

Achieving Resilient and Efficient Load Balancing in DHT-based P2P Systems

Di Wu, Ye Tian and Kam-Wing Ng
Department of Computer Science & Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{dwu, ytian, kwng}@cse.cuhk.edu.hk

Abstract

In DHT-based P2P systems, the technique of “virtual server” is widely used to achieve load balance. To efficiently handle the workload skewness, “virtual servers” are allowed to migrate between nodes. Among existing migration-based load balancing strategies, there are two main categories: (1) Rendezvous Directory Strategy (RDS) and (2) Independent Searching Strategy (ISS). However, none of them can achieve resilience and efficiency at the same time. In this paper, we propose a Gossip Dissemination Strategy (GDS) for load balancing in DHT systems, which attempts to achieve the benefits of both RDS and ISS. GDS doesn't rely on a few static rendezvous directories to perform load balancing. Instead, load information is disseminated within the formed groups via a gossip protocol, and each peer has enough information to act as the rendezvous directory and perform load balancing within its group. Besides intra-group balancing, inter-group balancing and emergent balancing are also supported by GDS. To further improve system resilience, the position of the rendezvous directory is randomized in each round. For a better understanding, we also perform analytical studies on GDS in terms of its scalability and efficiency under churn. Finally, the effectiveness of GDS is evaluated by extensive simulation under different workload and churn levels.

1 Introduction

In DHT (Distributed Hash Table)-based P2P systems, the previous work [10] has pointed out that, the uniformity of hash functions can't guarantee the perfect balance between nodes, and there exists an $O(\log N)$ imbalance factor in the number of objects stored at a node. What is more, due to the node heterogeneity or application semantics, it is possible for some nodes to be heavily loaded, while others are lightly loaded. Such unfair load distribution among peers will cause performance degradation, and also provide disin-

centives to participating peers.

To date, the importance of load balancing has motivated a number of proposals, e.g., [1], [7], [8], [3], [15], etc. Many of them are based on the concept of “virtual server” [10]. By shedding an appropriate number of virtual servers from heavy nodes to light nodes, load balancing can be achieved. According to the difference in load information management, existing migration-based approaches can be categorized into two representative strategies: (1) *Rendezvous Directory Strategy (RDS)* and (2) *Independent Searching Strategy (ISS)*.

In *RDS*, the load information of each peer is periodically published to a few fixed rendezvous directories, which are responsible for scheduling the load reassignment to achieve load balance. In *ISS*, a node doesn't publish its load information anywhere else. To achieve load balancing, the nodes should perform searching independently to find other nodes with inverse load characteristics, and then migrate load from the heavy nodes to the light nodes.

Due to information centralization, *RDS* can conduct the best load reassignment and be much more efficient than *ISS*. Existing *RDS* schemes (e.g., [3], [15]) mainly focus on the scalability problem of *RDS*, while little attention is paid to the resilience issue. In their approaches, the positions of rendezvous directories are static and known publicly to all the peers. It makes *RDS* vulnerable to the node-targeted attacks. In case that the rendezvous directory is occupied by malicious peers or overwhelmed by DoS traffic, the service of load balancing is halted. On the contrary, *ISS* is more resilient to the node-targeted attacks for there exists no central entity in the system. But its efficiency greatly depends on the searching scope. A big scope often incurs huge traffic and becomes unscalable in a large system. In case of a small scope, it is inefficient to achieve the system-wide load balance. Both *RDS* and *ISS* cannot achieve resilience and efficiency at the same time.

In this paper, we propose a new load balancing strategy for DHT-based P2P systems, called *Gossip Dissemination Strategy (GDS)*, which is scalable and achieves both the

efficiency of *RDS* and the resilience of *ISS*. In *GDS*, the whole system is formed into groups, and the gossip protocol is used for load information dissemination. After load information dissemination, every group member has the full load information of its own group and the utilization information of the whole system. Thus, each group member can act as the rendezvous directory to perform load reassignment within the group, but the position of the rendezvous directory is randomized to make *GDS* resistant to the node-targeted attacks. Besides intra-group load balancing, inter-group load balancing and emergent load balancing are also allowed in *GDS* to achieve a more balanced system state.

Particularly, we make the following contributions in this paper:

1. To the best of our knowledge, our proposed *Gossip Dissemination Strategy (GDS)* is the first migration-based load balancing strategy to achieve both resilience and efficiency. By utilizing gossip-based information dissemination and randomizing the positions of the rendezvous directory, *GDS* can retain the efficiency of *RDS*, while being more resilient to the node-targeted attacks.
2. We perform an analytical study on the scalability of *GDS*, and the impact of system churn on the efficiency of *GDS* to get a better understanding. It is found that, *GDS* can exhibit good scalability and the impact of churn could be minimized when the tuning parameters are properly configured.
3. We evaluate the performance of *GDS* via extensive simulations. Our simulation results show that, *GDS* can achieve comparable efficiency to *RDS* under different scenarios. But different from *RDS*, our proposed *GDS* is also resilient and scalable.

The remainder of this paper is structured as follows. We will first introduce the related work in Sec. 2. The detailed design of *Gossip Dissemination Strategy (GDS)* is presented in Sec. 3. In Sec. 4, we analyze the scalability of *GDS* and the impact of churn to *GDS*. In Sec. 5, extensive simulation is performed to verify the performance of *GDS*. Finally, Sec. 6 summarizes the whole paper.

2 Related Work

The topic of load balancing has been well studied in the field of distributed systems, while the characteristics of P2P systems pose new challenges for system designers. In this section, we briefly introduce existing approaches to load balancing in DHT-based P2P systems.

In DHT-based P2P systems, much research work is based on namespace balancing. Namespace balancing is trying to

balance the load across nodes by ensuring that each node is responsible for a balanced namespace. It is valid only under the assumption of uniform workload distribution and uniform node capacity. The balancing action can be invoked at the time of node join. Before joining the system, a node samples single/multiple points and selects the largest zone to split (e.g., [13], [7], etc). For better balancing effect, “*Virtual Servers*” were introduced in [10], by letting each node host multiple virtual servers, the $O(\log N)$ imbalance factor between nodes can be mitigated. If node heterogeneity is considered, we can allocate virtual servers proportional to node capacity (e.g., CFS [2]). In case that a node is overloaded, it simply removes some of its virtual servers. Such simple deletion will cause the problem of “load thrashing”, for the removed virtual servers may make other nodes overloaded. Pure namespace balancing like the above approaches doesn’t perform load migration, thus cannot handle workload skewness well.

A more general approach is the migration-based approach, which is applicable to various kinds of scenarios and able to handle workload skewness. A number of migration-based approaches have been proposed to date (e.g., [8], [3], [15], etc). Most of them are based on the concept of virtual servers. In [8], Rao et al. propose three simple load balancing schemes: “*one-to-one*”, “*one-to-many*” and “*many-to-many*”. Among them, “*one-to-many*” and “*many-to-many*” belong to the *RDS* category, while “*one-to-one*” belongs to the *ISS* category. To enable emergent load balancing, Godfrey et al. [3] made a combination of “*one-to-many*” and “*many-to-many*”, and use them in different scenarios. The scheme proposed in [15] also belongs to the *RDS* category, but its rendezvous directory is organized as a distributed k -ary tree embedded in the DHT. The effectiveness of *RDS* and *ISS* is compared analytically under different scenarios in [14].

In this paper, our proposed *Gossip Dissemination Strategy (GDS)* differs from previous work in that, we take both resilience and efficiency into account. By disseminating the load information and randomizing the rendezvous directory, we can exploit the benefits of both *RDS* and *ISS*.

Besides, there are some other approaches that try to realize load balancing in P2P systems, such as object balancing based on “Power of two choices” (e.g., [1], [6]), etc. Nevertheless, all these techniques can be regarded as complementary techniques and may be combined to provide a better load-balancing effect under certain scenarios.

3 System Design

In this section, we propose a *Gossip Dissemination Strategy (GDS)* for load balancing in DHT-based P2P systems. The objective of our design is to exploit the efficiency of *RDS* while improving the resilience and scalability at the

same time.

The basic idea of *GDS* is to disperse the responsibility of load balancing to all the peers, instead of limiting to only a few ones. At the same time, in order to avoid the inefficiency caused by independent searching, load information is disseminated among peers by gossip-like protocols. In each load-balancing round, peers are selected randomly as the rendezvous directories to schedule the reassignments of virtual servers for better balance.

In the following, we will introduce the system design in details.

3.1 Load Information Dissemination and Aggregation

GDS is built on top of ring-like DHTs, such as Pastry, Chord, etc. Based on the system size, one or more load balancing groups are formed among peers. For systems with only a few thousand peers, only one group is formed; but for systems with over tens of thousands of peers, peers form into multiple groups, each of which corresponds to a continuous region with equal size in the ring. All the peers within the same group share the same prefix, which is referred to as the “*GroupID*”.

In *GDS*, any group gossip protocol can be used for load information dissemination within the group. Here, to reduce the message traffic, a gossip “tree” embedded in the DHT is used for information dissemination. This tree doesn’t require explicit maintenance and just expands based on local information. The node who wants to disseminate its load information is the root of the tree. It sends the information to all the peers with the same *GroupID* in its routing table. Subsequently, when an intermediate node j receives the message from a node i (the one that has already received the information), j only forwards the message to every node k in its routing table satisfying that $prefix(i, j)$ is a prefix of $prefix(j, k)$. Here, $prefix(x, y)$ is defined as the maximum common prefix between node x and node y .

The load information of a node i to be published includes: (1) the node’s capacity: C_i ; (2) the load vector of all the virtual servers hosted by the node: $\vec{L}_i = \langle L_{i_1}, \dots, L_{i_m} \rangle$, where L_{i_k} refers the load of the k -th virtual server i_k hosted by node i . Denote $V(i)$ to be the set of virtual servers hosted by the node i , $i_k \in V(i)$, $k = 1..m$; (3) the node’s IP address. In every update period T_u , every node should publish its load information once.

For a small system with only one group G , the load information of each peer can be delivered quickly to all members within a short period. After that, each peer has the load information of all the other peers, and knows the total capacity and the total load of the whole system by $C = \sum_{i \in G} C_i$ and $L = \sum_{i \in G} \sum_{i_k \in V(i)} L_{i_k}$ respectively. Thus, the system utilization μ can be given by $\mu = L/C$.

A node i ’s utilization is given by $\mu_i = \frac{\sum_{i_k \in V(i)} L_{i_k}}{C_i}$. Based on the ratio between μ_i and μ , a node can be classified into three types: (1) Heavy node, if $\mu_i > \mu$; (2) Light node, if $\mu_i < \mu$; (3) Normal node, if $\mu_i = \mu$.

In case of a large system, there exist multiple groups. Each node i should run at least two virtual servers. Among them, one virtual server i_1 is with the identifier id_1 generated by hashing the node’s IP; and another virtual server i_2 is with the identifier id_2 generated by hashing id_1 . i_1 is called the *primary server* of node i ; and i_2 is called the *secondary server* of node i (see Fig. 1). Each node only publishes load information via its primary server. During load reassignment, the primary and secondary server can’t be moved to other nodes, but their size can be changed.

Given two arbitrary groups S and T , the probability that there exists at least one node i whose primary server i_1 is in group S and secondary server i_2 belongs to group T is given by $1 - e^{-c}$ [12], where c is the ratio between the group size and the number of groups. With $c = 5$, the probability is as high as 0.993.

It implies that, for any group, there exists at least one secondary server of its group members in any other group with high probability. It provides us with an opportunity to aggregate load information of other groups and calculate the system utilization. The information aggregation is important to achieve the system-wide load balancing.

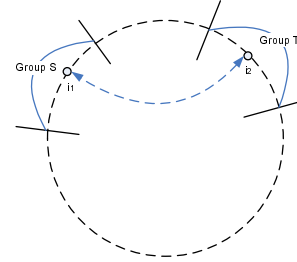


Figure 1. Primary server and secondary server.

The aggregation process is as follows: through gossiping in each group, for a given node i , it can have the full load information of at least two groups: group S , where its primary server i_1 is in; and group T , where its secondary server i_2 is in. To let other group members in group S learn the load status of group T , the node i also propagates the information about the average load level and the node number of group T within group S by piggybacking with its own load information. As the secondary servers of the group members in group S exist in almost all other groups, after each node disseminates the group load status, every member in the group S learns the load status of all other groups, and is

able to estimate the system-wide utilization independently.

3.2 Intra-Group Load Balancing

After load information dissemination, every member within a group holds the load information of the full group and the information of system-wide utilization. Each of them has the capability to act as the rendezvous directory to classify the nodes and schedule load reassignment in the group.

However, in order to be resilient to the node-targeted attacks, it is better to randomize the position of the rendezvous directory in each load-balancing round. Although some sophisticated randomization algorithms and security mechanisms can be used to make the position selection more secure, we adopt a simple but effective approach. It is based on the assumption that, a node-targeted attack is often costly and the probability to compromise a node within a short period is small.

The idea of our approach is as follows: we associate each round of load balancing with a sequence number seq_{lb} , which is known by all group members. seq_{lb} will be increased by 1 after every load balancing period. In each round, every node locally generates a random key by hashing seq_{lb} with a common hashing function. The prefix of a generated key should be replaced by *GroupID* to guarantee the node responsible for the key exists within the group. The node that hosts that key is selected as the rendezvous directory. After completion of load balancing, the node increases seq_{lb} by one and disseminates the value to all the group members for consistency checking of seq_{lb} . In the next round, another node will be chosen as the rendezvous directory due to the randomness of hashing function. Although the position of the rendezvous directory is still publicly known, but it becomes random and dynamic. Even if the current rendezvous directory is compromised, the service of load balancing can quickly be recovered. As there is no fixed rendezvous directory in the group, *GDS* is more resilient than *RDS* to the node-targeted attacks.

The problem of computing an optimal reassignment of virtual servers between heavy nodes and light nodes is NP-complete, therefore, a simple greedy matching algorithm similar to [3] is adopted here to perform virtual server reassignments.

3.3 Inter-Group Load Balancing

Inter-group load balancing is performed only when there exist multiple groups in the system. Since the distribution of light nodes and heavy nodes may be various in different groups, it is possible that even after intra-group load balancing, some groups may still have many light nodes, while

other groups may have many heavy nodes. To handle this situation, inter-group load balancing is allowed.

In a group S with many heavy nodes, its average utilization is higher than the average utilization of the whole system. Suppose that the node i is the current rendezvous directory of the group S , due to load information dissemination, the node i has the full load information of its own group and the load status of all other groups with high probability. The node i can select one group whose free capacity is slightly bigger than the required amount to perform inter-group load balancing.

The process is as follows: Given the inter-group load balancing to be performed between the group S and the group T , the rendezvous directory i of the group S first finds another node j within the group S whose secondary server exists in the group T , and then transfers the responsibility to j . Since j has the full load information of both the group S and the group T , it can perform the best-fit load reassignment among the nodes of the two groups.

3.4 Emergent Load Balancing

GDS can also provide emergent load balancing. In case that the utilization of one heavy node is beyond a threshold, it can directly contact the lightest node within its group; if there exist no light nodes in its own group, based on collected load information of other groups, it selects the group with the most amount of free capacity and sends a request to one random node in that group, which will return a light node to relieve its load.

With the combination of intra-group load balancing, inter-group load balancing and emergent load balancing, *GDS* can achieve a system-wide balanced state.

4 Performance Analysis

4.1 Scalability

The main concern is the scalability of *GDS*, which determines the applicability of *GDS* in P2P systems.

Define g to be the group size and N to be the system size. In case of a small system with a few thousands of peers, one group is formed and the group size g equals N . The message traffic that a node (including virtual node) experiences per interval is given by $W_{GDS} \approx O(g/T_u) = O(N)$. For example, given the update period $T_u = 60$ seconds, $N = 1,000$ and message size = 30 bytes, W_{GDS} is about $0.5k$ bytes/sec. If a node hosts multiple virtual servers, the traffic increases proportionally to the number of virtual servers. However, the traffic is still rather low and tolerable for normal users. Even for a powerful node with 20 virtual servers, the message traffic is about $10k$ bytes/sec. To

limit the traffic towards a node, each node can maintain a threshold number of virtual servers.

In a large system, multiple groups are formed. Define c to be the ratio between the number of nodes in a group and the number of groups, we have $g = \sqrt{cN}$. In case that $c \ll N$, the message traffic for a peer per interval is given by $W_{GDS} \approx O(g/T_u) = O(\sqrt{N})$. Given the update period $T_u = 60$ seconds, $N = 250,000$, $c = 4$ ($c \ll N$) and message size = 30 bytes, W_{GDS} is about 0.5k bytes/sec, which is comparable to the small system.

In our approach, c is an important tuning parameter. In order to guarantee efficient information dissemination between groups, c cannot be too small, and normally we set $c \geq 5$. As $c = g^2/N$, we can guarantee the above requirement by keeping the group size g bigger than \sqrt{cN} . Therefore, given the maximum possible system size N_{max} , we can define two thresholds: the minimum group size g_{min} , which is to guarantee a bigger c ; and the maximum group size g_{max} , which is to limit the message traffic towards a node.

In the evolution process of the system, the groups are split only when the group size is beyond the threshold g_{max} . The group split doesn't require global cooperation between nodes. As each node knows exactly the number of group members due to information dissemination, when the group size is beyond g_{max} , it will increase its *GroupID* by one bit locally. With the changes happening in each node, the system is reorganized into more groups. When the group size is below the threshold g_{min} , the merging of groups is incurred by self-adaptation similarly. Each node decreases the length of *GroupID* by one bit. The process is stopped when the *GroupID* is null, which means there exists only one group in the system.

4.2 Efficiency under Churn

The phenomenon of peer dynamics is called “churn”, and may cause the failure of load reassignments. There exist some peers that may leave after publishing their load information. If the rendezvous directory doesn't know their departure and still take them as live peers to perform load reassignment with other peers, it will result in the failure of some load reassignments and impact the efficiency of load balancing. Only the peers that are really alive at the time of load balancing are useful and we call them “real” live peers.

What we want to know is that, to what degree does the churn impact the efficiency of *GDS*, and how to tune the parameters to mitigate its impact?

Based on the measurement results from real P2P systems [9], we adopt the Pareto lifetime distribution in our analysis, in which the CDF (Cumulative Distribution Function) is given by $F(x) = 1 - (1 + \frac{x}{\beta})^{-\alpha}$, where α represents the heavy-tailed degree and β is a scale parameter. Under the

Pareto distribution, most peers have short lifetimes, while some peers stay much longer in the system.

When performing load balancing, we assume that the rendezvous directory only considers the peers that have made updates within the latest update period. By analysis, we have the following result:

Theorem 1 *Given the update period T_u , the probability that a peer is a “real” live peer in the peer set for balancing at the rendezvous directory is*

$$p_{real} = \frac{1}{\frac{\alpha-1}{\beta}T_u + \frac{\beta}{(2-\alpha)T_u}[(1 + \frac{T_u}{\beta})^{2-\alpha} - 1]}$$

for Pareto lifetime distribution with $\alpha > 2, \beta > 0$.

Proof Denote t_i to be the time for the i -th round of load balancing. Within the latest update period $[t_i - T_u, t_i]$, we denote the set of new joining peers as X . For the peers that leaves the system within this period, based on their live time and departure time, we divide them into three sets: (1) Y_1 , the set of nodes that are alive at time $t_i - T_u$, but leave the system without making any update in $[t_i - T_u, t_i]$; (2) Y_2 , the set of nodes that are alive at time $t_i - T_u$ and depart before t_i , but make one update before departing; (3) Y_3 , the set of nodes that join the system during $[t_i - T_u, t_i]$, but depart again before t_i .

During the load balancing action at t_i , the nodes in Y_1 will not be considered, for they don't make any update within $[t_i - T_u, t_i]$. But for the nodes in Y_2 and Y_3 , the rendezvous directory has no knowledge about their departure and still thinks they are alive. The nodes in Y_2 and Y_3 belong to “faked” live peers.

For our analysis is under steady state, the number of arrival peers equals the number of departure peers, thus we have $|X| = |Y_1| + |Y_2| + |Y_3|$. Let N be the number of live peers at time t_i , then the number of peers in the peer set for balancing at time t_i can be given by $N' = N + |X| - |Y_1| = N + |Y_2| + |Y_3|$. At time t_i , among N' peers, only N peers are “real” live nodes and p_{real} can be given by the ratio between N and N' . In order to compute p_{real} , we need to compute $|X|$ and $|Y_1|$ first.

Let λ be the arrival rate of peers under steady state. By applying Little's Law [4], we have $\lambda = (\alpha - 1)/\beta$ for Pareto lifetime distribution. Then the number of new arrival peers within $[t_i - T_u, t_i]$ is given by $|X| = N\lambda T_u$. After getting $|X|$, we then proceed to calculate $|Y_1|$.

Supposing that a peer is alive at time $t_i - T_u$, then according to [5], given the CDF of peer lifetime distribution as $F(x)$, the CDF of its residual lifetime R is given by $F_R(x) = P(R < x) = \frac{1}{E[L]} \int_0^x (1 - F(z)) dz$.

For Pareto lifetime distribution, we have $F_R(x) = 1 - (1 + \frac{x}{\beta})^{1-\alpha}$. Due to the randomness, the update time T of a peer since $t_i - T_u$ follows a uniform distribution in

$[0, T_u]$, with the PDF given by $f(x) = 1/T_u$. Define the probability that the peer's residual lifetime R is less than its first update time T as p_{Y_1} , then p_{Y_1} can be calculated by $p_{Y_1} = P(R < T) = \int_0^{T_u} P(R < x)f(x)dx = \frac{1}{T_u} \int_0^{T_u} F_R(x)dx$. Under Pareto lifetime distribution, we have $p_{Y_1} = 1 - \frac{\beta}{(2-\alpha)T_u} [(1 + \frac{T_u}{\beta})^{2-\alpha} - 1]$, $\alpha > 2$.

From the above, we can deduce $|Y_1| = N \times p_{Y_1}$ and get N' accordingly. After simple reduction, we can deduce the theorem based on $p_{real} = \frac{N}{N'}$. ■

To get a better understanding, we take a real P2P system Kad as a case for study. According to the measurement results in [11], the peer lifetime distribution in Kad follows a heavy-tailed distribution with the expectation of 2.71 hours, which can be approximated by a Pareto distribution with $\alpha = 2.1, \beta = 3.0$. Based on Theorem 1, we get the numerical results of p_{real} under different update periods (as shown in Table. 1).

Update Period T_u (in hour)	p_{real}
1/120	0.998472
1/60	0.996942
1/30	0.993880
1/12	0.984671

Table 1. Impact of churn

It can be observed that, by tuning the update period to be smaller, we can keep the p_{real} very high. In a group with 1000 nodes, by tuning $T_u = \frac{1}{60}$ hour (i.e., 60 seconds), there are less than 4 “faked” live peers in the balancing set on average. In the real environment, if the update period is properly configured, the impact of churn will not be a big issue for *GDS*.

5 Experimental Evaluation

5.1 Methodology

We evaluate the performance of *GDS* on top of a Chord simulator, which is enhanced to support virtual servers. The whole system consists of 8192 nodes, and the group thresholds g_{max} and g_{min} are set as 1024 and 256 respectively. Synthetic traces are generated to simulate the system churn. The peer lifetime satisfies a heavy-tailed Pareto distribution with $\alpha = 2.1$, which is close to the real P2P systems (e.g., [9], [11]). β is adjusted to simulate different churn levels.

In the experiment, the node capacity follows a Gnutella-like distribution, and the average node capacity is able to serve 100 requests per second. Initially, each node is assigned 5 virtual servers. Similar to [15], we also assume the size of a virtual server satisfies the exponential distribution.

The update period and the load balancing period are set as 60 and 180 seconds respectively.

The workload is generated by the requests from all the nodes uniformly, but the destinations are chosen from either the uniform or Zipf distribution. To serve a request will put one unit of load on the destination node. In case that a request is served by an overloaded node, the request is called “ill request”. We run the simulation for a rather long period, and take the average percentage of “ill requests” among all the requests as the metric to evaluate the effectiveness of load balancing algorithms.

For the reason of comparison, we also simulate another four strategies: (1) *Rendezvous Directory Strategy (RDS)*: there exists only one powerful rendezvous directory in the system, which can perform the optimal load reassignment; (2) *Independent Searching Strategy (ISS)*: the light node samples the id space for the heavy node, and the sample size k per interval is set as 1; (3) *Proportion Strategy*: the nodes remove or add virtual servers to make the load proportional to its capacity, similar to the approach adopted in [2]; (4) No Load Balancing for the system.

5.2 Experimental Results

5.2.1 Impact of Workload Level

In this experiment, we examine the effectiveness of load balancing strategies under different workload levels.

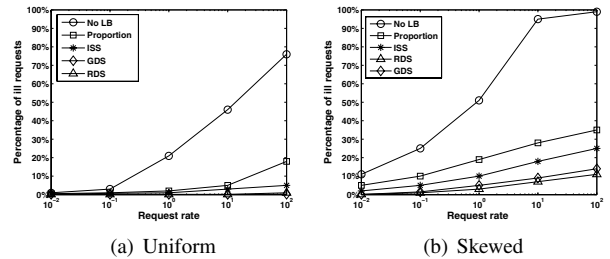


Figure 2. Percentage of ill requests under different degrees of workload (a) Uniform workload; (b) Skewed workload (zipf distribution with parameter 1.2).

By varying the request rate initiated by every node, the system workload level is changed accordingly and we record the percentage of ill requests to measure the effectiveness of load balancing strategies. Two kinds of workload distribution are considered: one is uniform distribution, another is skewed distribution (Zipf distribution with $\alpha = 1.2$).

The results are shown in Fig. 2. Under the uniform workload, *Proportion*, *ISS*, *RDS* and *GDS* can all guaran-

tee a low percentage of ill requests even with a high request rate. *GDS* is slightly worse than *RDS*, but outperforms *ISS* and *Proportion*. Under the skewed workload, all the strategies perform worse, but *GDS* and *RDS* still can retain a low percentage of ill requests. The reason lies in that, both *GDS* and *RDS* can utilize the global load information to achieve a more balanced system state, while *Proportion* and *ISS* can only perform balancing statically or based on limited information.

5.2.2 Impact of Workload Skewness

Under some scenarios, the workload may be highly skewed. We also study the effectiveness of load balancing strategies under different workload skewness. We vary the skewness by adjusting the parameter α of Zipf distribution. When $\alpha = 0$, it corresponds to a Uniform distribution without skewness.

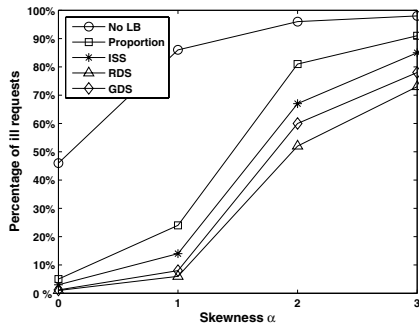


Figure 3. Percentage of ill requests under different workload skewness

Fig. 3 plots the percentage of ill requests under different workload skewness. It can be observed that, with the increase of skewness, the effectiveness of all the load balancing strategies is impacted greatly. When the workload is highly skewed (i.e., $\alpha = 3$), even *RDS* cannot handle the skewed workload well. However, under different skewness, *GDS* always has similar efficiency as *RDS*.

5.2.3 Impact of Workload Shift

In the above experiments, there is no shift on the workload distribution. It is desirable to know how the load balancing strategies respond in case of workload shifts. We still use the skewed workload (Zipf distribution with $\alpha = 1.2$), but we change the set of destinations in the middle.

Fig. 4 shows the impact of workload shift on the four load balancing strategies. The experiment lasts for 120 minutes, and the destination set is changed after 60 minutes.

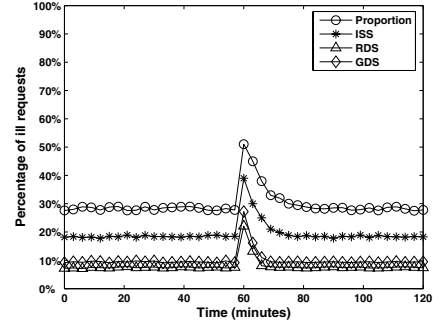


Figure 4. Percentage of ill requests under workload shift

We can find that, when the workload shift happens, there is a burst of ill requests due to the workload imbalance. After some time, all four strategies can smooth out the burst, but the response time is a bit different. Compared with *RDS* and *GDS*, *Proportion* and *ISS* need more time to converge to the level before workload shift happens. *GDS* has a short response time, but it is still a bit longer than *RDS*. It is because that, besides intra-group load balancing, *GDS* also needs to perform inter-group balancing in order to balance the whole system.

5.2.4 Impact of System Evolution

Although the scalability of *GDS* has been analyzed in Sec. 4, we further validate our results by simulation. In the experiment, the system size is varied from 500 to 9000. We measure the average message overhead per virtual server.

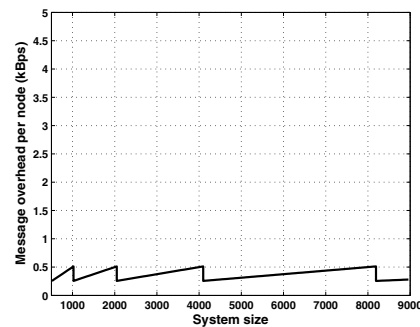


Figure 5. Message overhead per virtual server under different system size

From Fig. 5, we can observe that the message overhead per virtual server is less than $0.5k$ bytes/sec most of

the time, even with the increasing of the system size. It is because that, when the group size reaches a threshold, the group will be split to keep the traffic low. The effect of group splitting can be found in the figure. The message overhead increases continuously with the increasing of system size, but it drops down when reaching a threshold.

5.2.5 Impact of System Churn

We also study how the churn impacts on the load balancing strategies by varying the average peer lifetime. We run the experiments under both uniform and skewed workload distributions (Zipf distribution with $\alpha = 1.2$).

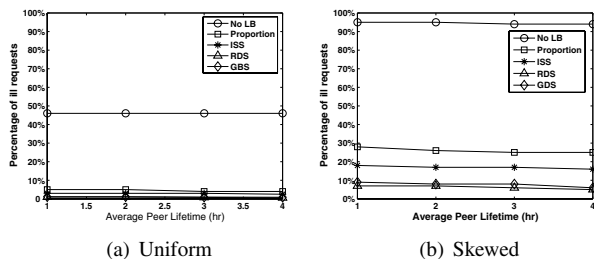


Figure 6. Percentage of ill requests under different churn rates (a) Uniform workload; (b) Skewed workload (zipf distribution with parameter 1.2).

Fig. 6 shows the percentage of ill requests under different churn rates. We can observe that, under both uniform and skewed workload, in spite of the changes of churn rate, *GDS* and *RDS* can always lead to a more balanced system state and minimize the percentage of ill requests. *GDS* and *RDS* perform slightly worse when the churn rate is higher. The reason is that, a high churn rate will increase the convergence time to the balanced state due to the existence of “faked” live peers, which cause the increase of ill requests accordingly.

In summary, we find that *GDS* can achieve almost the same efficiency as *RDS*. But different from *RDS*, *GDS* also achieves resilience and scalability at the same time.

6 Conclusion

In this paper, we propose a new load balancing strategy, called *Gossip Dissemination Strategy (GDS)*, to realize both the efficiency of *Rendezvous Directory Strategy (RDS)* and the resilience of *Independent Searching Strategy (ISS)*. Instead of reporting load information to the rendezvous directory or searching independently, the load information is disseminated via the gossip protocol. Gossiping makes it hard

to stop the load-balancing service by simply overwhelming a few rendezvous directories. In *GDS*, every peer can perform load scheduling within its group, but the responsibility is randomized to resist node-targeted attacks. Besides intra-group load balancing, *GDS* also supports inter-group load balancing and emergent load balancing. The performance of *GDS* is analyzed analytically, and we also perform extensive simulation to evaluate the effectiveness of *GDS* under different scenarios.

References

- [1] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *Proc. IPTPS'03*, 2003.
- [2] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In *Proc. 18th ACM Symposium on Operating Systems Principles SOSP'01*, 2001.
- [3] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in dynamic structured p2p systems. In *Proc. IEEE INFOCOM'04*, Hong Kong, Mar. 2004.
- [4] B. R. Haverkort. Performance of computer communication systems. *John Wiley & Sons*, 1998.
- [5] D. Leonard, V. Rai, and D. Loguinov. On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. In *Proc. ACM SIGMETRICS'05*, 2005.
- [6] M. Mitzenmacher. The power of two choices in randomized load balancing. *Doctoral Dissertation*, 1996.
- [7] M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *Proc. of ACM SPAA*, 2003.
- [8] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Proc. IPTPS'03*, Feb. 2003.
- [9] S. Saroiu. Measurement and analysis of internet content delivery systems. *Doctoral Dissertation, University of Washington*, Dec. 2004.
- [10] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM'01*, pages 149–160, San Diego, CA, Aug. 2001.
- [11] D. Stutzbach and R. Rejaie. Characterizing churn in peer-to-peer networks. In *Technical Report CIS-TR-05-03, Univ. of Oregon*, 2005.
- [12] C. Tang, M. J. Buco, R. N. Chang, S. Dwarkadas, L. Z. Luan, E. So, and C. Ward. Low traffic overlay networks with large routing tables. In *Proc. ACM SIGMETRICS'05*, 2005.
- [13] X. Wang, Y. Zhang, X. Li, and D. Loguinov. On zone-balancing of peer-to-peer networks: Analysis of random node join. In *Proc. of ACM SIGMETRICS*, 2004.
- [14] D. Wu, Y. Tian, and K.-W. Ng. On the effectiveness of migration-based load balancing strategies in dht systems. In *Proc. of IEEE ICCCN*, 2006.
- [15] Y. Zhu and Y. Hu. Efficient, proximity-aware load balancing for dht-based p2p systems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 16, Apr. 2005.