

On the Stability of Server Selection Algorithms against Network Fluctuations

Toshiyuki MIYACHI¹, Kenjiro CHO^{2,1}, and Yoichi SHINODA³

¹ School of Information Science, Japan Advanced Institute of Science and Technology,
Nomi, Ishikawa 932-1292 Japan

² Internet Initiative Japan Inc. Research Laboratory,
Chiyoda, Tokyo 101-0051 Japan

³ Center for Information Science, Japan Advanced Institute of Science and Technology,
Nomi, Ishikawa 932-1292 Japan

Abstract. When a set of servers are available for a certain client-server style service, a client selects one of the servers using some server selection algorithm. The best-server algorithm in which a client selects the best one among the available servers by some metric is widely used for performance. However, when a network fluctuation occurs, the best-server algorithm often causes a sudden shift of the server load and could amplify the fluctuation. Reciprocal algorithms in which a client selects a server with a probability reciprocal to some metric are more stable than the best-server algorithm in the face of network fluctuations but their performance is not satisfactory.

In order to investigate trade-offs between the stability and the performance in server selection algorithms, we evaluate the existing algorithms by simulation and visualize the results to capture the stability of the server load.

From the simulation results, we found that the performance problem of the reciprocal algorithms lies in selecting high-cost servers with a non-negligible probability. Therefore, we propose a 2-step server selection scheme in which a client selects a working-set out of available servers for efficiency, and then, probabilistically selects one in the working-set for resiliency. We evaluate the proposed algorithm through simulation and show that our method is adaptive to environments, easy to load-balance, scalable, and efficient.

1 Introduction

There are many client-server style services on the Internet. When a set of servers are available for a certain service, a client selects one of the servers using some server selection algorithm.

In many cases, the best-server algorithm in which a client selects the best one among the available servers by some metric is widely used. However, the best-server algorithm distributes the load unevenly to different servers so that it often places high loads on a few servers while the rest of the servers are lightly loaded.

Skewed load distribution itself is not a problem because it also allows to solve the high load of a server by either adding another server near the congestion point or replacing the server with more powerful one.

However, the best-server algorithm has another problem; a network fluctuation can trigger clients to shift to another server at a time, which in turn could lead to further network fluctuations. The best-server algorithm often causes a sudden shift of the server load in the face of network fluctuations, and it is difficult to manage by server placement.

On the other hand, the uniform algorithm randomly selects servers and evenly distributes the load among available servers. But the performance of the uniform algorithm is much worse than the best-server algorithm.

There are other algorithms which falls in between the best-server algorithm and the uniform algorithm but their performance is not as good as the best-server algorithm.

In this paper, we first evaluate the existing server selection algorithms in terms of stability and performance, and then, propose a new algorithm which is resilient to network fluctuations and efficient in performance.

2 Existing server selection algorithms

There are several algorithms to select a server to use among a set of available servers. Here, we describe three typical server selection algorithms.

2.1 Best-server algorithm

The best-server algorithm measures a metric such as hop count and round-trip time(rtt) to servers and selects one as the best server.

The performance of the best-server algorithm is optimal in static environment since each client selects the best performing server for receiving the service.

However, wide-area networks are never static and conditions keep changing. With the best-server algorithm, clients' requests often concentrate on a small number of servers with lower cost. When a network fluctuation occurs and metrics to these servers are changed, the clients reselect the best server. Many clients tend to select the same server, which often results in amplifying network fluctuations.

2.2 Uniform algorithm

The uniform algorithm selects a server out of available servers randomly, regardless of the metric.

The load of servers also becomes uniform. Therefore, this approach does not have the problem of skewed server load.

However, the performance of the uniform algorithm is poor because it does not take the access cost into consideration. Server placement also becomes difficult with the uniform algorithm since the performance is usually dominated by the worst performing server located far away.

2.3 Reciprocal algorithm

The reciprocal algorithm selects a server with the probability proportional to the reciprocal of the access cost. There are several reciprocal functions that can be used.

The loads of servers are distributed to some extent as each client uses all the servers with certain probabilities.

On the other hand, the performance is much worse than the best-server algorithm because servers with high access costs are probabilistically used.

2.4 A case study: DNS

DNS [1][2] is a service to translates host names to IP addresses. It is a distributed database and has a hierarchical tree structure. When a client sends a query to the local DNS server to resolve a host name, the local DNS server resolves the name by traversing the name hierarchy and returns the answer to the client. If there are multiple authoritative servers available to resolve a name, the local DNS server selects one out of the authoritative servers. It is a common practice to have a few authoritative name servers for both availability and performance. The characteristic of the server selection in DNS is that the number of authoritative servers for a name is relatively small, usually from 2 to 13.

There are several DNS implementations employing different server selection algorithms.

BIND The Berkeley Internet Name Domain system(BIND) [3] is the most widely used implementation. BIND (both version 8 and 9) maintains smoothed rtt(srtt) as a server metric, and the srtt is decayed while the server is not used. It is a variant of the reciprocal server algorithm.

DJBDNS and Microsoft Windows Internet Server

DJBDNS [4] and Windows Internet Server employ the uniform algorithm.

Although the heuristics employed by BIND work well for many cases, there are rooms to improve. The load distribution of 2 nearby servers is too skewed, and it is not suitable for applications requiring a large number of servers since it needs to keep the states of all servers [5].

3 Evaluation of the existing server selection algorithms with simulation

In order to evaluate the behavior of the existing server selection algorithms, we use simulations in which network fluctuations occur in a complex topology.

To create unbalanced server load, the simulation topology is generated by a topology generator based on a scale-free network model [6]. We adopt the following rules in the topology generator.

- The first node is placed without any edge.

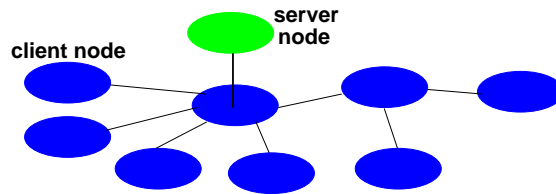


Fig. 1. Concept of the simulation topology

- Thereafter, a node is placed by choosing an existing node as a peer node to connect to with the probability proportional to the number of edges. This rule implements a scale-free model.
- After placing every ten nodes, a new server is placed at the node with the largest number of edges. (If the node already has a server, the next one is selected.)
- Whenever the number of clients exceeds 20 for a server, the server is split into 2 servers. (When making the topology, we use the best-server algorithm) When a server splits, the node connected with the server also splits. The edges of the original node are randomly re-assigned to the 2 new nodes.
- After placing every hundred nodes, a new edge is randomly created by choosing 2 nodes with the probability proportional to the number of edges. If the 2 nodes already have a direct edge, 2 nodes are re-selected. This rule is to create a loop in the topology.

Figure 1 shows the concept of the topology.

The link cost between nodes is initialized to 10 and the link cost between a server and its bounded node is 15.

To construct a large-scale topology, we place 500 nodes for the simulation using the above rules. Due to the server-split rule, the final topology has 510 nodes and 60 servers.

In order to simulate network fluctuations, the following 2 steps are performed 50 times in the simulation.

1. Choose a server randomly, and change the cost between the server and the bounded node to a random value in the range from 1 to 40.
2. Restore the cost to the original value.

We designed and implemented a simulator with these rules since there is no appropriate simulator for evaluating server selection algorithms in a complex topology. The simulator is written in the C language and consists of about 3000 lines of the code.

To visually understand the bias in server load, we draw the topology graphs with colored server loads.

3.1 Visualization of server loads

We made topology graphs with colored server loads, in order to easily understand the impact of the server selection algorithms to server loads. In particular, we are interested

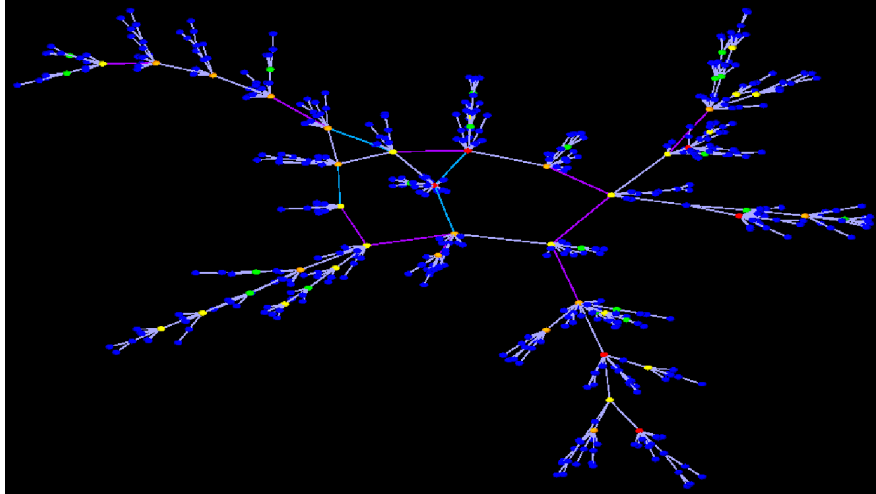


Fig. 2. The simulation topology

in observing the shift of the server loads when a network fluctuation occurs or when a new server is placed to distribute server loads.

We use Tulip [7], a graph visualization tool, for graph layout and rendering. Tulip comes with several layout algorithms including 3D layouts.

Figure 2 shows the simulation topology. In this topology, blue nodes show clients, and server nodes have other colors varying with the load of servers. In the topology making phase, each node selects servers 100 times whenever a new client is added. The server load is shown as the number of clients for each server. A green server has less than 600 clients, a yellow one has 600 to 1099 clients, an orange one has 1000 to 1599 clients, and a red one has more than 1600 clients.

With these topology graphs, it becomes intuitive to observe server loads by color as well as the movement of the server loads.

3.2 Best-server algorithm

Figure 3 and 4 shows the results of the simulation. Figure 3 shows the time-series average and maximum cost. In each step, each client selects a server 100 times. The average cost from each client to the selected servers is computed in each step, and then, the average cost over all the clients is computed. The maximum is the one with the largest average cost in the step. The average cost from a client to a server of the best-server algorithm is about 22. This is optimal in the simulation where the link cost between nodes is 10 and the link cost between a server and a node is 15 that is fluctuated between 1 to 40. The maximum cost from a client to a server in each step is between 50 and 60, which is the lower bound of the maximum cost.

Figure 4 shows the time-series server loads. It can be observed that, when there is an upward spike, there is a downward spike of the same size in the same step. For example,

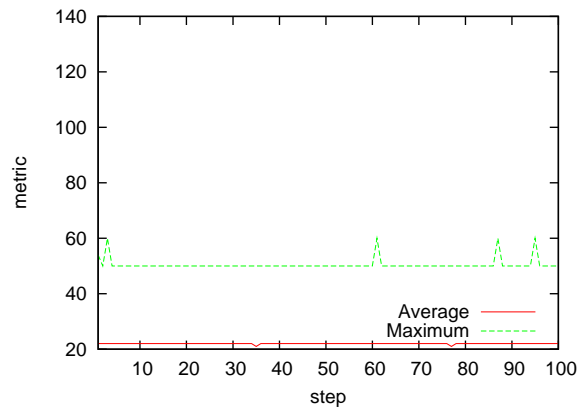


Fig. 3. Average and maximum costs of best-server algorithm. The average cost is about 22 that is optimal in our simulation.

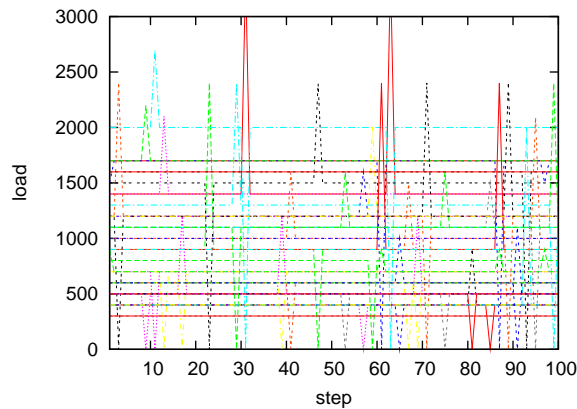


Fig. 4. Server load of best-server algorithm. When a network fluctuation occurs, the load of an affected server shifts to another server.

server 5's load becomes 2000 to 0 at step 63 because the cost between server 5 and the bounded node is increased from 10 to 38, and server 1's load increases from 1400 to 3400. That is, all the clients of server 5 move to server 1. It illustrates the effect of a network fluctuation to the server load.

For the same reason, it is easy to predict a similar load shift when a new server is placed and becomes the best server for many clients. Therefore, it is difficult to control the distribution of server load by placing new servers with the best-server algorithm.

These results illustrate the instability of the best-server algorithm that we have observed on the actual Internet.

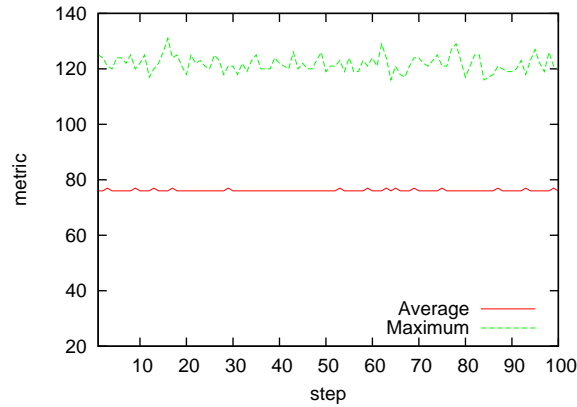


Fig. 5. Average and maximum costs of uniform algorithm. The average cost of the uniform algorithm is about 3.5 times higher than that of the best-server algorithm.

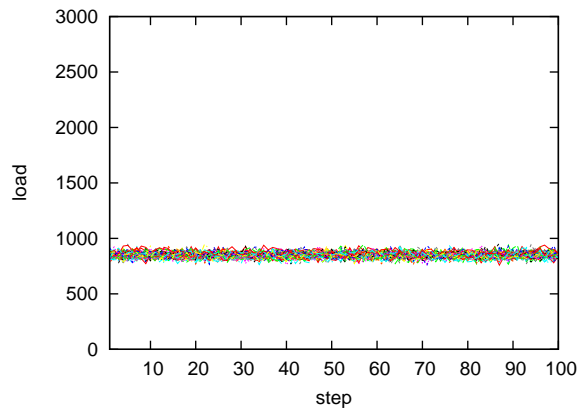


Fig. 6. Server load of uniform algorithm. All the servers have similar loads, and they are not affected by network fluctuations.

3.3 Uniform algorithm

Figure 5 and 6 show the simulation results of the uniform algorithm. The average cost is about 77 that is 3.5 times higher than the value of the best-server algorithm. It shows poor performance of the uniform algorithm.

The servers' loads are almost flat. It means there are small influences by network fluctuations.

These results are also what we expected.

3.4 Reciprocal algorithm

In this simulation, we evaluate two reciprocal functions, $1/cost$ and $1/cost^2$.

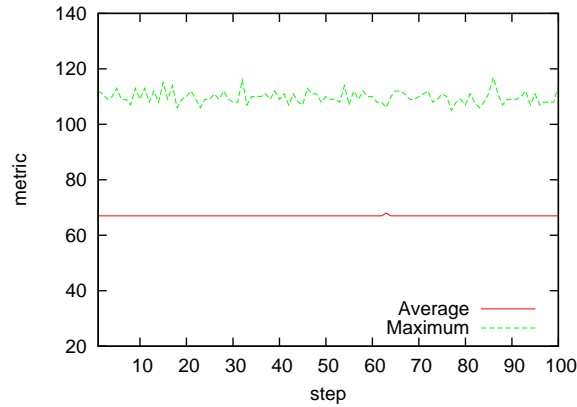


Fig. 7. Average and maximum costs of reciprocal algorithm($1/cost$). The average cost of the reciprocal algorithm($1/cost$) is 3 times higher than the best-server algorithm.

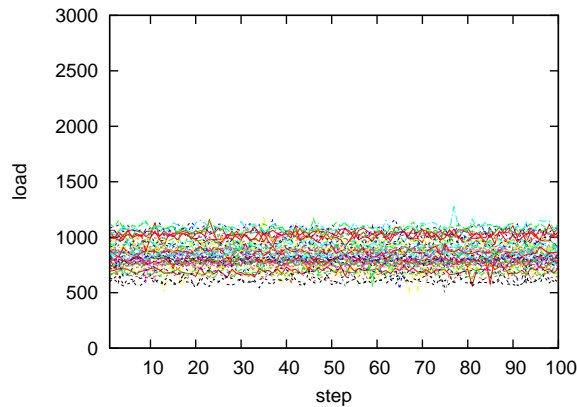


Fig. 8. Server load of reciprocal algorithm($1/cost$). The loads are relatively stable but the range of the loads is 2-3 times wider than that of the uniform selection

using function: $1/cost$ The simulation results are shown in Figure 7 and 8. The average cost is 67.5 that is 3 times higher than the value of the best-server algorithm and the performance improvement is only 14% compared with the uniform algorithm.

As for the stability, when a network fluctuation occurs, the influence on server loads is larger than the uniform algorithm but it is much smaller than the best-server algorithm.

using function: $1/cost^2$ The behavior of the reciprocal algorithm with $1/cost^2$ is shown in Figure 9 and 10.

The behavior is closer to the best-server algorithm because the probability of using a server with a lower cost is much higher than the $1/cost$ function.

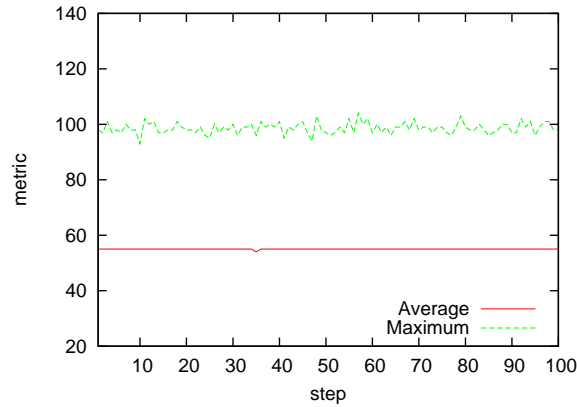


Fig. 9. Average and maximum costs of reciprocal algorithm($1/cost^2$). The average cost of the reciprocal($1/cost^2$) is 2.5 times higher than that of the best-server algorithm, and better than the reciprocal($1/cost$).

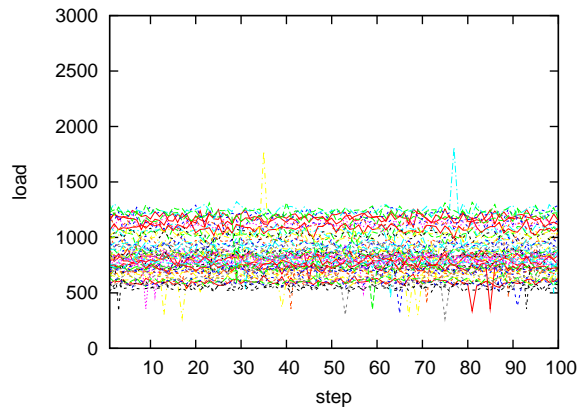


Fig. 10. Server load of reciprocal algorithm($1/cost^2$). The server load of the reciprocal algorithm($1/cost^2$) is not so skewed. The range of the server loads is wider than $1/cost$. The server load shifted by network fluctuations is absorbed by multiple servers.

The average cost is 55.5 that is 2.5 times higher than the value of the best-server algorithm.

The influence on server loads by a network fluctuation is smaller than the best-server algorithm, but there are some spikes in Figure 10. When a network fluctuation occurs near a server with high load, its load is distributed to multiple servers. Therefore, the probability of amplifying a fluctuation is smaller than the best-server algorithm.

3.5 Summary of the simulation

We summarize the results of the simulation.

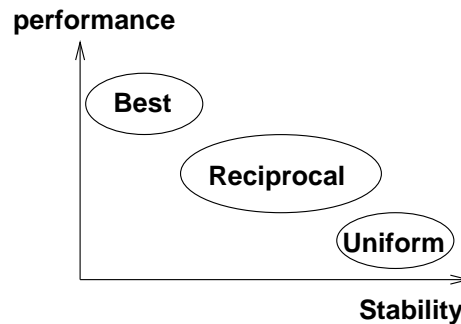


Fig. 11. The relationship of performance and stability with the 3 algorithms

The best-server algorithm always uses the best server. The uniform algorithm chooses one server regardless of the performance. The reciprocal algorithm changes the probability of selecting a server with server's cost.

Figure 11 shows the relationship of the performance and the stability of the 3 algorithms.

The best-server selection selects the best server to use, therefore the performance is always the best. However, when a network fluctuation occurs, it often amplifies the fluctuation. And it is difficult to distribute server loads by placing new servers as each client always selects the best server.

The performance of the uniform selection is the lowest among the algorithms examined but it is the most stable. However, the performance can be hardly improved even if a new server is placed at clients' concentration point.

The behavior of reciprocal algorithm is positioned in the middle of that of the best-server algorithm and that of the uniform algorithm. It selects a server with a high cost to some extent, and thus, the performance is not satisfactory.

Considering these results, we can avoid amplifying a network fluctuation if we probabilistically select one server out of multiple servers. However, the performance of the existing probabilistic algorithms is not good enough.

In the next section, we discuss this problem and propose a new algorithm.

4 Two-step server selection algorithm

The existing reciprocal algorithms do not have satisfactory performance, because they probabilistically use those servers which have fairly high costs. The target server should be selected from servers with small costs but a single server should not be selected to avoid amplifying a network fluctuation.

The problem lies in using a single algorithm for all available servers. A single probabilistic algorithm includes 2 different functions; selecting good servers and load-balancing among good servers. There are suitable algorithms for each function, but the existing algorithms relies on a single algorithm for both functions. Thus, the performance is inevitable to deteriorate, especially when poorly performing servers exist.

Thus, we propose a 2-step server selection algorithm that separates selecting good servers from load-balancing. In the first step, we select a small number of good servers as a working-set. In the second step, we probabilistically select a server to use out of the working-set, which allows us to use only good performing servers. This 2-step selection offers scalability, flexibility and efficiency.

We describe the details of our algorithm in this section.

4.1 Working-set Selection Algorithm

The objective of the working-set selection algorithm is to efficiently select a small set of good performing servers out of a large number of available servers.

Our insight is that the time granularity of the working-set selection can be much coarser than that of the target server selection. That is, we can reduce the cost of the working-set selection by increasing the probe interval. It is desirable to frequently probe all servers to quickly adapt to environmental changes. However, it is costly to probe the access costs of all servers, especially when the number of servers is large.

Our algorithm probes better performing servers more frequently since better performing servers are more likely to be selected for the working-set. The algorithm reduces the probe cost by increasing the probe interval of poorly performing servers. The disadvantage of this method is that it takes longer to detect sudden improvements of poorly performing servers. However, detecting improvement is less critical than detecting deterioration so that we believe it is a sensible design choice.

Our algorithm sorts servers by their costs, and adjusts the probe interval of each server to a value proportional to the server rank. Therefore, the probe interval becomes a linear function of the rank of the server.

When there are N servers, the rank of the best performing server is 1 and the rank of the worst performing server is N . Let i be the rank of a server. Then, the probe frequency of server i is

$$q(i) = \frac{C}{i}$$

Here C is a scaling factor. As there are N servers, the total number of the probes per unit time $Q(N)$ is

$$Q(N) = \sum_{i=1}^N q(i) = C \sum_{i=1}^N \frac{1}{i}$$

As the total number of server N increases, the total probes $Q(N)$ increases much more slowly so that the algorithm can handle a large number of servers.

The top W servers are selected for the working-set. As an optimization, we can probe servers in the working-set when the servers are actually used so that separate probes may not be necessary for these servers.

4.2 Target Server Selection Algorithm

The objective of the target server selection algorithm is to distribute the load to multiple servers in order to reduce the influence by a network fluctuation. Our algorithm selects one server out of the working-set using a reciprocal algorithm.

The servers in the working-set are the top W servers ranked by the working-set selection. A simple reciprocal algorithm is used to select a target server within the working-set to quickly adapt to cost changes of servers in the working-set.

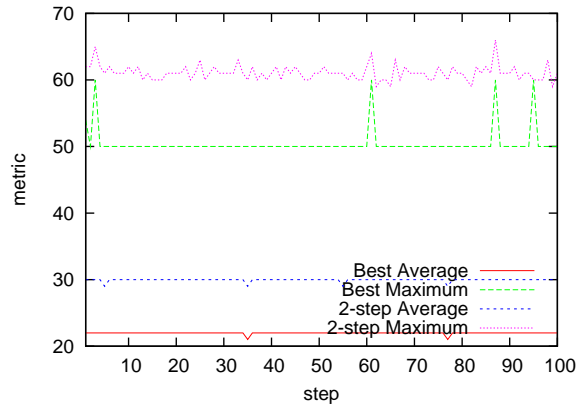


Fig. 12. The maximum and average values of the 2-step algorithm compared with the best-server algorithm. The average of the 2-step algorithm is only 36% higher than that of the best-server algorithm.

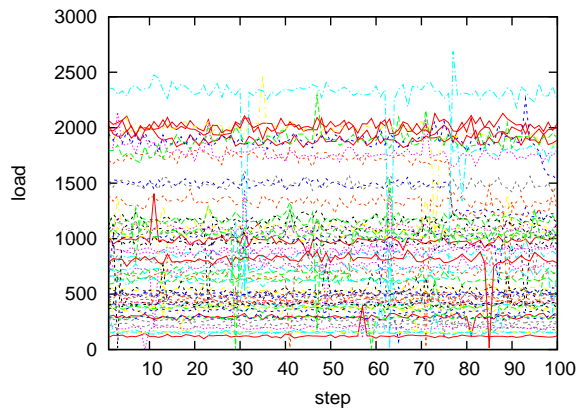


Fig. 13. The server load of the 2-step algorithm. The server loads are skewed as in the best-server algorithm. But when a network fluctuation occurs and the load of a server shifts, its load is distributed to multiple servers. It is more stable than the best-server algorithm.

The target server selection relies on the working-set selection to keep a rough set of best-performing servers. Just a rough set is required because, if the performance of a server in the working-set degrades, this server is excluded by the target server selection in a short-term and eventually taken out of the working-set by the working-set selection.

4.3 Result of the simulation

In the simulation, we use round-trip time (rtt) as a cost metric. Real rtt includes server processing time, network delay and other factors, and represents the required time for a client to actually receive a service. To filter out instantaneous outliers in rtt, we use

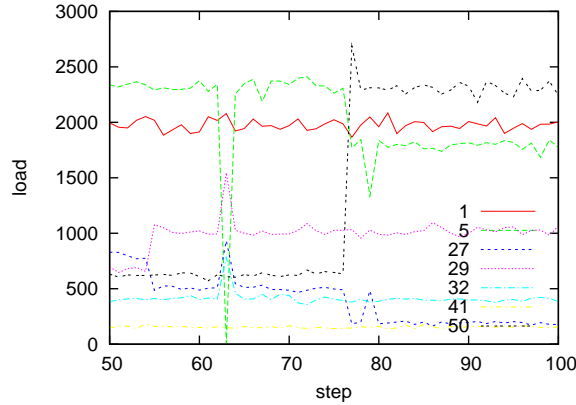


Fig. 14. The loads of 7 servers. The graph illustrates that multiple servers absorb the shifted load caused by a network fluctuation.

smoothed rtt ($srtt$). The value of $srtt$ is calculated as Exponentially-Weighted Moving Average (EWMA) by the following formula.

$$srtt = \alpha \times srtt + (1 - \alpha) \times rtt$$

Here, α is the weight of EWMA. Although it is common to use α between 0.7 to 0.8, we use 0.6 in our simulation to quickly adapt to changes.

The size of the working-set can be small in our method since, if the performance of a server degrades, it will be automatically replaced by the next server by the working-set selection algorithm.

We evaluated the 2-step algorithm with $W = 4$ on the same topology used for the other algorithms. The reciprocal algorithm used for selecting a target server is $1/cost$. We chose $1/cost$ instead of $1/cost^2$ to distribute the server load more evenly among the servers in the working-set.

Figure 12 shows the performance of the 2-step algorithm compared with the best-server algorithm. The average cost is about 30. It is only 36% lower than the best-server algorithm, 51.1% better than the reciprocal algorithm using $1/cost$, and 40% better than the other using $1/cost^2$.

Figure 13 shows changes of server loads. When network fluctuations occur, the load of the affected server is distributed to multiple servers.

To illustrate the impact of network fluctuations, Figure 14 shows the load of the 7 servers with the lowest $srtt$ from a certain client at step 74. At 63 step, the load of server 5 decreased sharply because rtt between server 5 and the bounded node was changed from 10 to 38. By this fluctuation, clients connected to server 5 moved to other servers. However, the load of the other servers did not change as much as server 5 because the load was distributed to multiple servers.

At step 77, the cost of server 50 is changed to 3 from 10. Server 50 was put into the working-set for many clients. As a result, many clients moved to server 50 from multiple servers and the load of server 50 increased.

Although the skew in the server load distribution is observed, it is not so large and is useful to control the load by server placement. It is difficult to manage large shifts of server load in a small time-scale but skews in a large time-scale allow to manage the server load by placing a new server near a high-load server. The proposed 2-step algorithm can suppress fluctuations of server load in different time scales while maintaining preferential server selection.

5 Validity of simulation

In this section, we discuss the generality of the topology and the fluctuations used in our simulation.

5.1 Topology

The simulation topology is made by the rules described in Section 3. In this topology, a small number of nodes have a large number of edges and become hubs. Many nodes have only one edge and become leaf nodes. Servers are placed at the hubs so that the distance from clients to the nearest server is fairly short on average. We believe that the properties of real network topologies are well reflected into our simulation topology. Although we have not fully investigated into the generality of our simulation topology, it is at least good enough for our evaluation of server selection algorithms.

Our simulation uses 510 clients and 60 servers. The scale of the topology is large enough to capture the complex behavior of a wide-area network.

5.2 Changing metric of servers

To simulate network fluctuations, we changed the cost of edges in the topology in different simulation steps.

In real networks, network fluctuations are caused by various factors such as the load of servers and routers. Our simulation does not take these factors into account but the simulated fluctuations are enough to capture the impact of different types of fluctuations. The simulation results and its visualization allow us to observe and predict the behavior of clients in the face of network fluctuations.

6 Conclusions

The best-server algorithm is widely used because of its simplicity and good performance but it could lead to amplifying network fluctuations.

In this paper, we have evaluated the existing server selection algorithms by simulation, and visualized the server load to capture their behavior in the face of network fluctuations. Then, we have proposed a 2-step algorithm that is adaptive to network fluctuations and still provides the performance comparable with the best-server algorithm.

There are two important properties of a server selection algorithm. One is preferential server selection in which clients prefer good performing servers. From operational

point of view, it allows operators to control the server load distribution by server placement. The other is that a server selection algorithm should not propagate network fluctuations. If the performance of two servers are similar, clients should use both of the servers equally.

Our proposed method improves the performance by separating working-set selection from target server selection. The proposed 2-step algorithm selects a small working-set out of available servers, and probabilistically selects a target server in the working-set.

We showed that the 2-step algorithm improves adaptability to metric changes of servers, load-balancing, scalability, and efficiency.

In future work, we will investigate the behavior of server selection algorithms using different network topologies and service types. We will also investigate the impact of server placement from an operational point of view.

References

1. P.Mockapetris.: Domain names - concepts and facilities. RFC1034, IETF, November 1987.
2. P.Mockapetris.: Domain names - implementation and specification. RFC1035, IETF, November 1987.
3. ISC BIND, <http://www.isc.org/>
4. DJBDNS, <http://www.djbdns.org/>
5. Ryuji Somegawa, Kenjiro Cho, Yuji Sekiya and Suguru Yamaguchi.: The Effects of Server Placement and Server Selection for Internet Services. IEICE Trans. on Commun. Vol.E86-B No.2. February 2003. p.542–551.
6. A.-L. Barabási, R. Albert, and H. Jeong.: Mean-field theory for scale-free random networks *Physica*, 272, 173-187 (1999).
7. Tulip Software, <http://www.tulip-software.org/>