DNS-based Mechanism for Policy-added Server Selection

Toshihiko SHIMOKAWA† Norihiko YOSHIDA‡ Kazuo USHIJIMA †
† Kyushu University ‡ Nagasaki University

Abstract—

Many service providers on the Internet use multiple mirror servers to cope with request congestion. Here, the problem is how to select the best server out of them.

We propose a DNS-based mechanism for server selection with preferred policies. DNS (Domain Name System) is used in almost all services and implementations. Therefore, our approach is generally applicable. The mechanism consists of two components. One is query preprocessor. The other is network status observer. The network status observer probes the status of network periodically. The query preprocessor traps queries to the DNS, and selects the optimal server automatically under a given policy. Some practical experiments substantiated the effectiveness of our mechanism.

Keywords— **Internet, Server selection, Load balancing, DNS, Name Server, Tenbin**

I. Introduction

There are very many services on the Internet these days. As the number of users has been increasing drastically, popular service providers face to request congestion. Therefore, they need to build servers with high scalability. One approach is to prepare multiple servers, and locate these in widely distributed area. Then, the problem moves to how to distribute requests among these servers. Put another way, the problem is how to select the best server out of them. Therefore, we need a server selection mechanism.

Server selection mechanisms used currently impose heavy tasks on users, or have less flexibility. Consequently, we study on flexible server selection mechanism for multiple servers environment. To realize this, we introduce "policy-added DNS server" attached to a Domain Name System(DNS)[1][2] server. It can coordinate various server selection policies. It works as a proxy server of DNS, and imposes no need of modifying working DNS network.

The rest of this paper is organized as follows. Section 2 overviews the server selection mechanisms. We show our policy-added server selection mechanism in Section 3. We evaluate our mechanism in Section 4. Section 5 overviews related works. Section 6 is a conclusion.

II. Server Selection

We need server selection mechanisms as mentioned above. In this section, we clarify requirements of server selection mechanisms. Then, we study about existing widely used mechanisms.

This research was supported by the JSPS-RFTF-96P00603.

A. Requirements

We consider that there are six requirements of server selection mechanisms.

1. Transparency to users
2. Service independency
3. Implementation independency
4. Scalability
5. Flexibility of server selection policy
6. Service provider independency

A.1 Transparency to users

Usually, users want to know only how to use a service. They do not want to care about a server selection mechanism. Therefore, the mechanism has to transparent to users.

A.2 Service independency

There are many services that will use a server selection mechanism, for example, WWW, FTP, IRC, NTP and so on. Therefore, the mechanism has to be independent of services.

A.3 Implementation independency

Almost all services have multiple implementation of server software and client software. For instance, there are two major WWW browser implementations, Netscape Navigator and Internet Explorer. And there are many other implementations, lynx, mozilla, arena, and so on. Therefore, the mechanism has to be independent of implementation.

A.4 Scalability

As mentioned in Section I, we need server selection mechanisms to build servers with high scalability. If it does not have scalability, all the system cannot be scalable.

A.5 Flexibility of server selection policy

Server selection will be done under some policies. The policy may differ to each other. Therefore, the mechanism must have a flexibility of server selection policy.

A.6 Service provider independency

This item has relevance to the above item. Service providers usually make a server selection policy for their services. However, occasionally, user side network administrators can make their own policy. For instance, their network has two uplinks to the Internet. Then, he/she wants to distribute network traffic both two uplinks. In such case, they make some server selection policy to implement them.

Therefore, it is desirable that server selection can perform both of the service provider side and user side. We call it "service provider independency".

B. Existing mechanisms

There are two widely used server selection mechanisms. We estimate them about above six requirements. In this section, we use RingServer[8] as an example. This is one of the biggest file archive servers in Japan. It has 18 mirror servers in widely distributed networks now.

B.1 User selection

In this mechanism, service provider gives different hostnames to their multiple servers. Then, users select one of them by them selves. For example, each server has its own name, `core.ring.gr.jp`, `crl.ring.gr.jp`, `toyama-ix.ring.gr.jp` and so on.

In this mechanism, a user, or a client software, is assumed to have enough knowledge to select a best server. This assumption is unrealistic. Users may be imposed an extra effort. This mechanism is not transparent to users.

However, it does not depend on service, implementation, nor service provider.

It lacks scalability. Because, when service provider increase their servers, users have to know about this new server. It is very difficult to inform users about this change. Therefore, increasing servers is difficult.

Server selection was done by each user. Therefore, it is impossible to install some server selection policy.

We consider that biggest disadvantage of this mechanism is user transparency. It lacks user transparency. We conclude that this mechanism is not appropriate.

B.2 DNS round robin

The latest DNS server has a capability of round robin selection. If service provider register multiple IP addresses for a server, then DNS server automatically selects one of them by round robin algorithm. For example, `ftp.ring.gr.jp` has 18 different IP addresses. When a user tries to use `ftp.ring.gr.jp`, DNS selects one of the 18 hosts.

DNS is used by client software and server software automatically. Therefore, users do not care about it. It is transparent to service and implementation. DNS works distributedly; therefore, it will not become a bottleneck.

However, round robin algorithm is too simple for effective server selection. It selects one of multiple serves one by one. If one of server resides near a client, it may select the other one. Therefore, the client may access a faraway server. Existing DNS server can use only the round robin algorithm.

We consider that this mechanism has many advantages. However, it lacks flexibility of server selection policy.

III. Policy-added Server Selection

We propose new server selection mechanism[3][4]. Our basic idea is simple. As mentioned in Section II-B.2, DNS has many advantages as a server selection mechanism. However, it lacks flexibility of server selection policy. Therefore we add the flexibility to DNS.

We introduce "query preprocessor" attached to existing DNS network. Many server selection policies can integrate into the "query preprocessor". Therefore, our mechanism can adopt many situations. We show these policies at Section III-A. Our "query preprocessor" works as a proxy DNS server. We show a detail of it at Section III-B.

"Query preprocessor" can perform server selection at both server side and client side. Many of other server selection mechanisms can perform on server side only. Therefore, if service providers prepare multiple servers and they do not prepare any sophisticated server selection mechanisms, users cannot enjoy these multiple servers enough. However, at this situation, users can use "query preprocessor" at client side. Then, they enjoy multiple servers.

When selection is done on client side, a problem is how to collect candidates. Because how to know which hosts are multiplied servers is not obvious. We show the solution at Section III-C.

Some policies use network status to evaluate candidates. Therefore, our mechanism contains "network status observer". We show a detail of it at SectionIII-D.

A. Server selection policy

There are various policies for server selection. Any of them can integrate into the "quer preprocessor". For example, we implemented, and/or will implement, below policies.

- Selects a server that have a shortest round trip time between a client and a server.
- Selects a server that have a highest throughput.
- Selects a server that have a lowest CPU load.
- Selects a server that have shortest routing path between a client and a server.
- Selects a server by administrator's decision.

Some of these are suitable for server side server selection, and some of these are suitable for client side server selection.

In this paper we focus on client side server selection. Because, there are many service providers that they prepare multiple servers but they do not prepare any sophisticated server selection mechanism. We consider that we need client side server selection mechanism. For example, round trip time or throughput between client side to all servers can measure from client side by using ICMP ECHO packet or HTTP GET command. Therefore, the server selection policy that use these types of information are suitable for client side server selection. In these policies, we take no notice of information that is hard to collect from client side, for example CPU load of servers.

B. Separate preprocessor

Now we introduce a "query preprocessor". An existing DNS server contains both of the hostname database and query processor. When it receives a request, it searches hostname database and returns an answer.

We separate these two tasks. "Query preprocessor" not only answers to requests, but also selects an IP address for the hostname using some server selection policies. It works as a proxy server, and imposes no need of modifying working DNS network.

Clients change only one configuration. They switch to use "query preprocessor" instead of existing DNS server. If clients use DHCP or PPP, the DNS server should be configured automatically. Therefore, in these cases, users do not need to be aware of this configuration change. And, many of other situation, network administrators configure which DNS server client machines use. Therefore, changing DNS server is negligible for transparency to users.

C. Collect candidates

When running "query preprocessor" on server side, collecting candidates is easy work. However running on client side, it is not easy. We have to collect candidates by some way. Actual candidates are IP addresses of the corresponding servers.

We collect candidates from DNS in the first place. There are four cases that how multiple servers' hostnames and IP addresses are stored in DNS.

1. Single hostname and single IP address
2. Single hostname and multiple IP addresses
3. Multiple hostnames and single IP address
4. Multiple hostnames and multiple IP addresses

In the case 1, we can collect only one IP address and one hostname. The only candidate is this IP address. Therefore, we cannot perform server selecting.

In the case 2, we can collect all the multiple IP addresses from DNS. We use these as candidates.

In the case 3, we can collect only one IP address like case 1. However, it is unusual that someone use this for load balancing. Frequently, it is used for virtual hosting.

In the case 4, we want to collect all the multiple IP addresses. To do this, we have to collect all the hostnames at first. However, unfortunately, there are no general way to do so. Collecting these hostnames automatically is impossible. Therefore, we collect these by hand. We show a detail in next section.

C.1 Multiple hostnames and multiple IP addresses

It is common to prepare multiple servers for load balancing. Giving them individual hostnames is not special, in these days. There are no general way to know these hostnames automatically. We must list up these hostnames by hand. Then, resolve these hostnames to IP addresses. We use these IP addresses as candidates.

We use all these IP addresses as candidates to resolve all these hostnames. "Query preprocessor" may answer an IP address that a requested hostname does not have.

Suppose for examples that there are two IRC servers "irc.foo.org" and "irc.bar.net", and they are connected each other. They provide same IRC space. The IP address of irc.foo.org is 133.5.0.1, and irc.bar.net is 192.50.13.250. When a user wants to connect irc.foor.org, his client software send a request to "query preprocessor". The request wants to resolve irc.foo.org. The query preprocessor uses either 133.5.0.1 or 192.50.13.250 as candidates. It can answer 192.50.13.250. Therefore, the user can connect irc.bar.net instead of irc.foo.org. However, these two IRC servers prepare same IRC space, the user can use this IRC server with no problem.

D. Network status observer

"Query preprocessor" communicates with "network status observer" to collect network status. The observer probes network status periodically. There are two observation methods. One is active observation, and the other is passive observation.

D.1 Active observation

In this method, the observer collects network status actively, for example:

- observe by probe packets
- observe by using service

The observer can send some probe packets, for example, ICMP echo request, and measures their round trip time. It uses the results to collect network status. In other case, it can send some real requests, for example, HTTP GET, to a server and measures through puts.

If a service that a client wants to use is known beforehand, the observer can use this real service to probe. For example, the observer may request WWW servers to transfer some contents and measure its transfer time. However, there are no general method to know the service beforehand. Therefore, we assume a service based on a hostname alias defined in RFC2219[6]. For example, if a client requests resolving a hostname that begins with "WWW", then "query preprocessor" assumes that the client wants to use WWW(http)service. Therefore, the observer can try to use http packets to collect network status.

One of weak point of the active observation method is that observer generates uselessly traffics. If we use real service request, traffic and server load is not negligible. On the other hand, probe packets, for example ICMP echo, are often filtered out by firewalls. Then, probe packets cannot reach servers, and the observer cannot collect network status. While service packets are rarely filtered out, and the observer can collect network status.

D.2 Passive observation

In this method, the observer passively observes network status. Advantage of this method is small network overhead. Examples of passive observation targets are listed below.

- traffic of service
- routing information

When observing these traffic of service, we can measure throughput between a client and a server. This measurement does not need useless traffic. And it is a part of real traffic, therefore it would be accurate information.

Routing information contains information to select a network route. The observer can use this information as a network status. For example, BGP[7] packets contain AS path list. A observer collects all AS path length for all of multiple servers.

IV. Consideration

A. *Implementation of prototype system*

We implement a prototype system. We call this "Tenbin[1]". Tenbin contains both of the query preprocessor and network status observer.

Query preprocessor consists of a request receiver, a policy database, a policy decider, and some policy executor. Some of these policy executors communicate with their own network status observer.

We describe how Tenbin works. When the request receiver receives a request from clients to resolve a hostname, it parses the request. If it does not know a query type of the request, Tenbin forwards the request to predefined existing DNS server. If it know the query type, the policy decider selects a server selecting policy for the request. The decider selects a policy executor from the policy database using the requested information.

The request receiver passes the request to a decided policy executor. The executor communicates with its network status observer to obtain network status information if needed. Then, policy executor selects an IP address by his policy.

When there is no specific policy for the request, the request receiver uses a default policy. The current default policy is that simply forwards the received request to predefined existing DNS server.

Finally, the selected IP address is sent back to the requester.

We use an object-oriented script language Ruby[5] for coding Tenbin.

B. *Evaluation using prototype system*

We examined Tenbin's capabilities and performance on some networks.

We examined transition of server selection results. In this examination, we used `ftp.ring.gr.jp` for server selection targets. The `ftp.ring.gr.jp` has 14 IP addresses[2], and they locate at 14 different networks.

In this examination, our server selection policy is "shortest round trip time". We measured round trip time between Tenbin and 14 servers every 30 minutes by sending ICMP ECHO packets.

[1]Tenbin stands for "Tenbin is Experimental Name server for load Balanced INternet"

[2]We described that RingServer has 18 mirror servers in Section II-B. However it had 14 mirror servers when we made this experimentation.

TABLE I
Selected times at Kyushu Univ.

hostname	count	ratio(%)
ring.nacsis.ac.jp.	1486	51.1
ring.ocn.ad.jp.	602	20.7
ring.etl.go.jp.	223	7.68
ring.aist.go.jp.	150	5.16
ring.asahi-net.or.jp.	130	4.47
ring.so-net.ne.jp.	124	4.27
ring.crl.go.jp.	74	2.54
ring.jah.ne.jp.	55	1.89
ring.shibaura-it.ac.jp.	43	1.48
ring.omp.ad.jp.	9	0.31
ring.ip-kyoto.ad.jp.	9	0.31
ring.htcn.ne.jp	1	0.03

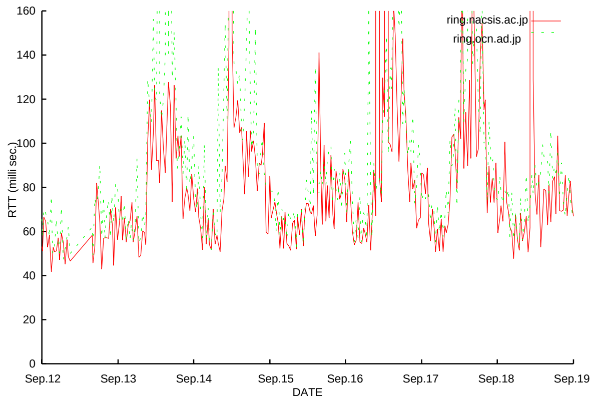

Fig. 1. round trip times

We show the result in Table I. Tenbin selects one of 14 servers that has a shortes round trip time. This table show the selected count of every 14 servers.

We plot transition of two RTTs in Fig.1.

B.1 Result difference by location

Next, we studied difference of selected host by location of Tenbin. We locate Tenbin some locations. We show the result of selected hosts at Nagasaki University(Table II), WIDE Project FUKUOKA NOC(Table III) and WIDE Project NEZU NOC(Table IV). These tables show only top three hosts of selected hosts.

We see from Table I and Table II that these two results are similar. From a network topology standpoint of view, Kyushu University and Nagasaki University locate at same AS(Autonomous System). We consider that this results similarity is caused by network topology similarity.

We also see from these two tables that a tendency of selected hosts show concentration.

From a view point of geographical location, Kyushu University and WIDE Project FUKUOKA NOC locate in a same campus. However, from a network topology standpoint of view, they are at different ASs. We see from Table I and Table III that these two results are very different. We consider that this results difference is caused by net-

TABLE II
Selected times at Nagasaki Univ.

hostname	ratio(%)
ring.nacsis.ac.jp.	88.2
ring.ocn.ad.jp.	10.0
ring.shibaura-it.ac.jp.	0.53

TABLE III
Selected times at WIDE FUKUOKA NOC

hostname	ratio(%)
ring.shibaura-it.ac.jp.	20.0
ring.asahi-net.or.jp.	16.4
ring.omp.ad.jp.	10.6
ring.ocn.ad.jp.	10.6

TABLE IV
Selected times at WIDE NEZU NOC

hostname	ratio(%)
ring.asahi-net.or.jp.	58.0
ring.nacsis.ac.jp.	29.1
ring.ocn.ad.jp.	12.2

TABLE V
Selected times at WIDE FUKUOKA NOC (before 1999/10/14 10:00)

hostname	ratio(%)
ring.omp.ad.jp.	58.3
ring.ip-kyoto.ad.jp.	22.1
ring.htcn.ne.jp.	11.0

work topology difference.

From a view point of network topology, WIDE Project FUKUOKA NOC and WIDE Project NEZU NOC locate at the same AS. However, these two results show different tendency. We think this difference comes with difference of connectivity between other ASs.

We see from Table III that the tendency of selected hosts does not show concentration. This comes from a change of network topology.

For example, topology between WIDE Backbone and NSPIXP3 [3] changed on October 14. Table V is a result of selected hosts before October 14. It shows a tendency.

These results show that Tenbin selects the nearest server automatically. Therefore, users can use the best server automatically.

B.2 Overheads

We examined an overhead of Tenbin. When Tenbin forwards a request to the existing DNS server, average overhead time of Tenbin was 9.1 ms. We consider that this overhead is negligible.

C. Limitation

C.1 Specific hosts

There can be a case that users or administrators want to use a specific server out of multiple servers. Our DNS server may select another host. There are two solutions for this problem.

1. Use an IP address directly instead of the hostname. It is difficult to use the IP address for ordinary users. However, the case which users want to use a specific server must be a special case, administrative work for example. Therefore, this is practical.
2. Giving individual hostnames one by one. Users can access a specific host using its specific hostname. However, if users use this specific host to receive ordinary service, load balancing does not function.

[3]one of major Internet eXchange in Japan

C.2 Selection using only hostname

To use a service, information other than hostname is required: the port number of TCP/UDP, path name of contents for example. If these are different between multiplied servers, our mechanisms is not usable.

For instance, some IRC servers provide services on multiple port for large number of clients. However, not all servers provide such service. Therefore, we cannot make all these IRC servers as candidates.

V. Related works

Smart Clients[9] is a one of the client side selection mechanism. Smart Clients use Java applet to deliver server status to clients. This system has to modify both server and client software. It depends on implementation.

SWEB[10] is a mixture of round robin DNS and server side selection mechanism. In this system, server selection was done by two steps. First step, DNS server selects a first server by round robin. Second step, the selected first server selects a final server out of multiple servers and redirect request to the final server. This system can evaluate server status on the first server. However, to evaluate server status, servers have to modify to collect server status. It depends on server implementation.

The system by Shigechika et al. [11] uses network selection model. It assigns a single IP address to multiple servers, and lets routing mechanism select an appropriate server. It lacks flexibility of server selection policy.

The weighted round robin algorithm[12] is a simple extension of round robin DNS. It assigns multiple IP addresses to a server according to servers' capacity. It can evaluate servers' capacity and its load occasionally. However, it lacks flexibility of server selection policy.

Approach of Cluster DNS[13] is similar to ours. It adds server selection mechanism to DNS. However it, and all of above except round robin DNS, assume that they, at least a part of them, run on server side. Therefore, it lacks service provider independency.

Distributed Director[14] is a product for load balancing. It uses the applicabity gateway selection model and the meta server selection model. It uses routing information to select a server. It depends on router implementation.

VI. Conclusion

In this paper we clarify requirements of server selection mechanism. Then, we propose a design for flexiblely server selection mechanism using DNS server that embeds various server selection policy. This system design has wide applicability, and runs distributedly. We implement a prototype system, and evaluate it.

The future direction of this study will include:

- Detailed result analysis
- Estimate using widely distributed service
- Design and implement various policy executor.
- Develop better policy selection mechanisms

Current Tenbin implements a few policy executor. We must implement other policy executors, and evaluate these. We plan to implement some policy executor. One use routing information. It communicate with external routing server to obtain routing information. In the first step, we use BGP AS path length as routing information. Current all policy exector embed its network status observer. It will be first network status observer that does not embed network status observer. We have to evaluate overhead of communication with external network status observer.

The other policy executor select highest through put server. Its network status observer will send HTTP GET request to candidate servers. We will compare the results of throughput policy and round trip policy.

Acknowledgments

The authors would like to thank Prof. Hiroshi Esaki(Tokyo University), and Mr. Ikuo Nakagawa (Intec Web and Genome Informatics Corporation) for providing host for running Tenbin at WIDE Project NEZU NOC. We also thank the members of RingServer Project for helpful discussions.

References

[1] P. Mockapetris, "DOMAIN NAMES – CONCEPTS AND FACILITIES", RFC 1034, Nov. 1987

[2] P. Mockapetris, "DOMAIN NAMES – IMPLEMENTATION AND SPECIFICATION", RFC 1035, Nov. 1987

[3] T. Shimokawa, N. Yoshida, K. Ushijima, "Flexible Load Balancing Mechanism using Name Server," Internet Conference '99, Dec. 1999 (in Japanese)

[4] T. Shimokawa, N. Yoshida, K. Ushijima, "Flexible Server Selection Using DNS," International Workshop on Internet 2000, Apr. 2000

[5] Y. Matsumoto, "Ruby the Object-Oriented Script Language, " http://www.ruby-lang.org/

[6] M. Hamilton, R. Wright, "Use of DNS Aliases for Network Services," RFC 2219, Oct. 1997

[7] Y. Rekhter, T. Li, "A Border Gateway Protocol 4 (BGP-4)," RFC1771, Mar. 1995

[8] Ring Server Project, http://www.ring.gr.jp/

[9] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, D. Culler. *Using Smart Clients to Build Scalable Services.*, Usenix '97.

[10] D. Andresen, T. Yang, V. Holmedahl and O. Ibarra., *SWEB: Towards a Scalable WWW Server on MultiComputers*, Proceedings of the 10th International Parallel Processing Symposium, April, 1996.

[11] N. Shigechika, O. Nakamura, N. Sasakawa, J. Murai, "Network and Information Providing System for Nagano Olympic" Journal of Information Processing Society of Japan Vol.39 No.2, 1998(in Japanese)

[12] T. Baba, S. Yamaguchi, "A DNS based implementation on widely load balancing mechanism, " IPSJ SIG Notes, 98-DSM-9, pp.37-42, May 1998 (in Japanese)

[13] V. Cardellini, M. Colajanni, P.S. Yu, *DNS dispatching algorithms with state estimators for scalable Web-server clusters*, World Wide Web Journal, Baltzer Science, vol. 2, no. 3, July 1999, pp. 101-113.

[14] CISCO Systems Inc. DistributedDirector, http://www.cisco.com/warp/public/cc/cisco/mkt/scale/distr/index.shtml