22	31
EW3 prefix (total 24921)	2592	2588	2582	2604	2682
Maximum effect	75	71	65	87	165
EW3 busy clusters (total 2062)	385	386	388	390	412
Maximum effect	4	5	7	9	31
Nagano prefix (total 9853)	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 33468)	3646	3651	3640	3669	3756
Maximum effect	124	129	118	147	234
Sun busy clusters (total 2536)	527	525	529	530	542
Maximum effect	15	13	17	18	30

Table 4: The effect of AADS dynamics on client cluster identifying.

and extend global presence. We present one application of clustering—Web caching. 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 Web caching simulation

We examine the usefulness of network aware clustering in a Web caching simulation and contrast it with the simple approach. We present our approach to identifying spiders and proxies and then describe proper placement of proxies between clients and servers.

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 (almost) every resource in 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 8(a)) since the spider will issue most of the requests and a majority of the URLs accessed by it will not be accessed repeatedly. Next, we locate existing proxies since (Figure 8(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 when 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).

4.1.2 Identifying spiders/proxies

A spider cluster often issues a very large amount of requests within a short period and the host(s) within that client cluster issuing a large percentage of the requests is suspected to be a spider. 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/proxies by examining the request arrival time. From Figure 9(c), we don't see any similarity between the access pattern of the spider and that of the entire server log shown in Figure 9(a). There are certain correspondences between the access pattern of the proxy shown in Figure 9(b) 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 10. 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 behind it (see Figure 9). 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

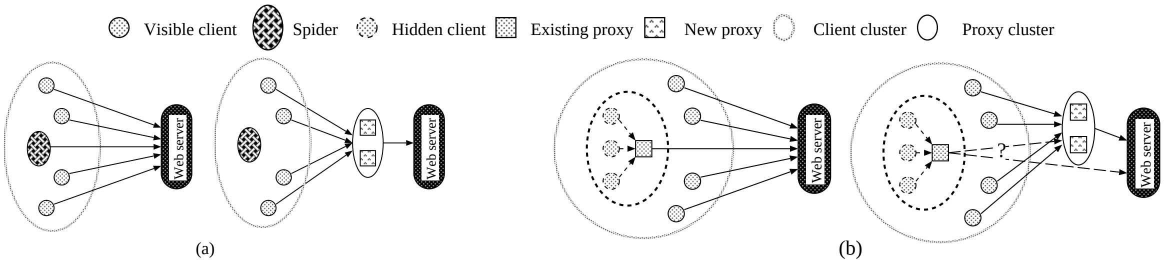


Figure 8: Eliminating spiders and existing proxies from the server logs: (a) spiders; (b) existing proxies.

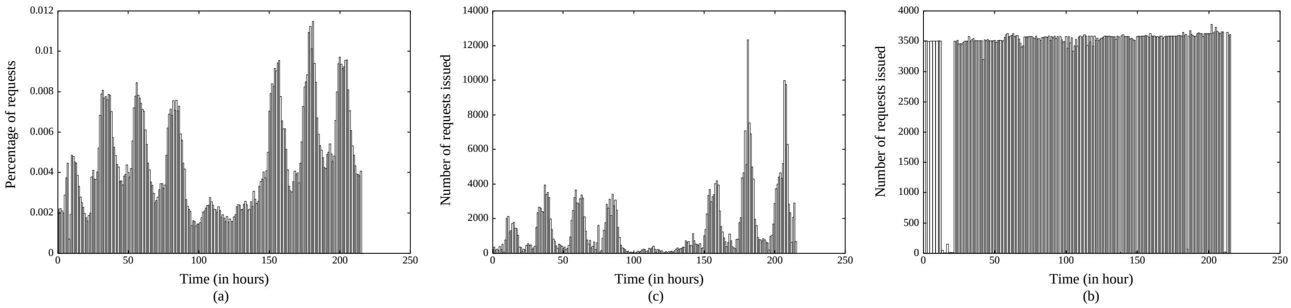


Figure 9: Histogram of the requests in the Sun server log: (a) the entire server log, (b) a client cluster containing a proxy, (c) a client cluster containing a spider.

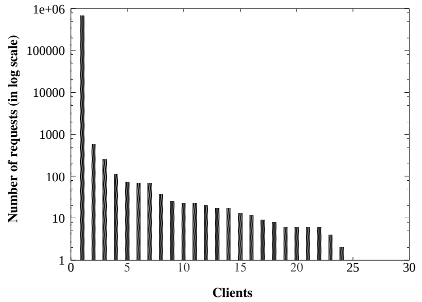


Figure 10: The client requests distribution of a client cluster containing a spider in the Sun server log which issues 692,453 requests (99.79% of all requests in the cluster).

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 4(b) and 5(b) in Section 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 request, 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 are a lot of different `User-Agent` fields from the same host responsible for a lot of requests, then it is very 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% 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 5). 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 less-busy client clusters, filtered out, ranges from 1 to 46 clients and the number of requests issued by 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 [5].

Similar thresholding on clusters identified using the simple approach leads 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 less-busy client clusters ranges from 1 to 695. The results are very different from that of the network-aware approach, which implies the simple approach does not do a good job on grouping clients.

Approach	Network-aware	Simple
Total number of client clusters	9853	23523
Threshold (requests per client cluster)	2744	696
Number of busy client clusters	717 (32691 clients, 8167590 requests)	3242 (30774 clients, 8167335 requests)
Busy client clusters (requests)	2744 - 339632 (1 - 1343 clients)	696 - 339632 (4 - 63 clients)
Less-busy client clusters (requests)	1 - 2741 (1 - 46 clients)	1 - 695 (1 - 4 clients)

Table 5: Experimental results of thresholding client clusters on the Nagano server log.

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, number of requests issued, the URLs accessed, or the number of bytes fetched from server. The proxies assigned to clients in the same client cluster form a proxy cluster and would co-operate 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. Here, we only address the first approach in our experiments.

4.1.5 Experimental results

We use the client cluster results in the trace-driven simulation to evaluate the benefit of proxy caching. We place proxies in front of each client cluster. We implement the Piggyback Cache Validation [12] scheme with a fixed *ttl* (time-to-live) expiration period at each proxy cache. By default, a cached resource is considered stale once a period of one hour has elapsed. When the 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. 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 several logs [5] though we only show the results on the Nagano server log[9].

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 total hit ratio and byte hit ratio observed at server. Figure 11 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

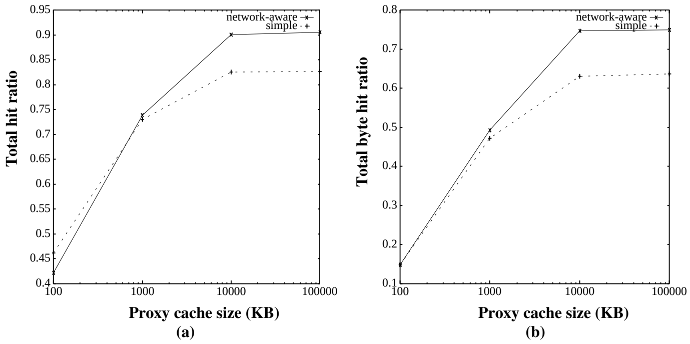


Figure 11: Simulation results on Web server performance vs proxy cache size of the Nagano server log (x axis is in log scale): (a) is the total hit-ratio, (b) is the total byte hit ratio.

of the simple approach. Figures 11(a) and (b) show the total hit ratio and byte hit ratio observed at 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 under-estimate (around 10%) both hit ratio and byte hit ratio observed at server when 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 rates of around 40% [2] 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 than other logs. We didn't observe a high hit ratio on all the logs.

To evaluate proxy performance, we fix the cache size as infinite and examine the hit ratio and byte hit ratio at each proxy (Figure 12). 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 top 100 client clusters in the reverse order of number of requests (i.e., the first 100 client cluster if we ordered them in the reverse order of number of requests issued). Figures 12(a) and (b) show the number of requests and size (in KB) of requests issued in client clusters in reverse order of number of requests, respectively. Figures 12(c) and (d) show the hit ratio and byte hit ratio observed at each proxy. The great differences of the client requests and proxy hit ratio results got 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

[9] Requests to resources accessed by clients less than 10 times are ignored.

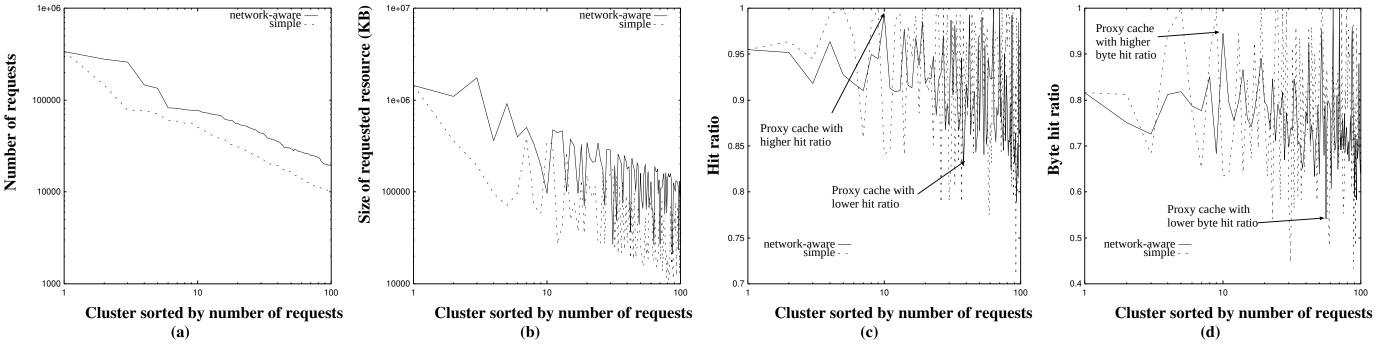


Figure 12: Simulation results on proxy cache performance of the top 100 client clusters in the Nagano server log (x axis is in log scale): (a) and (b) are the number of requests and size of requested resources in client clusters in reverse order of number of requests, respectively; (c) and (d) are the proxy cache hit ratio and byte hit ratio of client clusters in reverse order of number of requests, respectively.

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 on 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 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.

4.2 Other applications of client clustering

Besides Web caching, client clustering is also useful in several other applications. Real-time clustering enables content distribution service providers to deliver the right content to the right customer at the right time. 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. Also, with the knowledge of customer's location and the traffic load origination, the service providers can to do better load balancing by provisioning in advance.

5. RELATED WORK

We are not aware of any work that uses BGP routing information to do clustering of clients for the range of applications we have examined. Simply using *nslookup* to do clustering is both expensive and unlikely to yield full results. A clustering model based on distance between servers and clients [6] was used to lower the cost of document dissemination; the implementation used *traceroute*. Using *traceroute* along the lines of [13] (although they use it for a different purpose) or [6] is also more expensive than using routing table information. In particular it is not feasible for using it under real-time considerations, e.g., when a Web event is in progress. Several efforts are underway at research and product levels in improving Web caching and content distribution. At the DNS layer some companies (e.g., Akamai [1], Digital Island [10]) use client's location to serve content from a nearby site to lower load on the server and user perceived latency. The clustering methodology could be used by such companies.

6. CONCLUSION AND FUTURE WORK

We have proposed a novel way to identify client clusters using BGP routing information. Experimental results show that our method is able to group more than 99.9% of the clients captured in a wide variety of server logs into clusters. Sampling validation results demonstrate that our method passes validation tests in over 90% of the sampled 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 server log to a set of interesting client clusters. The cluster information can be used in applications such as content distribution, caching, and network management. A useful by-product is identification of spiders and proxies among Web clients by examining the access patterns of corresponding client clusters. Our methodology is immune to BGP dynamics, independent of the range and diversity of server logs, and computationally non-intensive. Ongoing work includes using information on ASes to reduce the error ratio of client cluster identification.

Acknowledgments

We would like to thank several people who answered questions including Jennifer Rexford, Anja Feldmann, Tim Griffin, Bill Manning, Vern Paxson, Craig Labovitz, Thomas Narten, and Steven Bellovin. We would also like to thank those who read an earlier version of this document and offered suggestions including Jennifer Rexford, Chuck Cranor, Tim Griffin, Oliver Spatscheck, Sylvia Ratnaswamy, Phong Vo, Walter Willinger, Craig Wills. Discussions with Emden Gansner, Nick Duffield, and S. Keshav were also beneficial. We thank Ellen Zegura for her significant help in improving the presentation of the paper. We thank those who provided us with Web server logs, without which such experimental evaluations would be impossible. We invite large portal sites to make their logs available as well.

7. REFERENCES

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

[2] 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.

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

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

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

[6] A. Bestavros and C. Cunha. Server-initiated Document Dissemination for the WWW. In *IEEE Data Engineering Bulletin*, September 1996.

[7] 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. http://www.research.att.com/~breslau/pubs/zipf.ps.gz.

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

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

[10] Digital island, http://www.digitialisland.com.

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

[12] B. Krishnamurthy and C. E. Wills. Study of piggyback cache validation for proxy caches in the World Wide Web. In *Proc. USENIX Symp. on Internet Technologies and Systems*, pages 1–12, December 1997, http://www.usenix.org/events/usits97.

[13] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. Under submission, http://www.cs.bell-labs.com/who/shavitt/pub/stat.ps.gz.

[14] 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.

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

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

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

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

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

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

[21] Vbns route information, December 1999, http://www.vbns.net/route/index.html.