Load Balance with Imperfect Information in Structured Peer-to-Peer Systems

Hung-Chang Hsiao, *Member*, *IEEE Computer Society*, Hao Liao, Ssu-Ta Chen, and Kuo-Chan Huang

Abstract—With the notion of virtual servers, peers participating in a heterogeneous, structured peer-to-peer (P2P) network may host different numbers of virtual servers, and by migrating virtual servers, peers can balance their loads proportional to their capacities. The existing and decentralized load balance algorithms designed for the heterogeneous, structured P2P networks either explicitly construct auxiliary networks to manipulate global information or implicitly demand the P2P substrates organized in a hierarchical fashion. Without relying on any auxiliary networks and independent of the geometry of the P2P substrates, we present, in this paper, a novel load balancing algorithm that is unique in that each participating peer is based on the partial knowledge of the system to estimate the probability distributions of the capacities of peers and the loads of virtual servers, resulting in imperfect knowledge of the system state. With the imperfect system state, peers can compute their expected loads and reallocate their loads in parallel. Through extensive simulations, we compare our proposal to prior load balancing algorithms.

Index Terms—Peer-to-peer systems, load balance, heterogeneity.

1 Introduction

Peer-to-peer (P2P) networking is an emerging technique for next-generation network applications. P2P networks (or *overlays*) are application-level networks built on top of end systems, which provide message routing and delivery. Popular P2P applications include file sharing, distributed computing, media streaming, and others, which rely on their P2P network infrastructures for message routing, information searches, and/or content delivery.

P2P network infrastructures are key building blocks in the design and implementation of successful P2P applications. Potential P2P substrates are based on *distributed hash tables* or DHTs for short. Examples of DHTs are Chord [1] and Pastry [2].

As peers participating in a DHT are often heterogeneous, the work in [1] introduces the notion of *virtual servers* to cope with the heterogeneity of peers. Participating peers in a DHT can host different numbers of virtual servers, thus taking advantage of peer heterogeneity. Let V be the set of virtual servers in the system, and N be the set of participating peers. By reallocating virtual servers in V to peers in N, [3], [4], [5] suggest that each peer $i \in N$ deal with the load balancing problem by minimizing the following:

$$\left| \frac{\sum_{v \in V_i} L_v}{C_i} - \mathcal{A} \right|, \tag{1}$$

where $V_i \subset V$ represents the set of virtual servers allocated to peer i; L_v is the load value of a virtual server $v \in V_i$; C_i is the capacity value of peer i; and

$$\mathcal{A} = \frac{\sum_{v \in V} L_v}{\sum_{i \in N} C_i}$$

is the expected load value per unit capacity. We denote

$$LBF_i = \frac{\sum_{v \in V_i} L_v}{C_i}$$

in (1) as the *load imbalance factor* of peer i. Minimizing $|LBF_i - \mathcal{A}|$ implies that peer i manages the loads of virtual servers proportional to its capacity, leading to a *load-balanced* DHT network.

First, we note, in this paper, that the reallocation of a virtual server from a source peer to a destination peer can be simply done by simulating the *leave* and *join* operations offered by a typical DHT. Second, L_v of any virtual server v at a particular time is the sum of the loads of the objects (data items) stored in v at that time. Possible metrics for measuring the load of an object may include the storage size of the object, the mean bandwidth required for serving the object, and so on. Third, C_i represents the maximum load that peer i is willing to hold, which may denote the available disk space, processor speed, and the bandwidth of peer i, for example. We do not assume a particular resource in this paper. However, similar to prior studies (e.g., [3], [4], [5]), we assume that there is only one bottleneck resource in the system, leaving multiple-resource load balancing in the future.

For balancing loads among participating peers, we are also concerned with the minimization of the systemwide performance metric—*movement cost*—as much as possible [3], [4], [5]. More precisely, we attempt to determine a subset $S \subseteq V$, denoting the virtual servers selected to be moved, such that the movement cost, MC, defined in the following is minimized:

$$MC = \sum_{v \in S} L_v. \tag{2}$$

• *The authors are with the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan 701, Taiwan. E-mail: hchsiao@csie.ncku.edu.tw, {p7895128, p7697104}@mail.ncku.edu.tw.*

Manuscript received 7 Sept. 2009; revised 22 Dec. 2009; accepted 4 Apr. 2010; published online 18 May 2010.
Recommended for acceptance by M. Guo.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-09-0409.
Digital Object Identifier no. 10.1109/TPDS.2010.105.

Accordingly, by the minimization of (1) and (2), we intend to distribute loads to participating peers evenly with the minimum movement cost.

In this study, we are interested in solving the load balancing problem in a fully decentralized manner. While pioneer studies presenting load balancing algorithms for distributed systems can be found in the literature (e.g., [6], [7]), these studies either propose centralized load balancing algorithms [6] or often aim at static, small-scale systems and/or homogeneous environments [7]. As the centralized algorithms may introduce the performance bottleneck and the single point of failure, recent proposals presenting centralized algorithms that rely on a few rendezvous nodes[1] to balance the loads of peers in a DHT can be found in [3], [8].

To the best of our knowledge, the existing load balancing algorithms that operate in a distributed manner and aim at dynamic and heterogeneous DHTs include the studies [4], [5]. In [4], [5], each node, k, typically specifies a target load, T_k, and $T_k \leq C_k$. Without exceeding T_k, k can only accept virtual servers, such that the total load of the virtual servers in k is no more than T_k. Obviously, the ideal setting for T_k is $\mathcal{A} \times C_k$, that is, k is expected to manage the load proportional to its capacity. While Shen and Xu [5] explicitly configure T_k for each peer k independent of the system state, the proposal by Zhu and Hu [4] acquires the global knowledge of the entire system to compute the expected load per unit peer capacity, $\mathcal{A}$, spontaneously and to disseminate the value of $\mathcal{A}$ to all participants, allowing each peer k to compute its $T_k = \mathcal{A} \times C_k$.

In this paper, we present a novel load balancing algorithm to minimize both (1) and (2) as much as possible. The unique feature of our proposal is that each participating peer estimates and represents the "system state" as the probability distributions for the capacities of nodes and the loads of virtual servers. The approximated probability distributions not only help estimate the expected load a peer should perceive but also provide hints for each peer in the system to schedule the transfers of virtual servers. Unlike [3], [8], the participating peers in our proposal operate independently, and they need not rely on dedicated nodes to pair virtual servers and participating peers, eliminating the performance bottleneck and single point of failure. On the other hand, the decentralized algorithm in [4] not only depends on global knowledge (i.e., $\mathcal{A}$) but also requires coordination among the participating peers to transfer their virtual servers; this may introduce extra workload to the peers responsible for the coordination. In contrast, each peer in our proposal independently and solely manipulates partial information of the system and then reassigns its virtual servers to other peers based on the approximated system state. While it can be compared with [5], our proposal is independent of the geometry of DHTs.

1.1 Roadmap

The remainder of the paper is organized as follows: Section 2 discusses the related works. Our load balancing algorithm is presented in Section 3. We evaluate our proposal in simulations, and the simulation results are discussed in Section 4. We summarize our study in Section 5.

1. The terminologies, peers and nodes, are interchangeable in this paper.

2 Related Work

Eariler studies (e.g., [6], [7]) have proposed load balancing algorithms, targeting at static, small-scale, and/or homogeneous environments. Due to space limitation, we provide a concise review of the load balancing techniques designed for DHTs in this section. As the previous study [9] has provided a survey on the load sharing algorithms in traditional high-performance computing systems, we refer interested readers to [10] for an interesting survey on the load balancing algorithms in DHTs—the emerging Internet-based autonomous network infrastructures.

Consider an object set O and a peer set N. Conventional DHTs (e.g., Chord [1] and Pastry [2]) assume that the loads of objects in O are identical. The load of a peer can thus be estimated as the number of objects hosted by the peer. Assuming that the load of each object $o \in O$ is $l_o = 1$, earlier studies [1], [11] show that the load imbalance factor in a typical DHT can be up to $O(\log n)$, where $n = |N|$ is the total number of nodes participating in the system. To be more precise, the maximum load of a peer can be $O(\log n)$ times the average. Later, several proposals (e.g., [11]) have reduced the load imbalance factor to a constant. In contrast to [11], the proposals (e.g., [3]) exploit the heterogeneity of peers, such that the number of objects allocated to a peer is proportional to the peer's capacity. Unlike [1], [3], [11], our work presented in this paper does not assume that objects in the system contribute the identical load to peers. That is, our proposal does not aim to balance loads among peers in terms of numbers of objects allocated to peers.

Instead of simply assuming $l_o = 1$ for all $o \in O$ and evenly partitioning the key space into each peer, Chord [1] suggests the notion of *virtual servers*. Different virtual servers in a system manage disjoint key subspaces, with a virtual server serving as an elementary entity for balancing loads among peers. We note that if a virtual server v manages the key subspace $\mathcal{S}$, then the objects, which may have unequal loads and whose keys are within $\mathcal{S}$, contribute their loads to v. The idea of virtual servers enables a DHT to reallocate the virtual servers, such that the resultant load of a peer is proportional to the peer's capacity.

Based on the concept of virtual servers, the *many-to-many* framework is presented in [12] to cope with the load imbalance in a DHT. In the many-to-many framework, light and heavy nodes register their loads with some dedicated nodes, namely, the *directories*. The directories compute matches between heavy and light nodes and then, respectively, request the heavy and light nodes to transfer and to receive designated virtual servers. As noted by the authors in [3], the many-to-many framework essentially reduces the load balancing problem to a centralized algorithmic problem. As the entire system heavily depends on the directory nodes, the directory nodes may thus become the performance bottleneck and single point of failure. In contrast, in this paper, we are

TABLE 1
Notations Frequently Used in This Paper

Symbol	Description
N	set of participating peers
V	set of virtual servers hosted by the peers in N
n, m	$n = \|N\|$ and $m = \|V\|$
C_i	capacity of peer $i \in N$
L_v	load of virtual server $v \in V$
$\mathcal{A}$	expected load per unit capacity
$\tilde{\mathcal{A}}$	estimate for $\mathcal{A}$
T_i	target load of peer i
$\mathcal{I}$	set of peers sampled by each peer
$\Pr(\mathbf{X})$	probability distribution of the capacities of participating peers
$\widetilde{\Pr}(\mathbf{X})$	estimate for $\Pr(\mathbf{X})$
$F_X(\cdot)$	probability density function of $\widetilde{\Pr}(\mathbf{X})$
$\Pr(\mathbf{Y})$	probability distribution of the loads of virtual servers
$\widetilde{\Pr}(\mathbf{Y})$	estimate for $\Pr(\mathbf{Y})$
$F_Y(\cdot)$	probability density function of $\widetilde{\Pr}(\mathbf{Y})$
$F_{X-}(\cdot)$	probability density function representing the remaining capacity of the underloaded peers
$F_{Y+}(\cdot)$	probability density function characterizing the loads of virtual servers that are selected to moved
ρ, $\hat{\rho}$, φ, $\hat{\varphi}$, δ, $\hat{\delta}$	system parameters known by each participating peer in our proposal (see Section 3.2)

particularly interested in fully distributed solutions to the load balancing problem.

The study most relevant to ours is perhaps the work by Zhu and Hu [4]. Zhu and Hu [4] organize virtual servers in a DHT network into an auxiliary tree-shaped overlay network on top of the DHT. Such tree-shaped overlay is used to compute and disseminate the global aggregate, namely, the average

$$\mathcal{A} = \frac{\sum_{v \in V} L_v}{\sum_{i \in N} C_i},$$

allowing each peer k in the system to be based on $\mathcal{A}$ to compute exactly its target load threshold T_k and then to identify whether it is heavy or light. Additionally, the tree overlay facilitates the reallocation of virtual servers. The reallocation is performed in a bottom-up fashion toward the root of the tree. As we will discuss in Section 4, while [4] strives to distribute loads evenly due to the virtual servers, the nodes in the tree experience skew the workload for coordinating the reallocation of virtual servers, thus introducing another load imbalance issue.

In contrast to [4], our solution attacking the load balancing problem need not rely on any auxiliary tree networks, which are clearly failure-prone and thus require sophisticated maintenance. In addition, while the solution in [4] aggregates global information to compute $\mathcal{A}$ and then to reassign virtual servers, each peer in our proposal is independently based on partial knowledge of the system to reallocate its virtual servers. Specifically, the nodes in our proposal need not coordinate the reallocation of their virtual servers, leading to a design more appropriate for large-scale, dynamic environments.

Shen and Xu [5] also present a fully decentralized method for migrating objects (i.e., data items) stored in a DHT. Akin to the solution in [4], Shen and Xu suggest organizing a DHT into a two-level hierarchical network, where the higher level of the network consists of local clusters. Objects with excess loads are moved in a local cluster to balance the loads of the peers located in the same cluster. For those objects that cannot be moved in a local cluster, they can be transferred to peers in some foreign clusters. Unlike [5], our proposal does not assume any specific geometry of DHTs and is thus applicable to any DHT network.

3 Our Proposal

In Table 1, we summarize the notations frequently used in this paper for easy reference.

3.1 Algorithm Sketch

In this paper, we assume that the entire hash space provided by a DHT is $[0, 1]$, and each virtual server in the DHT has a unique ID selected independently and uniformly at random from the space $[0, 1]$.

Let N be the set of participating peers, and V be the set of virtual servers hosted by the peers in N in the DHT. Denote the set of virtual servers in peer i by V_i. Each peer $i \in N$ in our proposal estimates the load, which is denoted by T_i, that it should perceive, where $T_i = \tilde{\mathcal{A}} \times C_i + \epsilon$, $\tilde{\mathcal{A}}$ is an estimation for the expected load per unit capacity, i.e.,

$$\mathcal{A} = \frac{\sum_{v \in V} l_v}{\sum_{i \in N} C_i},$$

and ϵ is a predefined system parameter. If the current total load of i is greater than T_i (i.e., i is *overloaded*), then i migrates some of its virtual servers to other peers. Otherwise, i is *underloaded*, which does nothing but waits to receive the migrated virtual servers. For an overloaded peer (e.g., peer i), i picks those virtual servers for migration, such that 1) i becomes underloaded, and 2) the total movement cost, MC, in (2) is minimized due to the reallocation. If i is an underloaded peer, then i may be requested to receive a migrated virtual server, and i accepts such a virtual server if the added load due to the virtual server will not overload itself; otherwise, i rejects such virtual server. Algorithm 1 (REALLOCATION(i)), which given as follows illustrates our idea.

input : $\widetilde{\Pr}(\mathbf{X}), \widetilde{\Pr}(\mathbf{Y}), \mathcal{I}, n$

$\tilde{\mathcal{A}} \leftarrow \ln n \cdot \frac{\int_{y=0}^{L_{max}} (yF_Y(y)dy)}{\int_{x=0}^{C_{max}} (xF_X(x)dx)}$;
$T_i \leftarrow \tilde{\mathcal{A}} \times C_i + \epsilon$;
switch LOAD(i) **do**
  case $> T_i$
    $U_i \leftarrow \emptyset$;
    while LOAD(i) $> T_i$ and $V_i \neq U_i$ **do**
      $v \leftarrow \arg\min\{L_v | v \in V_i - U_i\}$;
      find $j \in \mathcal{I}$ satisfing Eq. (4) to accommodate v;
      if j accepts v **then**
        $V_i \leftarrow V_i - \{v\}$;
        $U_i \leftarrow U_i \cup \{v\}$;
      break;
  case $\leq T_i$
    while LOAD(i) $< T_i$ **do**
      receive v to host;
      $V_i \leftarrow V_i \cup \{v\}$;
    break;

Algorithm 1: REALLOCATION(i). Peer i computes the reallocation of its local virtual servers, where LOAD(i) $= \sum_{v \in V_i} L_v$.

Notably, similar to studies in [3], [4], [5], the participating peers in our proposal balance their loads periodically every time period (e.g., $\mathcal{T}$ minutes). However, we impose no global synchronization among the peers, and in our performance study to be discussed later on, each peer schedules its load balancing algorithm every $\mathcal{T}$ minutes according to its local clock. Next, Algorithm 1 intends to minimize MC by simply choosing a virtual server v with the minimum load each time until the hosting peer becomes underloaded.[2]

As we can see in Algorithm 1, the challenges of implementing the algorithm are 1) how a peer precisely and timely estimates $\mathcal{A}$ and 2) how an overloaded peer seeks the peers to receive its migrated virtual servers for balancing the loads among peers. To deal with these issues, our idea is to represent the capacities of participating peers and the loads of virtual servers as the probability distributions, which are denoted by $\Pr(\mathbf{X} < x)$ and $\Pr(\mathbf{Y} < y)$, respectively. Both $\Pr(\mathbf{X} < x)$ and $\Pr(\mathbf{Y} < y)$ provide valuable information to help the participating peers estimate $\mathcal{A}$, and the overloaded peers discover the underloaded peers to share their excess loads.

In this paper, we present a novel scheme for each peer i to approximate $\Pr(\mathbf{X} < x)$ and $\Pr(\mathbf{Y} < y)$ by "sampling" a few number of nodes, denoted by set $\mathcal{I}$, in the system. We will detail the estimation of the two probability distributions $\Pr(\mathbf{X} < x)$ and $\Pr(\mathbf{Y} < y)$ in Section 3.2.

Given $\mathcal{I}$, denote the approximation performed by each peer i for the probability distributions $\Pr(\mathbf{X} < x)$ and $\Pr(\mathbf{Y} < y)$ as $\widetilde{\Pr}(\mathbf{X} < x) = \int_0^{C_{max}} F_X(x)dx$ and $\widetilde{\Pr}(\mathbf{Y} < y) = \int_0^{L_{max}} F_Y(y)dy$, respectively. Here, $F_X(x)$ and $F_Y(y)$ are, respectively, the probability density functions for $\widetilde{\Pr}(\mathbf{X} < x)$ and $\widetilde{\Pr}(\mathbf{Y} < y)$, and C_{max} (L_{max}) is an arbitrary value greater than the capacity (load) of any peer (virtual server) in the system. Based on $\widetilde{\Pr}(\mathbf{X} < x)$ and $\widetilde{\Pr}(\mathbf{Y} < y)$, i then computes the expected load per unit capacity $\tilde{\mathcal{A}}$ as

$$\frac{\int_{y=0}^{L_{max}} (m \cdot yF_Y(y)dy)}{\int_{x=0}^{C_{max}} (n \cdot xF_X(x)dx)} = \frac{m}{n} \cdot \frac{\int_{y=0}^{L_{max}} (yF_Y(y)dy)}{\int_{x=0}^{C_{max}} (xF_X(x)dx)},$$

where $n = |N|$ and $m = |V|$. As suggested by [14], [15], $\frac{m}{n} = \ln n$.[3] Therefore, $\tilde{\mathcal{A}}$ estimated by i is

$$\tilde{\mathcal{A}} = \ln n \cdot \frac{\int_{y=0}^{L_{max}} (yF_Y(y)dy)}{\int_{x=0}^{C_{max}} (xF_X(x)dx)}. \quad (3)$$

Notably, in our implementation, we further improve the accuracy of $\tilde{\mathcal{A}}$ by having

$$\tilde{\mathcal{A}} = \frac{\sum_{j \in \mathcal{I}'} \tilde{\mathcal{A}}_j}{|\mathcal{I}'|},$$

where $\mathcal{I}'$ denotes the nodes in $\mathcal{I}$ whose $\tilde{\mathcal{A}}$s are available to i, and $\tilde{\mathcal{A}}_j$ represents the estimation for $\mathcal{A}$ by peer $j \in \mathcal{I}'$. We will discuss in Section 3.4 how we estimate the number of participating peers, $|N|$, and the number of virtual servers, $|V|$.

Having $\tilde{\mathcal{A}}$ and thus $T_i = \tilde{\mathcal{A}} \times C_i + \epsilon$, peer i can then determine whether it is overloaded or not. As mentioned, if i is overloaded, i dispatches some of its local virtual servers to other peers to reduce its excess load. In our proposal, i reallocates its excess load to the underloaded peers in $\mathcal{I}$, such that the underloaded peers accept the load proportional to their "available capacities." By the available capacity of a peer k, we indicate $T_k - \sum_{u \in V_k} L_u$, that is, the remaining capacity that k can manage without being overloaded. Hence, our intention is to reallocate a virtual sever, v, to an underloaded peer, j, whose available capacity is proportional to the total available capacity of the system, that is,

$$\frac{T_j - \sum_{u \in V_j} L_u}{\sum_{k \in X^-} (T_k - \sum_{u \in V_k} L_u)} = \frac{L_v}{\sum_{u \in Y^+} L_u},$$

where X^- denotes the set of underloaded peers, and Y^+ represents the set of virtual servers selected for migration by the overloaded peers.

In our proposal (i.e., Algorithm 1), let v be one of the virtual servers that i intends to reallocate, assuming i is an overloaded peer. As peer i does not have global knowledge of the system, i seeks peer $j \in \mathcal{I}$ to accommodate v if j satisfies the following:

$$\mathcal{R}_{Y^+}(v) \in [\mathcal{R}_{X^-}(j).pred, \mathcal{R}_{X^-}(j)], \quad (4)$$

where $\mathcal{R}_{X^-}(j)$ is the ratio of the total available capacity of the underloaded peers in $\mathcal{I}$ whose remaining capacities are no less than j's remaining capacity (i.e., $\geq T_j - \sum_{u \in V_j} L_u$) to the total available capacity of all the underloaded peers in $\mathcal{I}$; and $\mathcal{R}_{X^-}(j).pred$ is the ratio for some peer $k \in \mathcal{I}$ immediately preceding $\mathcal{R}_{X^-}(j)$, that is, there is no other peer $k' \in \mathcal{I}$, such that $\mathcal{R}_{X^-}(k) < \mathcal{R}_{X^-}(k') < \mathcal{R}_{X^-}(j)$. Precisely,

2. Notice that finding the set of the virtual servers (denoted by S_i), hosted by peer i, with the minimum MC subject to $\sum_{v \in V_i} L_v - \sum_{v \in S_i} L_v \leq T_i$ is a typical *knapsack problem*, which is $\mathcal{NP}$-hard [13].

3. Conventional DHTs (e.g., Chord) assumes that each object generates the same load to the system. Hence, we can compute the load imbalance factor of a peer by calculating the number of objects hosted by that peer. Studies [1], [11] conclude that the load imbalance factor in typical DHTs can be up to $O(\log n)$ times the average, where n is the number of peers in the system. However, with the idea of virtual servers, if each peer hosts $\log n$ virtual servers (i.e., the total number of virtual servers is $m = n \log n$), then, the load imbalance factor becomes $\Theta(1)$ [14], [15].

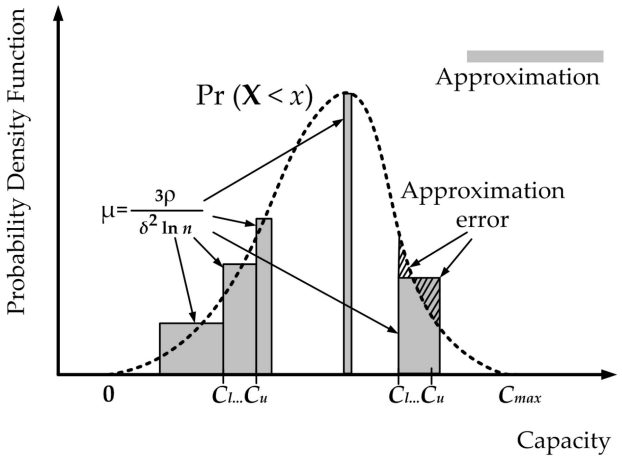


Fig. 1. An illustration of Algorithm 2.

$$\mathcal{R}_{X^-}(j) = \frac{\int_{x=T_j-\sum_{u\in V_j} L_u}^{C_{max}} (x \cdot F_{X^-}(x)dx)}{\int_{x=0}^{C_{max}} (x \cdot F_{X^-}(x)dx)}. \quad (5)$$

Here, $F_{X^-}(x)$ is the approximated probability density function that characterizes the available capacity distribution of the underloaded peers.

On the other hand, $\mathcal{R}_{Y^+}(v)$ in (4) is the ratio of the total load of the virtual servers selected for migration whose loads are no less than L_v to the total load of these virtual servers, that is,

$$\mathcal{R}_{Y^+}(v) = \frac{\int_{y=L_v}^{L_{max}} (y \cdot F_{Y^+}(y)dy)}{\int_{y=0}^{L_{max}} (y \cdot F_{Y^+}(y)dy)}, \quad (6)$$

where $F_{Y^+}(y)$ is the probability density function representing the load distribution of the virtual servers selected to be migrated by the overloaded peers. Consequently, by (4), we align the probability distribution of the available capacities of the underloaded peers and the probability distribution of the loads of virtual servers selected for migration, aiming for satisfying

$$\frac{T_j - \sum_{u\in V_j} L_u}{\sum_{k\in X^-}(T_k - \sum_{u\in V_k} L_u)} = \frac{L_v}{\sum_{u\in Y^+} L_u}.$$

Note that in our design, we simply derive $F_{X^-}(x)$ and $F_{Y^+}(y)$ based on the sampled peer set $\mathcal{I}$. This will be discussed in Section 3.3.

3.2 Approximating $\Pr(\mathbf{X})$ and $\Pr(\mathbf{Y})$

3.2.1 Computing $\widetilde{\Pr}(\mathbf{X})$

In our proposal, each peer, i, depends on *random walk* (Appendix A) to sample a set of nodes independently and uniformly at random from the network. Based on the set of the sampled nodes, i approximates the probability distribution of nodes capacity (i.e., $\Pr(\mathbf{X})$). We first state the procedure, which is Algorithm 2, performed by i to estimate the distribution of the capacities of participating peers and then conclude the quality of Algorithm 2 in Theorem 1.

TABLE 2
The $\ln^2 n$ Samples ($\rho = \frac{\ln 2+\ln 10}{\varphi \ln n}$, $\delta = 0.99$, and $\varphi = 1$)

System size (n)	Overhead $\left(\frac{\text{samples}}{n}\right)$	Error ($\delta\mu$)	Probability ($1-2n^{-\rho\varphi}$)
1,000	.047717	$\leq$ .1902	$>$ .9
10,000	.008483	$\leq$ .1070	$>$ .9
100,000	.001325	$\leq$ .0685	$>$ .9
1,000,000	.000191	$\leq$ .0476	$>$ .9
10,000,000	.000026	$\leq$ .0349	$>$ .9

input : N
output : $\mathcal{I}, \mathcal{L}$
1 $\mathcal{L} \leftarrow \{0, C_{max}\}$;
2 $\mathcal{I} \leftarrow \{n_1, n_2, \cdots, n_{\varphi \ln^2 n}\} \subseteq N$, where $C_{n_i} \leq C_{n_j}$ (for any $i < j$), and $n_i (1 \leq i \leq \varphi \ln^2 n)$ is picked independently and uniformly at random from N;
3 $\hat{\mathcal{I}} \leftarrow \mathcal{I}$;
4 $\ell \leftarrow \lceil \frac{3\rho|\mathcal{I}|}{\delta^2 \ln n} \rceil$;
5 **while** $\hat{\mathcal{I}} \neq \emptyset$ **do**
6 $\quad I \leftarrow \{n_1, n_2, \cdots, n_i\} \subseteq \hat{\mathcal{I}}$ and $|I| = \ell$;
7 $\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{C_{n_i}\}$;
8 $\quad \hat{\mathcal{I}} \leftarrow \hat{\mathcal{I}} - I$;
9 **return** $\mathcal{I}, \mathcal{L}$;

Algorithm 2: DISTRIBUTION(i). Return $\mathcal{I}$ is the set of sampled nodes, and $\mathcal{L}$ is the estimation for the probability distribution of the capacities of nodes in N.

Algorithm 2 (DISTRIBUTION(i)) depicts the details performed by any peer, i, to approximate the probability distribution of the capacities of nodes in N. The idea is to have each peer i sample $\varphi \ln^2 n$ nodes, denoted by the peer set $\mathcal{I}$, independently and uniformly at random in the network, where φ ($\varphi \geq 1$) is the system parameter. i then seeks an interval, $[C_l, C_u]$ (where $l, u \in \mathcal{I}$), such that the probability of a node with a capacity value in $[C_l, C_u]$ is not less than $\frac{3\rho}{\delta^2 \ln n}$, where ρ and δ are the system parameters. i removes the nodes in $\mathcal{I}$ with capacities in $[C_l, C_u]$ and iterates this process until $\mathcal{I}$ becomes empty. The resultant disjoint intervals $[C_l, C_u]$, each defining the probability (i.e., $\frac{3\rho}{\delta^2 \ln n}$) for the event that any peer j has the capacity value $C_j \in [C_l, C_u]$, approximate the probability distribution for the capacities of nodes in N.

Fig. 1 illustrates Algorithm 2, which finds a set of pairs (l, u), such that for each $[C_l, C_u] \subset [0, C_{max}]$ the probability of a node having $C_i \in [C_l, C_u]$ is at least $\frac{3\rho}{\delta^2 \ln n}$.

Our approximation scheme (i.e., Algorithm 2) guarantees the following result, whose detailed proof is given in Appendix B.

Theorem 1. *Let $\mu = \frac{3\rho}{\delta^2 \ln n}$ be the fraction of nodes with capacities in $[C_l, C_u]$, where $0 < \delta < 1$. Let γ, which Algorithm 2 estimates using $\varphi \ln^2 n$ samples, be the fraction of nodes with capacity values in $[C_l, C_u]$. Algorithm 2 then guarantees that the approximation error, $|\gamma - \mu|$, is bounded from above by $\Pr(|\gamma - \mu| \leq \delta\mu) > 1 - 2(\frac{1}{n})^{\rho\varphi}$.*

Table 2 illustrates the numerical example for Theorem 1 by letting $\rho = \frac{\ln 2+\ln 10}{\varphi \ln n}$ and $\varphi = 1$, such that $1 - 2n^{-\rho\varphi} > 0.9$. As φ is 1, each node performs $\ln^2 n$ samples. When the system size is up to $n = 10^5$ or more, the approximation

TABLE 3
The $\ln^3 n$ Samples ($\rho = \frac{\ln 2 + \ln 10}{\varphi \ln n}$, $\delta = 0.99$, and $\varphi = \ln n$)

System size (n)	Overhead $\left(\frac{\text{samples}}{n}\right)$	Error ($\delta\mu$)	Probability ($1 - 2n^{-\rho\varphi}$)
1,000	.329618	$\leq .0275$	$> .9$
10,000	.078132	$\leq .0116$	$> .9$
100,000	.015260	$\leq .0059$	$> .9$
1,000,000	.002637	$\leq .0034$	$> .9$
10,000,000	.000419	$\leq .0022$	$> .9$

error $|\gamma - \mu|$ for μ would be ≤ 0.0685, with the probability greater than 0.9. This only samples about $\leq 0.1\%$ of the participating peers (i.e., the ratio of samples to the number of participating nodes).

If each node performs $\ln^3 n$ samples (i.e., $\varphi = \ln n$), then we can considerably reduce the approximation error, although this is at the expense of a greater overhead in performing the sampling. As presented in Table 3, when the number of participating nodes is up to 10^5 or more, performing $\ln^3 n$ samples allows the approximation error $|\gamma - \mu|$ of ≤ 0.0059 compared with that of ≤ 0.0685 with $\ln^2 n$ samples. However, performing $\ln^3 n$ samples introduces an order of magnitude of sampling overheads compared with that using $\ln^2 n$ samples.

Notably, given $\mathcal{L} = \{C_0 = 0, C_1, C_2, \ldots, C_k = C_{max}\}$ due to Algorithm 2, we represent $\int_{x=k}^{C_{max}} (xF_X(x)dx)$ in (3) by $\sum_{j=i}^{k-1} (\frac{C_j + C_{j+1}}{2} \cdot \gamma)$, where $C_i < k < C_{i+1}$ and $C_i, C_{i+1} \in \mathcal{L}$. We perform the same procedure for estimating $\int_{x=x_1}^{x_2} (xf(x)dx)$, where $f(x)$ denotes $F_Y(x)$, $F_{X^-}(x)$, and $F_{Y^+}(x)$ in (3), (5), and (6), respectively.

3.2.2 Computing $\widetilde{\Pr}(\mathbf{Y})$

Similar to the procedures in Algorithm 2 for estimating the probability distribution for the loads of virtual servers (i.e., $\Pr(\mathbf{Y})$), each peer i randomly selects $\hat{\varphi} \ln^2 m$ virtual servers in the network, where $m = |V|$. By referring to the load values of the sampled virtual servers, i estimates the probability distribution for the load values of virtual servers in the system.

3.3 Computing $F_{X^-}(x)$ and $F_{Y^+}(y)$

As mentioned in Section 3.1, $F_{X^-}(x)$ represents the estimated probability distribution for the available capacities of the underloaded peers in the system. In our design, we simply estimate $F_{X^-}(x)$ based on the set of samples $\mathcal{I}$. More specifically, each peer i performs the same procedures as those in Algorithm 2 for computing $F_{X^-}(x)$ with the given input samples:

$$\mathcal{I}_C = \left\{ k | T_k - \sum_{j \in V_k} L_j > 0, \quad \forall k \in \mathcal{I} \right\}. \tag{7}$$

Similarly, each peer i approximates $F_{Y^+}(y)$, the probability distribution of the loads of the virtual servers selected for migration in the system. i computes $F_{Y^+}(y)$ with the samples given in the following:

$$\mathcal{I}_L = \left\{ \mathcal{M}_k | T_k - \sum_{j \in V_k} L_j < 0, \quad \forall k \in \mathcal{I} \right\}, \tag{8}$$

where $\mathcal{M}_k \subseteq V_k$ is the set of virtual servers chosen by an overloaded peer, k, for migration. Notably, our study intends to minimize the total movement cost, MC, as defined by (2). Hence, k will select a subset of its virtual servers with the minimum total load, such that $T_k \geq \sum_{j \in V_k} L_j - \sum_{l \in \mathcal{M}_k} L_l$. Particularly, $\mathcal{M}_k$ includes the virtual servers with the top $|\mathcal{M}_k|$ minimum loads in V_k in our implementation.

3.4 Estimating $|N|$ and $|V|$

As $m = n \ln n$ due to [14], [15], it suffices to estimate m and, in turn, to approximate n. Denote $\widetilde{m}$ as the approximation for m. Similar to Algorithm 2, our proposal computes $\widetilde{m}$ as follows: Let $\mu' = \frac{|\mathcal{R}|}{|\mathcal{H}|}$, where $\mathcal{H}$ is the entire hash space in the system, such that any virtual server has a unique ID selected independently and uniformly at random from $\mathcal{H}$, and $\mathcal{R} \subset \mathcal{H}$ is a hash subspace. Our proposal samples $\varphi' \ln^2 |\mathcal{H}|$ virtual servers, which are denoted by set $\mathcal{J}$ and each selected uniformly at random from V. Based on $\mathcal{J}$, we approximate μ' by computing $\gamma' = \frac{|\mathcal{J}'|}{\varphi' \ln^2 |\mathcal{H}|}$, where $\mathcal{J}' \subset \mathcal{J}$ is the set of the sampled peers whose IDs are in $\mathcal{R}$. Consequently, $\widetilde{m} = \frac{|\mathcal{J}'|}{\gamma'}$ is the estimation for m.

Corollary 1, which is shown as follows, states the quality of our proposal in estimating m. As its proof is similar to that of Theorem 1, we do not include its details.

Corollary 1. *Let $\mu' = \frac{3\rho'}{\delta'^2 \ln |\mathcal{H}|}$ be the fraction of virtual servers with IDs in any hash subspace $\mathcal{R}$ in the size of $\mu'|\mathcal{H}|$, where $0 < \delta' < 1$. Let γ', which our proposal estimates using $\varphi' \ln^2 |\mathcal{H}|$ samples, be the fraction of virtual servers with their IDs in $\mathcal{R}$. Our proposal then guarantees $\Pr(|\gamma' - \mu'| \leq \delta'\mu') > 1 - 2(\frac{1}{|\mathcal{H}|})^{\rho'\varphi'}$. Specifically, let k be the number of virtual servers whose IDs are in $\mathcal{R}$ in the set of the $\varphi' \ln^2 |\mathcal{H}|$ samples. With the probability $> 1 - 2(\frac{1}{|\mathcal{H}|})^{\rho'\varphi'}$,*

$$\frac{k}{(1+\delta')\mu'} < m < \frac{k}{(1-\delta')\mu'}. \tag{9}$$

Notably, let $|\mathcal{H}| = 2^\ell$ (e.g., $\ell = 160$ in Chord [1] with the SHA-1 hash function [16]). Sampling at least $\varphi' \ln^2 |\mathcal{H}| = \Theta(\frac{\ell \ln \varepsilon}{\rho'})$ virtual servers suffices, such that (9) holds in a probability of $> 1 - \frac{1}{\varepsilon}$.

4 Simulations

4.1 Experimental Setting

We have developed an event-driven simulator to assess the performance of our proposal. Our simulator implements the Chord DHT protocol [1]. In default, we simulate $n = |N| =$ 10,000 Chord peers. In the 10,000-node Chord network, we deploy $m = |V| = n \ln n$ virtual servers [14], [15], with each initially assigned to a peer selected uniformly at random from N.

In addition to our proposal presented in this paper, we include in our simulator the many-to-many approach with a single directory [3] and the tree-based approach [4] for comparison. Notably, our discussion in this section mainly considers the static scenario without peers joining and leaving. We refer interested readers to Appendix C for the performance results of our proposal operating in a dynamic

TABLE 4
The Ideal LBF in the Simulations

(capacities of peers, loads of virtual servers)	Ideal LBF
(Pareto, Pareto) (or $(\mathbf{P},\mathbf{P})$)	9.87
(Pareto, Gaussian) (or $(\mathbf{P},\mathbf{G})$)	5.01

environment. Appendix C also details the effects of varying system parameters relevant to our proposal.

Similar to prior studies (e.g., [4], [5]), the capacity of a simulated peer in our simulations follows the power-law distribution, that is, the Pareto distribution, with the default shape parameter of 2 in accordance with [5]. This represents that a small number of capable peers are present in the system. The load value of a simulated virtual server also has the Pareto distribution, indicating that a few virtual servers in the system serve popular (or large) data objects. As the default probability distributions for the capacities of peers and the loads of virtual servers both follow the power-law distribution, we also investigate the scenario where the loads of virtual servers have the Gaussian distribution [4].

Table 4 presents the "ideal" load imbalance factor, i.e.,

$$LBF = \frac{\sum_{i\in N}\sum_{v\in V_i} L_v}{\sum_{i\in N} C_i},$$

given the set of peers, N, and the set of virtual servers, $V = \cup_{i\in N} V_i$ for the different combinations of the probability distributions of the capacities of peers and the loads of virtual servers in the simulations. For example, the ideal LBF is equal to 5.01 in case the capacities of peers and the load of virtual servers follow the Pareto and Gaussian distributions (denoted by $(\mathbf{P},\mathbf{G})$ in Table 4), respectively.

The major performance metrics we measure are as follows:

- *Load imbalance factor*: The load imbalance factor for node $i \in N$ is defined as

$$LBF_i = \frac{\sum_{v\in V_i} L_v}{C_i}.$$

- *Movement cost*: Given the set of peers, N, and the set of virtual servers, V_i, for each peer $i \in N$, peer i migrates a subset of its virtual servers, V_i' (where $V_i' \subset V_i$), to other peers. The movement cost as defined by (2) denotes $\sum_{v\in S} L_v$, where $S = \cup_{i\in N} V_i'$.
- *Protocol message overhead*: The protocol message overhead represents the protocol messages required due to the load balancing algorithms. We simulate the details of the Chord routing protocol, a detailed discussion of which will be presented in Section 4.2.3.

Regarding our proposal in the simulations, first we note that as our proposal has each peer, i, issuing a random walker to the sampled nodes (virtual servers) from the network, the walker associated with i takes k walk steps, and among the k visited nodes (k' virtual servers), i only collects the last $\varphi \ln^2 n$ nodes ($\hat{\varphi}\ln^2 m$ virtual servers) as its samples if $\varphi \ln^2 n \le k$ ($\hat{\varphi}\ln^2 m \le k'$). Otherwise, the number of the sampled nodes (virtual servers) is k (k'). $k = 100$ is the default.

Second, for each peer implementing our proposal, $\varphi = 1$, $\hat{\varphi} = 1$, $\delta = .99$, $\hat{\delta} = .99$, $\rho = \frac{\ln 2 + \ln 10}{\varphi \ln n}$, and $\hat{\rho} = \frac{\ln 2 + \ln 10}{\hat{\varphi}\ln m}$ in the simulations, where ρ and $\hat{\rho}$ depend on n and m, respectively. To estimate n and m, each simulated peer in our proposal implements the technique discussed in Section 3.4. Specifically, by Corollay 1, we let $|\mathcal{H}| = |N| = 10{,}000$, $\delta' = \frac{1}{2}$, and $\varepsilon = 10$, such that $\widetilde{m}$, which approximates m, is bounded from above by $\frac{3m}{2}$ and from below by $\frac{m}{2}$, with a probability of no less than $1 - \frac{1}{\varepsilon} = 0.9$.

Third, as mentioned in Section 3.1, each peer i in our proposal needs to estimate its target load $T_i = \bar{\mathcal{A}} \times C_i + \epsilon$. In our simulations, the default value of ϵ is 0.

Finally, our algorithm operates periodically (see Section 3.1). In the static environment investigated in our simulations, each participating peer performs our algorithm for a single round in default.

4.2 Comparative Study

4.2.1 Load Imbalance Factor

Fig. 2 depicts the simulation results for the load imbalance factor, where the result for each investigated algorithm is averaged from 100 runs. While `centralized directory` and `tree` in Fig. 2, respectively, represent the many-to-many approach [3] and the tree-based approach [4], `w/o`

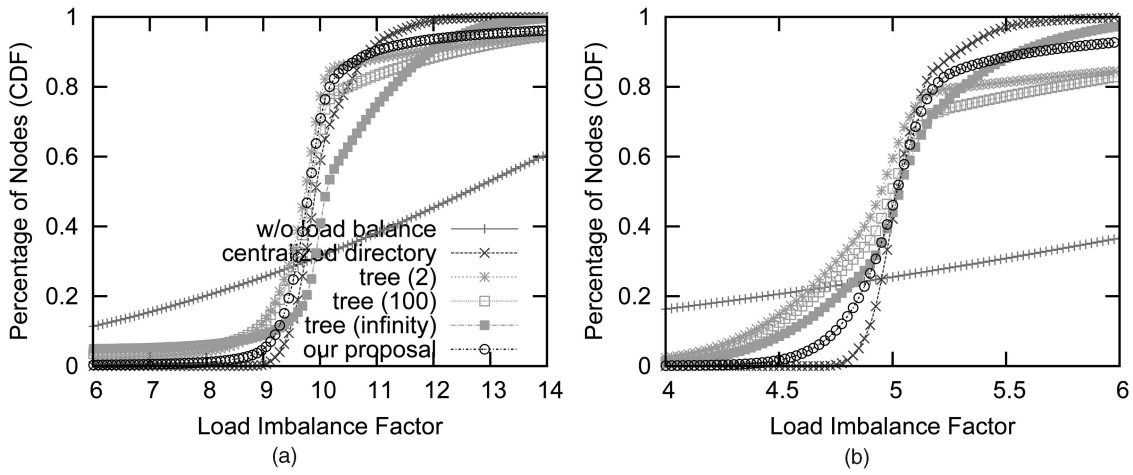


Fig. 2. Load imbalance factor. (a) $(\mathbf{P},\mathbf{P})$. (b) $(\mathbf{P},\mathbf{G})$.

load balance denotes the system that is not optimized by any load balancing algorithm. For comparison, in our simulations, centralized directory employs a single directory node, which maintains the global knowledge of the entire system, to pair the participating peers and the virtual servers.

In contrast, in the decentralized tree-based approach, each internal node, i, of the tree has a predefined pairing threshold t (here, $t = 2, 100, \infty$ in our simulations). If the distinct virtual servers information, S, collected by the rendezvous peer i reaches t, then i pairs the underloaded peers known to i and the virtual servers in S. For details, we refer interested readers to [4]. Notably, by $t = \infty$, we indicate that only the root node performs the reallocation of virtual servers for all offspring nodes in the tree. Furthermore, in the simulated tree-based networks, we have each internal node in the trees maintain two children nodes.

The simulation results in Fig. 2 illustrate that centralized directory performs the best among the investigated algorithms in terms of the load imbalance factor. This is because the directory node in centralized directory maintains the global information of the entire system, and for each virtual server v selected to be reallocated from the overloaded peers, the directory node seeks to find a peer j among all light peers to host v by brute-force search, such that the resultant load imbalance factor of j (i.e., LBF_j) is minimized [3]. Notably, centralized directory can precisely compute

$$\mathcal{A} = \frac{\sum_{v \in V} L_v}{\sum_{\in N} C_i}$$

and can then identify whether a peer is overloaded or not. While centralized directory performs the best, ours performs very well, and is comparable with centralized directory, particularly in the practical setting $(\mathbf{P}, \mathbf{P})$ (Fig. 2a). This reveals that 1) our proposal can provide an effective approximation for the probability distributions of the capacities of peers and the loads of virtual servers, and 2) aligning such two probability distributions can effectively lead to a systemwide load-balanced state.

As shown in Fig. 2, tree performs fairly. Even each peer in tree has the accuracy information of $\mathcal{A}$, and it can exactly determine whether it is underloaded or not. In the tree-based approach, a rendezvous peer implements the best-fit scheme to pair underloaded peers and virtual servers [4]. More precisely, for the unpaired virtual servers known to a rendezvous peer i in the tree, from the heaviest to the lightest, i seeks a peer j, such that allocating a virtual server to j minimizes j's remaining capacity. As a result, the remaining capacity of j may not further host any virtual servers. This is particularly true in tree (∞), where the overloaded peers, whose load imbalance factors are >9.87 in Fig. 2a and >5.01 in Fig. 2b, cannot manage their loads well.

4.2.2 Movement Cost

Fig. 3 shows the movement costs due to centralized directory, tree, and our proposal. In Fig. 3, the movement costs of tree and our proposal are normalized to that of centralized directory. Centralized directory performs best in terms of the load imbalance factor.

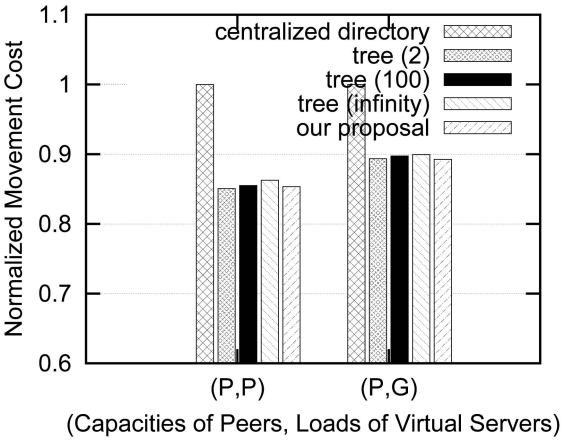


Fig. 3. Movement cost.

However, this is at the expense of a greater movement cost. In contrast, tree and our proposal have a comparable movement cost less than that of centralized directory.

4.2.3 Protocol Message Overhead

The load balancing algorithms we investigate in this paper introduce the protocol message overhead. Let us consider centralized directory. The protocol message overhead includes the messages introduced by each peer, i, to update the directory on the knowledge regarding i's capacity and the load of each i's local virtual servers. In addition, the message overhead comprises those generated by the directory to inform the overloaded peers which of the peers can share loads. For fairness in the comparison among the load balancing algorithms, in our implementation of centralized directory, each peer issues independent routes each for its local virtual servers in the DHT network to update the directory, while the directory then informs the overloaded peers though end-to-end messages where to migrate their loads.

Unlike in centralized directory, in tree, any two peers (e.g., peers i and j) with a link in the tree network communicate with one another by routing in the Chord substrate. Let i be the parent of j in the tree. j reports the total load of its virtual servers to i. i can then summarize its load and the loads of its children nodes, and in turn it reports the aggregated load to its parent. Once the root node computes $\mathcal{A}$, the average load per unit capacity a peer should perceive, the root node disseminates such knowledge to nodes elsewhere in the system through the tree network. Thereafter, each peer, k, identifies whether it is overloaded. If k is overloaded, k propagates information regarding its virtual servers designated for migration; otherwise, k simply reports its remaining capacity (i.e., k is an underloaded peer). Again depending on the tree network, such knowledge is sent through the tree links toward the root node. An internal node of the tree pairs the virtual servers and the underloaded peers if its collected information reaches a predefined pairing threshold, as discussed in Section 4.2.1. We note in our simulations that an internal tree node informs an overloaded peer through end-to-end communication the destination peers that can share the load.

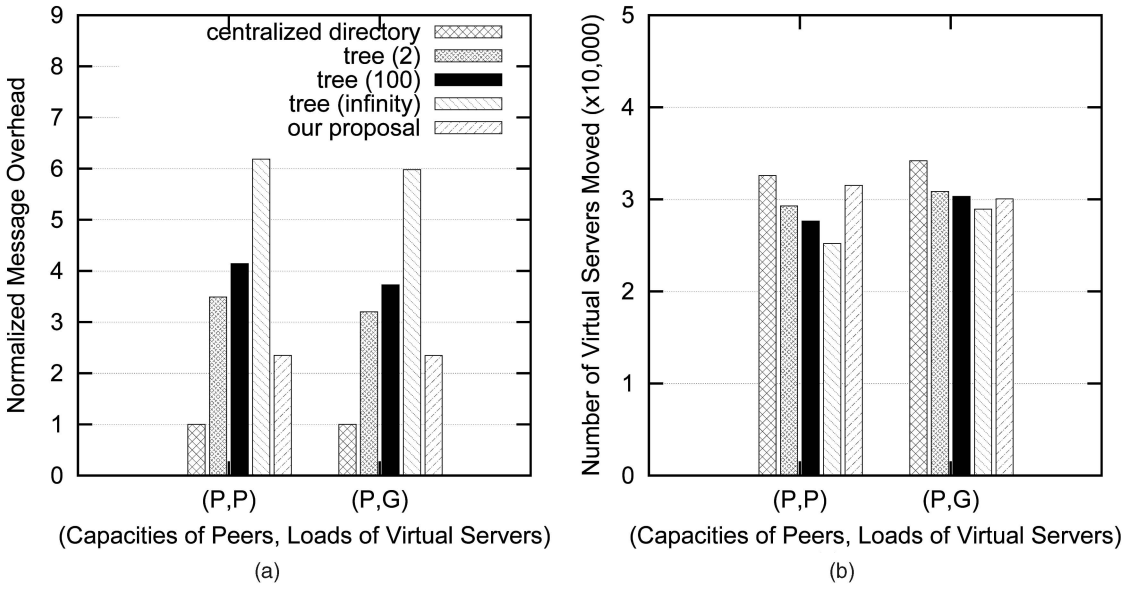


Fig. 4. (a) The total protocol message overhead, and (b) the number of virtual servers moved.

In contrast to `centralized directory` and `tree`, each peer in our proposal solely relies on random walk to estimate the system state (i.e., $\Pr(\mathbf{X})$ and $\Pr(\mathbf{Y})$). Each walk step performed by a walker in the Chord network takes one end-to-end message. When a walker visits a peer i (or a virtual server v), the walker replies to the peer creating the walker i's capacity information (or v's load information) by taking an end-to-end message.

Fig. 4a presents the simulation results of the total protocol message overhead. We notice that the overheads measured for the load balancing algorithms are normalized to that due to `centralized directory`. Obviously, `tree` (∞) performs the worst, as the root node of the tree aggregates the information regarding the capacities of all peers and the loads of all virtual servers through the tree network built on top of the simulated DHT. On the contrary, both `centralized directory` and our proposal take the overheads considerably less than those due to `tree` (2), `tree` (100), and `tree` (∞). Our proposal generates the protocol message overhead more than `centralized directory` because each peer implementing our proposal needs to collect (partial) knowledge regarding the system and to estimate the system state.

In addition, the protocol message overhead also comprises those due to the migration of virtual servers. In a DHT with virtual servers, migrating a virtual server is simulated by *leave* and *join* operations offered by the DHT, thus generating protocol messages for repairing the routing tables maintained by virtual servers in the network. As the leave and join operations often depend on the design of a DHT network, we measure instead the number of virtual servers moved due to the load balancing algorithms.

Fig. 4b illustrates the number of virtual servers moved as required by the studied load balancing algorithms. The number of virtual servers moved in `centralized directory` is slightly more than that generated by our proposal. As `tree` (∞) migrates less virtual servers for the scenario $(\mathbf{P}, \mathbf{P})$, it unbalances the loads of the peers (see Fig. 2).

With regard to the protocol message overhead, we further investigate the number of protocol messages received by each peer. Fig. 5 plots the simulation results for the scenario of $(\mathbf{P}, \mathbf{P})$, as we do not observe any clear difference from the

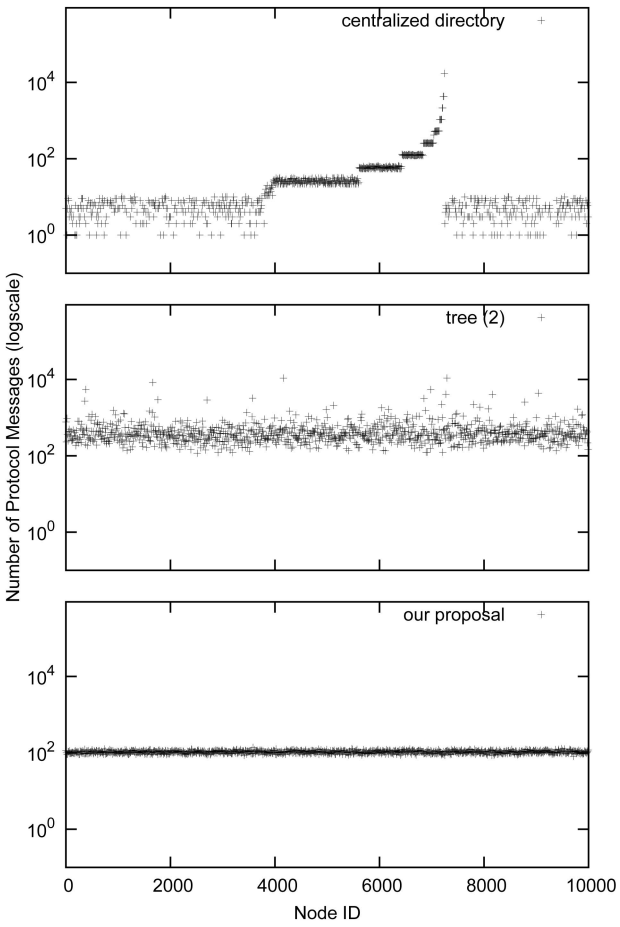


Fig. 5. The distribution of protocol messages for $(\mathbf{P}, \mathbf{P})$.

results of $(\mathbf{P}, \mathbf{G})$. As shown in Fig. 5, `centralized directory` introduces hotspots to the system. The directory node and the nodes near the directory in the overlay experience exponential numbers of protocol messages. Similarly, the tree-based approach also generates hot spots. In Fig. 5, the only tree-based approach, `tree` (2), is presented, as `tree` (100) has a performance result similar to that of `tree` (2), while `tree` (∞) degenerates to `centralized directory` in terms of the number of protocol messages received by each peer. We note that the hotspot nodes in the centralized directory and tree-based approaches will experience more workloads in computing the reallocation of the virtual servers compared with the nonhotspot nodes. Unlike `centralized directory` and `tree`, our proposal distributes the protocol messages evenly in the system, thus introducing nearly identical computational workloads to the participating peers in the reallocation of their virtual servers.

5 Summary

In this paper, we have presented a novel load balancing algorithm for DHTs with virtual severs. Our proposal is unique in that we represent the system state with probability distributions. Unlike prior solutions that often rely on global knowledge of the system, each peer in our proposal independently estimates the probability distributions for the capacities of participating peers and the loads of virtual servers based on partial knowledge of the system. With the approximated probability distributions, each peer identifies whether it is underloaded and then reallocates its loads if it is overloaded. Our proposal is driven by rigorous performance analysis and validated by extensive simulations. The simulation results reveal that that our proposal performs well and that it is comparable with the centralized directory approach and outperforms the tree-based solution in terms of the load imbalance factor, the movement cost of virtual servers, and/or the protocol message overhead. In particular, while the centralized directory and tree-based approaches introduce hotspots to the system, the participating peers in our proposal perceive the nearly identical workloads in manipulating our load balancing algorithm.

Appendix A

Sampling Based on Random Walk

As our proposal in estimating $\Pr(\mathbf{X})$ and $\Pr(\mathbf{Y})$ relies on sampling the participating peers and the virtual servers independently and uniformly at random from the network, we present a uniformly sampling technique based on the *Metropolis-Hastings method* [17], [18], in this section.

To sample peers and virtual servers uniformly at random from the network, our idea is to represent the DHT network with virtual servers as a graph G, in which the vertices represent the virtual servers, and the connection between two vertices v and u exists in G if v's routing table includes u as its neighbor. We perform *random walk* over G such that any virtual server in G is visited in the probability of $\frac{1}{m}$. Consequently, if a peer i has k virtual servers, then a random walker j would visit i with the probability of $\frac{k}{m}$. As a result, in our proposal, when j visits i, j selects i as a sample in the probability of $\frac{1}{k}$, leading to the participating peers being sampled uniformly. Thus, this is adequate to explain how our proposal samples virtual servers independently and uniformly at random from the network. Notably, each participating peer in our implementation issues a single random walker to sample peers and virtual servers simultaneously.

Denote a DHT with virtual servers by $G = (V, E)$, where V is the set of virtual servers in the system, and $E = \{(v, u) | v \in V_i \text{ links to } u \in V_j, \forall i \neq j \in N\}$. Each node $i \in N$ intends to sample a virtual server from V independently and uniformly at random. It works as follows: i arbitrarily selects a virtual server $v \in V_i$ to create a random walker. Let x be a virtual server in V that receives the walker. x then dispatches the walker to one (say y) of its "neighbors" in G, denoted by $\mathcal{G}_x$ ($\mathcal{G}_x = \{y | xy \in E\}$), according to the following probability:

$$\Pr_{x,y} = \begin{cases} \phi T_x \min\left(1, \frac{T_y}{T_x}\right), & \text{if } y \in \mathcal{G}_x, \\ 0, & \text{if } y \notin \mathcal{G}_x, \\ 1 - \sum_{w \neq x} \Pr_{x,w}, & \text{if} x = y, \end{cases} \tag{10}$$

and

$$T_x = \frac{1}{deg_x}, \tag{11}$$

where ϕ is an arbitrarily positive value, and $0 < \phi < 1$ such that $1 - \sum_{w \neq x} \Pr_{x,w} > 0$. $deg_x = |\{(x, y) | (x, y) \in E\}|$ is the number of outgoing connections maintained by the virtual server x.

Notably, (10) allows the uniform probability distribution π, such that v selects any virtual server w from V with the probability of $\pi(w) = \frac{1}{m}$. This is because 1) the Markov chain defined by the probability transition matrix $[\Pr_{x,y}]$ is aperiodic (each virtual server has a positive probability to select itself), 2) irreducible (there is a positive probability to pick any virtual server from V), and 3) time reversible ($\pi(x)\Pr_{x,y} = \pi(y)\Pr_{y,x}$ for all $x \neq y \in V$) [19]. It follows that $\pi(x) = \pi(y)$ as $\Pr_{x,y} = \Pr_{y,x}$, and thus $\pi(x) = \pi(y) = \frac{1}{m}$.

As the Markov chain defined by (10) allows i to sample any virtual server w with the intended probability of $\pi(w) = \frac{1}{m}$, i takes time to approximate the uniform probability distribution π. Particularly, in our proposal, a node, i, issues a random walker, say j, to sample virtual servers. j visits the virtual servers in the system and updates the set of samples maintained by i. If the size of the samples reaches a predefined threshold, for example, $\ln^2 m$ or $\ln^3 m$ (see Section 3.2), i replaces its samples in a FIFO order. Once π is approximated, the virtual servers are sampled by j with the probability close to the intended. In the following, we analyze the time steps required by j to approach the uniform distribution π.

We first define the following notations:

Definition 1. *The variation distance between two probability distributions D_1 and D_2 on a countable state space S is given by*

$$\|D_1 - D_2\| = \frac{\sum_{x \in S} |D_1(x) - D_2(x)|}{2}, \tag{12}$$

where the factor $\frac{1}{2}$ guarantees that the variation distance is between 0 and 1.

Definition 2. *Let D be the stationary probability distribution of a Markov chain with state space S. Let D_x^t be the probability distribution of the state of the chain starting at x after t time steps. Given any small constant ϵ, we define the mixing time of the Markov chain as*

$$\tau(\epsilon) = \max_{x \in S} \tau_x(\epsilon), \tag{13}$$

where

$$\tau_x(\epsilon) = \min\left\{t : \left\|D_x^t - D\right\| \le \epsilon\right\}. \tag{14}$$

By Definition 2, $\tau_x(\epsilon)$ is the first time step t at which the variation distance between D_x^t and D is no more than ϵ. Among $\tau_x(\epsilon)$s, $\tau(\epsilon)$ is the maximum over all the xs in S. If $\tau(\epsilon)$ is polynomial in the problem size (i.e., m) and $\ln(\epsilon^{-1})$, then the Markov chain is *rapidly mixing* [20].

Theorem 2. *Given $G = (V, E)$, in our proposal, the random walker j initiated by any node $i \in N$ takes*

$$\tau(\epsilon) = O(m \cdot deg_{\max} \cdot \mathcal{D} \cdot (\ln m - \ln \epsilon)) \tag{15}$$

to approximate the uniform probability distribution π with the variation distance ϵ, where $\mathcal{D}$ represents the diameter of G, and $deg_{\max}$ is the maximum degree of a virtual server in V.

Proof. Let $\mathcal{MC}$ be any Markov chain with the state transition matrix $[\Pr_{x,y}]$ and the uniform probability distribution π. Denote $\pi_{\min}$ as the smallest stationary probability of visiting a state in $\mathcal{MC}$. Denote $Q(e) = Q(s_1 s_2) = \pi(s_1)\Pr_{s_1,s_2}$ as the conditional probability for the state transition from s_1 to s_2 in $\mathcal{MC}$. Let Γ be a collection of simple paths γ_{xy} between all distinct states x and y in $\mathcal{MC}$, that is, $\Gamma = \{\gamma_{xy} | \forall x \ne y\}$. For a given Γ, define $\rho(\Gamma) = \max_e \frac{1}{Q(e)} \sum_{\gamma_{xy} \ni e} (\pi(x)\pi(y))$, and $\mathcal{D}(\Gamma)$ is the maximal number of edges connecting any two states in Γ. Then, due to [21], [22], $\tau(\epsilon) \le \rho(\Gamma)\mathcal{D}(\Gamma)\ln(\frac{1}{\pi_{\min}\epsilon})$ for any Γ.

Consequently, in our proposal, since $Q(e) \ge \frac{\phi}{m \cdot deg_{\max}}$ (where $\pi_{\min} = \frac{1}{m}$), $\mathcal{D}(\Gamma) \le \mathcal{D}$ and

$$\begin{aligned}\rho(\Gamma) &= \max_e \frac{1}{Q(e)} \sum_{\gamma_{xy} \ni e} (\pi(x)\pi(y)) \\ &\le \frac{m \cdot deg_{\max}}{\phi} \cdot \left(\sum_{x \in V} \pi(x)\right)^2 \\ &= \frac{m \cdot deg_{\max}}{\phi},\end{aligned}$$

the proof follows. □

Consider a DHT network with virtual servers, $G = (V, E)$. Theorem 2 shows that peer i samples the virtual servers in G in a rapid fashion. This is because a typical G has a small diameter $\mathcal{D}$. More specifically, as suggested by [14], a DHT network $G = (V, E)$ with $|V| = m$ virtual servers has a diameter of $O(\ln m)$, and the maximum degree of a virtual server in G is $O(\ln m)$.

APPENDIX B

THEOREM 1

Proof. Let X_i, where $1 \le i \le \varphi \ln^2 n$, be the indicator random variable, where $X_i = 1$ if the ith node picked independently and uniformly at random has the capacity value in $[C_l, C_u]$; $X_i = 0$, otherwise. Let $X = \sum_{i=1}^{\varphi \ln^2 n} X_i$. By the *linearity of expectations*, the expected fraction of sampled peers with the capacity values in $[C_l, C_u]$ is

$$\begin{aligned}\mathbf{E}[\gamma] &= \mathbf{E}\left[\frac{X}{\varphi \ln^2 n}\right] \\ &= \frac{1}{\varphi \ln^2 n} \mathbf{E}\left[\sum_{i=1}^{\varphi \ln^2 n} X_i\right] \\ &= \mu.\end{aligned}$$

Let $t = (\varphi \ln^2 n)\ln(1+\delta)$. As X_i $(1 \le i \le \varphi \ln^2 n)$ are independent and

$$\Pr(\gamma \ge (1+\delta)\mu) = \Pr\left(e^{\gamma t} \ge e^{(1+\delta)\mu t}\right),$$

by *Markov's inequality*, we have the *Chernoff bound* [23] as follows:

$$\begin{aligned}\Pr(\gamma \ge (1+\delta)\mu) &= \Pr\left(e^{\gamma t} \ge e^{(1+\delta)\mu t}\right) \\ &\le \frac{\mathbf{E}[e^{\gamma t}]}{e^{(1+\delta)\mu t}} \\ &= \frac{\mathbf{E}\left[e^{\frac{\sum_{i=1}^{\varphi \ln^2 n} X_i}{\varphi \ln^2 n} \cdot t}\right]}{e^{(1+\delta)\mu t}} \\ &= \frac{\prod_{i=1}^{\varphi \ln^2 n} \mathbf{E}\left[e^{\frac{X_i}{\varphi \ln^2 n} \cdot t}\right]}{e^{(1+\delta)\mu t}}.\end{aligned}$$

Because $1 + y \le e^y$ for any y, we have

$$\begin{aligned}\mathbf{E}\left[e^{\frac{X_i}{\varphi \ln^2 n} \cdot t}\right] &= \mu \cdot e^{\frac{t}{\varphi \ln^2 n}} + (1 - \mu) \\ &= 1 + \mu\left(e^{\frac{t}{\varphi \ln^2 n}} - 1\right) \\ &\le e^{\mu\left(e^{\frac{t}{\varphi \ln^2 n}} - 1\right)}.\end{aligned}$$

Thus,

$$\begin{aligned}\Pr(\gamma \ge (1+\delta)\mu) &\le \frac{\prod_{i=1}^{\varphi \ln^2 n} \mathbf{E}\left[e^{\frac{X_i}{\varphi \ln^2 n} t}\right]}{e^{(1+\delta)\mu t}} \\ &\le \frac{\prod_{i=1}^{\varphi \ln^2 n} e^{\mu\left(e^{\frac{t}{\varphi \ln^2 n}} - 1\right)}}{e^{(1+\delta)\mu t}} \\ &= \frac{e^{\left(e^{\frac{t}{\varphi \ln^2 n}} - 1\right)\mu \varphi \ln^2 n}}{e^{(1+\delta)\mu t}} \\ &= \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu \varphi \ln^2 n}.\end{aligned}$$

For $0 < \delta < 1$, since $\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \le e^{\frac{-\delta^2}{3}}$ (this is because $\delta - (1+\delta)\ln(1+\delta) + \frac{\delta^2}{3} \le 0$ for $\delta \in [0, 1]$), we then have

$$\begin{aligned}\Pr(\gamma \ge (1+\delta)\mu) &\le \left(e^{\frac{-\delta^2}{3}}\right)^{\mu \varphi \ln^2 n} \\ &= \left(\frac{1}{n}\right)^{\rho \varphi}.\end{aligned} \tag{16}$$

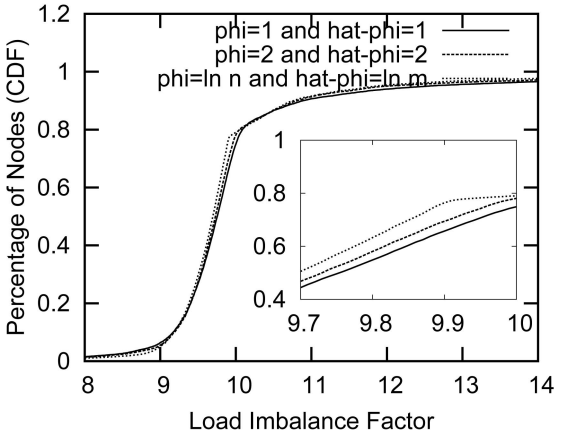


Fig. 6. The effect of varying φ and $\hat{\varphi}$ for $(\mathbf{P},\mathbf{P})$ (we let $\varphi=\hat{\varphi}$ in this experiment).

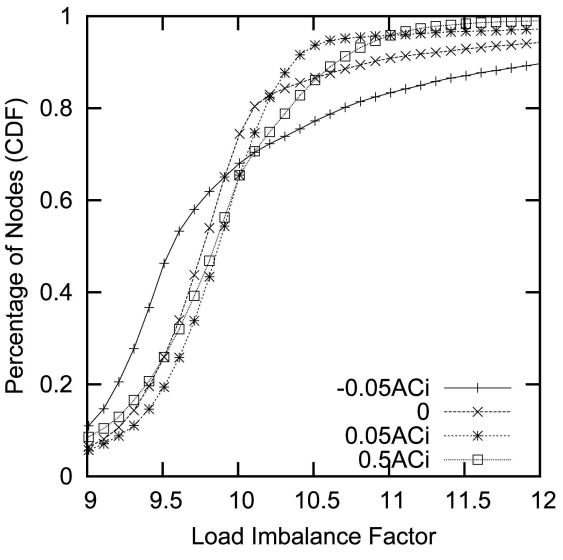
Similarly, let $t'=(\varphi\ln^2 n)\ln(1-\delta)<0$, where $0<\delta<1$. Hence, $\Pr(\gamma\le(1-\delta)\mu)=\Pr(e^{\gamma t'}\ge e^{(1-\delta)\mu t'})$. Applying Markov's inequality yields the following:

$$\begin{aligned}\Pr(\gamma\le(1-\delta)\mu)&=\Pr\left(e^{\gamma t'}\ge e^{(1-\delta)\mu t'}\right)\\&\le\frac{\mathbf{E}\left[e^{\gamma t'}\right]}{e^{(1-\delta)\mu t'}}\\&=\frac{\prod_{i=1}^{\varphi\ln^2 n}\mathbf{E}\left[e^{\frac{X_i}{\varphi\ln^2 n}\cdot t'}\right]}{e^{(1-\delta)\mu t'}}\\&\le\frac{e^{\left(\frac{t'}{e^{\varphi\ln^2 n}}-1\right)\mu\varphi\ln^2 n}}{e^{(1-\delta)\mu t'}}\\&=\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu\varphi\ln^2 n}.\end{aligned}$$

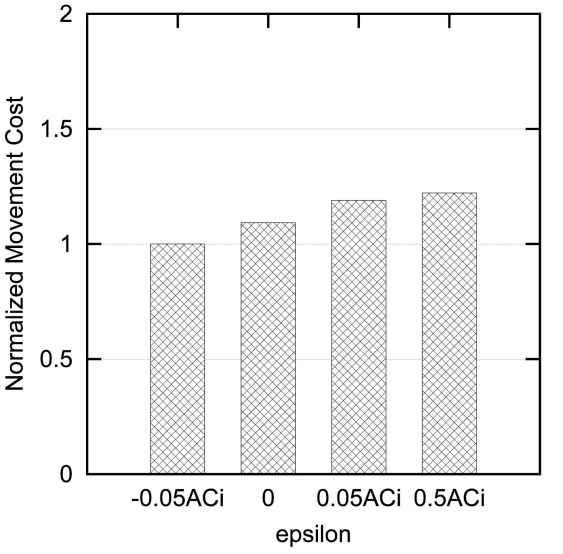
As $0<\delta<1$, we have $\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\le e^{\frac{-\delta^2}{3}}$, yielding

$$\begin{aligned}\Pr(\gamma\le(1-\delta)\mu)&\le\left(e^{\frac{-\delta^2}{3}}\right)^{\mu\varphi\ln^2 n}\\&=\left(\frac{1}{n}\right)^{\rho\varphi}.\end{aligned}\tag{17}$$

Combining (16) and (17) proves the theorem. □



(a)



(b)

Fig. 7. The effect of varying ϵ from -0.05 to $0.5\mathcal{A}C_i$ for $(\mathbf{P},\mathbf{P})$, where (a) is the load imbalance factor, and (b) is the movement cost (normalized to that of configuring $\epsilon=-0.05T_i$).

APPENDIX C
FURTHER PERFORMANCE RESULTS FOR OUR PROPOSAL

C.1 The Effects of System Parameters

In this section, we study the effects of the system parameters relevant to our proposal.

C.1.1 The Effect of Varying φ and $\hat{\varphi}$

Theorem 1 states that the approximation errors of the estimated probability distributions for the capacities of nodes and the loads of virtual servers decrease if we increase the numbers of samples by enlarging φ and $\hat{\varphi}$, respectively. Fig. 6 depicts the simulations results for the scenario of $(\mathbf{P},\mathbf{P})$, where $\varphi=k$ $(\hat{\varphi}=\hat{k})$ represents $k\ln^2 n$ $(\hat{k}\ln^2 m)$ samples for the peers (virtual servers). As we can see from Fig. 6, increasing the number of samples from $\ln^2 n$ $(\ln^2 m)$ to $\ln^3 n$ $(\ln^3 m)$ modestly improves the load imbalance factor.

C.1.2 The Effect of Varying ϵ

In our proposal, each peer i estimates its target load $T_i=\mathcal{A}\times C_i+\epsilon$ (ϵ is 0 in default in our simulations), where $\mathcal{A}$ is estimated based on i's approximated probability distributions for the capacities of peers and the loads of virtual servers. i may vary T_i by adapting ϵ. Specifically, by slightly increasing ϵ, i may accept excess load such that the reallocation of virtual servers becomes more flexible, resulting in increasing the number of candidate peers that can host a designated virtual server.

Fig. 7 plots the simulation results by having each peer i configure its $\epsilon=-0.05\mathcal{A}C_i, 0, 0.05\mathcal{A}C_i, 0.5\mathcal{A}C_i$, such that the resultant T_is become $0.95\mathcal{A}C_i$, $\mathcal{A}C_i$, $1.05\mathcal{A}C_i$, and $1.5\mathcal{A}C_i$,

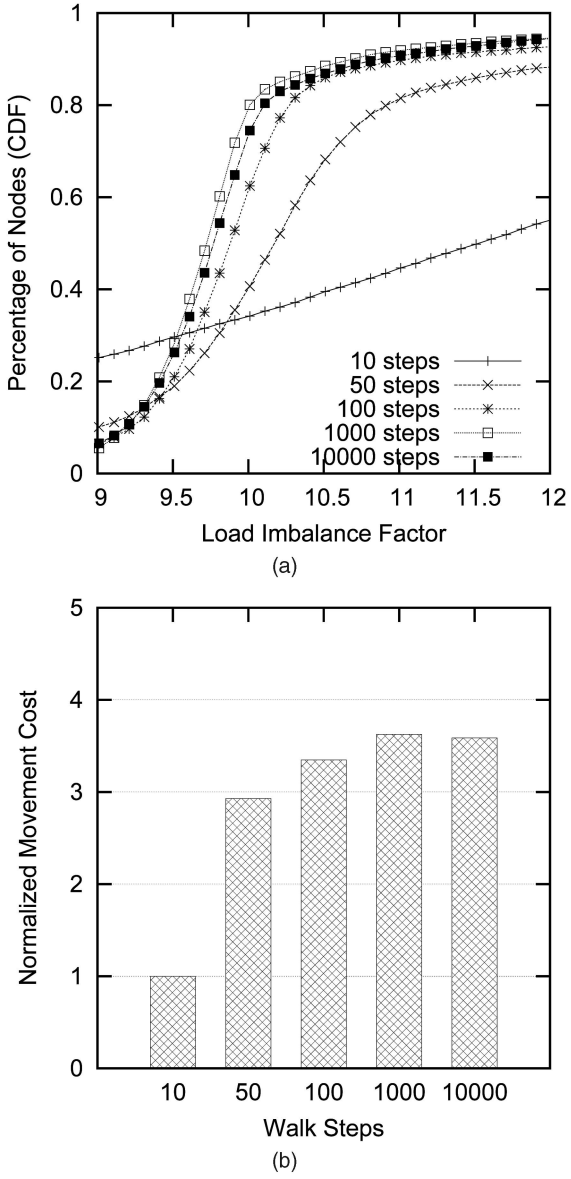


Fig. 8. The effect of varying the number of random walk steps from 10 to 10,000 for $(\mathbf{P},\mathbf{P})$, where (a) the load imbalance factor, and (b) the movement cost (normalized to that of implementing 10 walk steps).

respectively. As we can see from Fig. 7, slightly increasing ϵ improves the load imbalance factor (see $0.05\mathcal{A}C_i$ in Fig. 7a). However, this increases the movement cost compared with that with $\epsilon = 0$ (Fig. 7b).

C.1.3 The Effect of Varying Random Walk Steps

Theorem 2 concludes that the DHT network allows a rapidly mixing time in sampling nodes uniformly at random from the network. In our proposal, each peer needs to issue a random walker to sample peers and virtual servers from the system. The rapidly mixing time implies that each walker need not take considerable time to traverse the network until the peers and virtual servers can be (almost) guaranteed to be sampled independently and uniformly at random from the network.

Fig. 8 illustrates the effect of the different numbers of walk steps for our proposal. Here, the number of walk steps varies from 10 to 10,000. The simulation results indicate that our proposal with 100 walk steps suffices in offering the load imbalance factor comparable with that of performing 1,000 (or 10,000) walk steps (Fig. 8a), validating the effectiveness and efficiency of our design. Fig. 8b additionally depicts that the movement cost with 100 walk steps is approximate to that with more walk steps.

C.1.4 The Effect of Different Algorithmic Rounds

`centralized directory` and `tree` are deterministic, and they cannot improve the load imbalance factors of the participating peers with more algorithmic rounds if the system is static. By a static system, we mean that N, V, the capacities of the peers in N, and the loads of the virtual servers in V are not changed over time. In contrast, even the system remains static. A participating peer in our proposal may improve its load imbalance factor with more rounds of Algorithm 1 (Section 3.1). In this section, we study the effect of the different algorithmic rounds for our proposal. We notice that for each round, each peer issues a random walker that takes 100 walk steps to estimate the system state (i.e., $\Pr(\mathbf{X})$ and $\Pr(\mathbf{Y})$).

Fig. 9 shows the simulation results by varying the algorithmic rounds performed by each peer from 1 to 50. The results indicate that more algorithm rounds do help improve the load imbalance factor. However, our proposal with a few algorithmic rounds (e.g., 10 rounds in Fig. 9) suffices in offering the load imbalance factor comparable with that with a large number of rounds (e.g., 50 rounds in Fig. 9). Specifically in the scenario of $(\mathbf{P},\mathbf{P})$, our proposal performs reasonably well even with a single algorithmic round, demonstrating that the participating peers can effectively approximate $\Pr(\mathbf{X})$ and $\Pr(\mathbf{Y})$ to reallocate their virtual servers based on such approximation.

Figs. 10a and 10b, respectively, detail the total movement cost and the total number of virtual servers moved for "each" of the 50 algorithmic rounds of our proposal performed by the participating peers in the scenario of $(\mathbf{P},\mathbf{P})$. Notably, the x- and y-axis in Fig. 10 are in logarithmic scale. As we can see, when the system becomes better load balanced, our proposal takes less movement costs and migrates less virtual servers, indicating that our solution strives to exploit effectively the remaining capacities of underloaded peers without exceeding their target loads. Figs. 10a and 10b further illustrate that our load balancing algorithm converges rapidly in a logarithmic fashion by reducing the movement cost (by minimizing the number of virtual servers moved) from 10^6 (10^4) to 10^2 (10^1). Particularly, when the system is in a nearly load-balanced state, our proposal will introduce almost no movement cost in maintaining such a state. In Fig. 10, a total of $\approx 10^1$ virtual servers among $\approx 10^5$ ones, each with an absolute load value ≥ 10, in the system are moved due to the $50th$ round performed by *all* participants.

C.2 System Dynamics

This section investigates the performance of our proposal operating under a dynamic environment. We consider the scenarios that 1) the virtual servers change their loads without peers joining and departure, and 2) the participating peers join and leave. Notably, for the second scenario, we assume that each peer replicates each of its local virtual servers to its successor peer on the Chord ring. If a peer leaves the system, the successor peers can recover the virtual servers hosted by the departure peer (we refer the

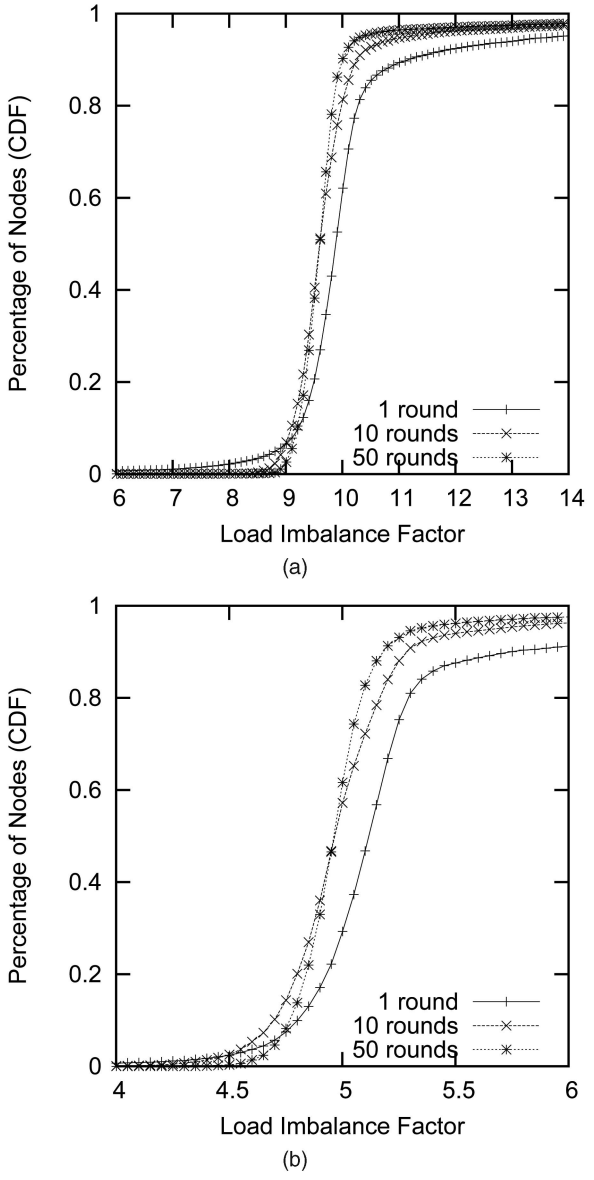


Fig. 9. The effect of varying the number of algorithmic rounds for (a) $(\mathbf{P}, \mathbf{P})$ and (b) $(\mathbf{P}, \mathbf{G})$.

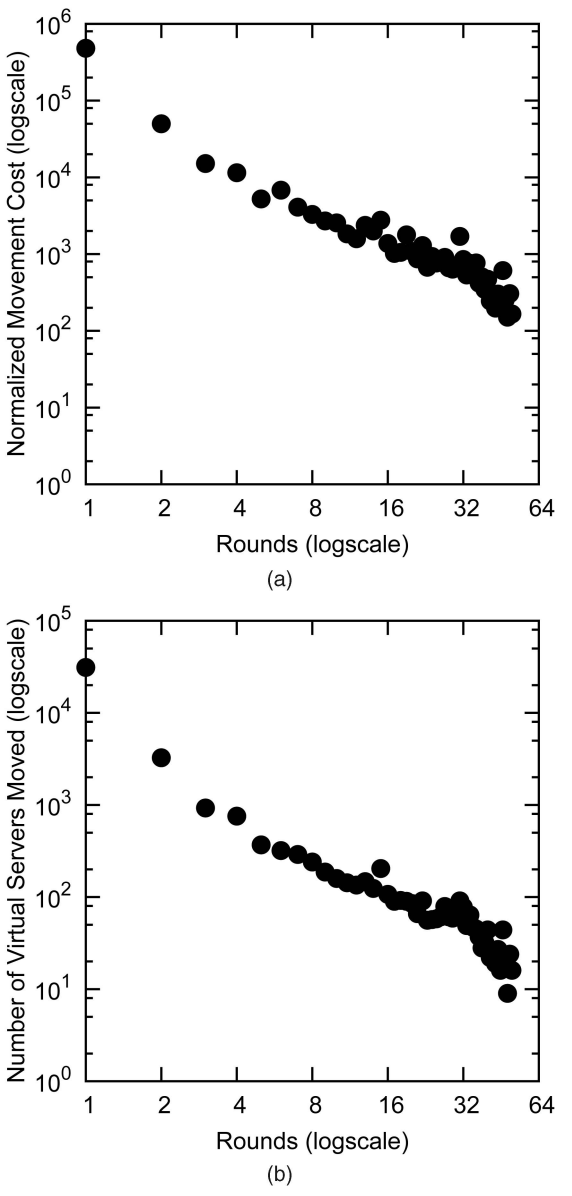


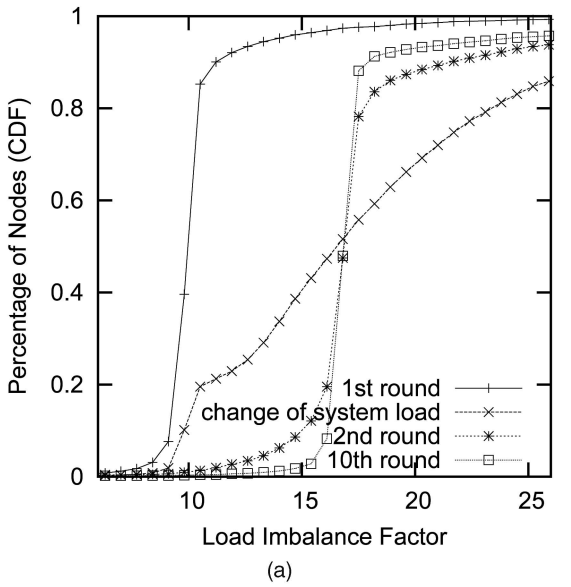
Fig. 10. (a) The movement cost and (b) the number of virtual servers moved by the 50 algorithmic rounds of our proposal for $(\mathbf{P}, \mathbf{P})$.

readers to [14] for the details regarding the replication of virtual servers). On the other hand, the newly coming peers join the system with the capacities following Pareto distribution, as mentioned in Section 4.1.

Fig. 11a demonstrates the simulation results for the loads of virtual servers varied over time. In this experiment, each peer performs our algorithm for 10 rounds. Before a peer performs its second round, each of its local virtual servers increases its load five times in a probability of 20 percent, resulting in the change of the ideal systemwide load imbalance factor from 9.87 to 17.59. In Fig. 11a, `change of system load` is the snapshot for the load imbalance factors of all the participating peers before they perform their second round of the load balancing algorithm. The simulation results show that our proposal adapts to the change of the system load well (see second and tenth `round` in Fig. 11a), and it can rebalance the loads among the participating peers.

Similarly, in Fig. 11b, the load of each virtual server is decreased to the ratio of $\frac{1}{5}$ in a probability of 40 percent, such that the resultant ideal load imbalance factor varies from 9.87 to 6.71. The results again demonstrate that our proposal adapts to the varied system load. In particular, our proposal is independent of the probability distribution of the loads of virtual servers. We observe the same performance result (the result is not shown in this paper) by varying the capacities of the participating peers over the system operational period, concluding that our solution is also independent of the probability distribution of the capacities of participating peers.

Fig. 12 depicts the simulation results for the dynamic scenario where participating peers join and leave. Each simulated peer has the expected lifetime of 100 minutes and 20 minutes, respectively (see Figs. 12a and 12b) [24]. In the experiments, each alive peer performs our proposal every 10 minutes, and it issues a random walker that takes a step



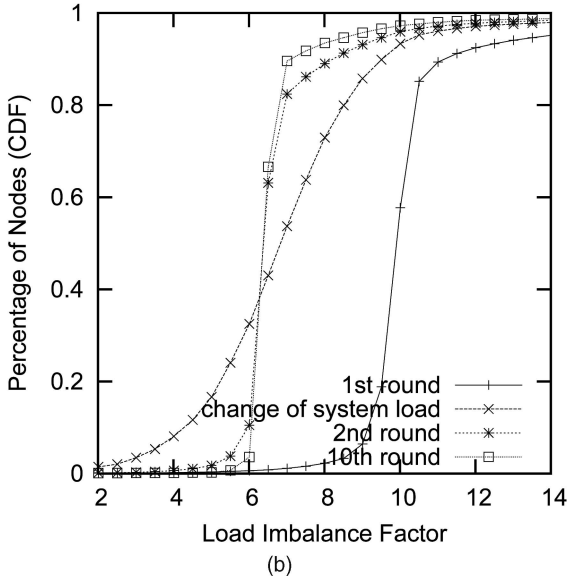
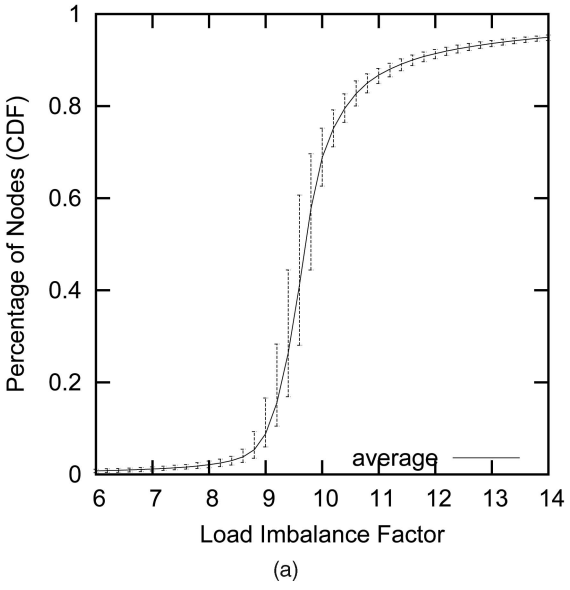


Fig. 11. The effect of varying the loads of the virtual servers for $(\mathbf{P}, \mathbf{P})$, where (a) each of the 20 percent of the randomly picked virtual servers, v, increases its load to $5L_v$, and (b) each of the 40 percent of the randomly picked virtual servers, v, decreases its load to $\frac{L_v}{5}$.



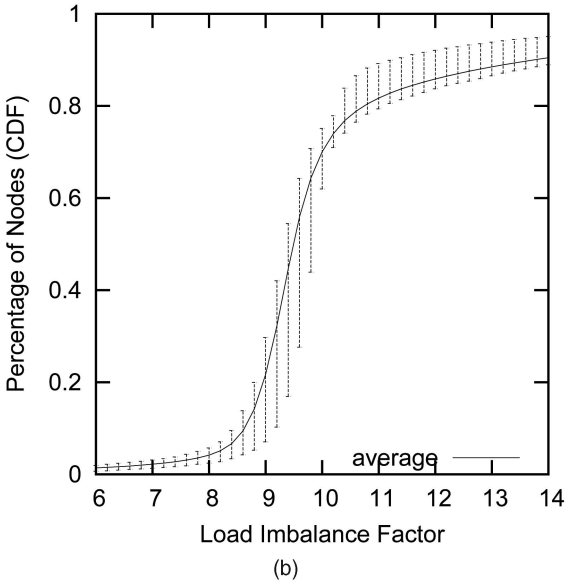


Fig. 12. The effect of peers joining and departure, where (a) the expected peer lifetime is 100 minutes, and (b) the expected peer lifetime is 20 minutes.

every 0.1 minutes, such that the walker takes 100 walk steps between any two consecutive rounds of our algorithm. We simulate the system operating for a total of 500 minutes, that is, in the duration of 500 minutes each peer performs 50 rounds of our proposal. We measure the load imbalance factor of each alive peer after each algorithmic round. Notably, the ideal systemwide load imbalance factor is around 9.87 in expectation as 1) each newly coming peer joins the system with the capacity following the default Pareto distribution defined in Section 4.1, and 2) the set of virtual servers does not vary over time. We simulate 10,000 alive peers in the system.

As we can see from Fig. 12a, when the peer has a relatively larger lifetime in expectation (the expected lifetime of 100 minutes represents that 1,000 peers join and 1,000 peers depart in expectation between two consecutive algorithmic rounds), our proposal performs very well. If the arrival and departure rates of peers increase up to five times (i.e., each alive peer has the expected lifetime of 20 minutes, which means that between two consecutive algorithmic rounds 5,000 peers join and 5,000 peers depart in expectation), each alive peer remains to balance its load reasonably well (Fig. 12b).

ACKNOWLEDGMENTS

The authors are grateful to the anonymous reviewers who have provided them with valuable comments to improve their study. This work was partially supported by Taiwan National Science Council under Grant 97-2221-E-006-132 and Grant 98-2221-E-006-096, and by the Ministry of Education, Taiwan, under the NCKU Project of Promoting Academic Excellence & Developing World Class Research Centers.

REFERENCES

[1] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. Networking,* vol. 11, no. 1, pp. 17-21, Feb. 2003.

[2] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science,* pp. 161-172, Springer, Nov. 2001.

[3] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," *Performance Evaluation,* vol. 63, no. 6, pp. 217-240, Mar. 2006.

[4] Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 4, pp. 349-361, Apr. 2005.

[5] H. Shen and C.-Z. Xu, "Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured P2P Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 6, pp. 849-862, June 2007.

[6] L.M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. Software Eng.,* vol. 11, no. 5, pp. 491-496, May 1985.

[7] L.M. Ni, C.-W. Xu, and T.B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," *IEEE Trans. Software Eng.,* vol. 11, no. 10, pp. 1153-1161, Oct. 1985.

[8] C. Chen and K.-C. Tsai, "The Server Reassignment Problem for Load Balancing in Structured P2P Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 2, pp. 234-246, Feb. 2008.

[9] T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Trans. Software Eng.,* vol. 14, no. 2, pp. 141-154, Feb. 1988.

[10] Y. Zhu, "Load Balancing in Structured P2P Networks," *Handbook of Peer-to-Peer Networking,* Springer, July 2009.

[11] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," *Proc. 16th ACM Symp. Parallel Algorithms and Architectures (SPAA '04),* pp. 36-43, June 2004.

[12] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," *Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS '02),* pp. 68-79, Feb. 2003.

[13] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[14] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," *Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01),* pp. 202-215, Oct. 2001.

[15] G.H. Gonnet, "Expected Length of the Longest Probe Sequence in Hash Code Searching," *J. ACM,* vol. 28, no. 2, pp. 289-304, Apr. 1981.

[16] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, Sept. 2001.

[17] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *J. Chemical Physics,* vol. 21, no. 6, pp. 1087-1092, June 1953.

[18] W.K. Hastings, "Monte Carlo Sampling Methods Using Markov Chains and Their Applications," *Biometrika,* vol. 57, no. 1, pp. 97-109, Apr. 1970.

[19] S.M. Ross, "Markov Chains," *Introduction to Probability Models,* ninth ed., pp. 185-280, Academic Press, 2007.

[20] M. Mitzenmacher and E. Upfal, "Coupling of Markov Chains," *Probability and Computing: Randomized Algorithms and Probabilistic Analysis,* pp. 271-294, Cambridge Univ. Press, 2005.

[21] P. Diaconis and D. Stroock, "Geometric Bounds for Eigenvalues of Markov Chains," *The Annals of Applied Probability,* vol. 1, no. 1, pp. 36-61, Feb. 1991.

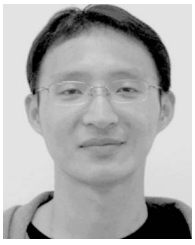
[22] A. Sinclair, "Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow," *Combinatorics, Probability and Computing,* vol. 1, no. 4, pp. 351-370, Dec. 1992.

[23] H. Chernoff, "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations," *Annals of Math. Statistics,* vol. 23, no. 4, pp. 493-507, 1952.

[24] S. Saroiu, P.K. Gummadi, and S.D. Gribble, "Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. Multimedia Computing and Networking (MMCN '02),* Jan. 2002.

Hung-Chang Hsiao received the PhD degree in computer science from the National Tsing-Hua University, Taiwan, in 2000. He is currently an associate professor in computer science and information engineering, National Cheng-Kung University, Taiwan, since August 2009. He was also a postdoc researcher in computer science at the National Tsing-Hua University, from October 2000 to July 2005. His research interests include peer-to-peer computing and overlay networking. He is a member of the IEEE Computer Society.

Hao Liao received the BS degree in electrical engineering from the National Cheng-Kung University, Taiwan, in 2004. He is currently working toward the PhD degree in computer science and information engineering at the National Cheng-Kung University. His research interests include peer-to-peer computing, algorithm design, and performance analysis.



Ssu-Ta Chen received the BS degree in computer science and information engineering from Tamkang University, Taiwan, in 2005, and the MS degree in computer science and information engineering from the National Cheng-Kung University, Taiwan, in 2010. His research interests include algorithm design and analysis for peer-to-peer networks.

Kuo-Chan Huang received the BS degree in computer science and engineering from the National Taiwan Ocean University in 2008 and the MS degree in computer science and information engineering from the National Cheng-Kung University, Taiwan, in 2010. His research interests include peer-to-peer computing, and social networking.