IMPROVED LOAD BALANCING ALGORITHMS IN DHT-BASED DYNAMIC P2P SYSTEMS

Jin-Woo Song  
Department of Computer Science  
Yonsei University  
Seoul, Korea  
fantaros@cs.yonsei.ac.kr

Sung-Bong Yang  
Department of Computer Science  
Yonsei University  
Seoul, Korea  
yang@cs.yonsei.ac.kr

ABSTRACT
Load balancing for DHT (Distributed Hash Table)-based peer-to-peer (P2P) systems has been studied extensively. The DHT-scheme simply assigns each object to a specific peer with a hash function. This could result in \(O(\log N)\) load imbalance, where \(N\) is the number of peers in a system. It is natural that a P2P system experiences frequent join and leave of both objects and peers. This dynamic characteristic may intensify load imbalance and, therefore, could increase the overhead for load balancing. In this paper, we propose two load balancing schemes, the object lifetime-aware load balancing (OLLB) and the peer activity-aware load balancing (PALB) schemes. The OLLB scheme uses the predicted lifetimes of objects to deal with the trade-off between the load balancing and the load movement. And the PALB scheme manages intelligently the virtual server operations to reduce the load unbalance occurred by dynamic peer activities. The simulation results show that the proposed schemes improve the load balancing performance with less overhead in the dynamic environment.

KEY WORDS
Peer to Peer Network Technologies for Management, DHT, Dynamic Environment, Load Balancing, Lifetime.

1 Introduction
In recent years, peer-to-peer (P2P) systems have been studied extensively by many researchers in Computer Science, especially in networking. One of the main advantages of a P2P system is the scalable management of a large amount of objects distributed across many peers in a network. Such a system provides a fertile soil for file sharing applications such as Napster, Gnutella, and Kazaa because of its interesting potentials like self-organization, decentralization, and scalability.

Structured P2P systems such as Chord [1], Pastry [2], Tapestry [3], and CAN [4] provide a DHT (Distributed Hash Table) abstraction. These structured systems assign a unique identifier to each object and to each peer with a hash function. The identifier space for peers is divided among the peers in a system, and each peer stores the lookup entries for objects (called object keys) each of which is mapped to an identifier in its portion of the identifier space [1]. Note that an object key is the address of an object stored at a different peer and a unique identifier is assigned to the address with a hash function. So, the object key is stored in a peer who is responsible for that address according to the DHT abstraction.

For example, Chord uses Consistent Hashing [5] for naming the identifiers of objects and each object key is assigned to a “deterministic” peer whose identifier’s value is close to the value \(x\) of the object key’s identifier and not smaller than \(x\) in the address space.

It is natural that a great number of objects are created and deleted quite often in a P2P system due to the fact that peers may join and leave freely. So we should concern about balancing the loads among peers adequately to their own capacities, called load balancing. We assume that there is one bottleneck resource but the types of loads may be diverse, for example, the number of bits to store an object key, the access frequency of an object, or the CPU processing time to serve an object. Note that the capacities of peers are heterogeneous since each peer in a P2P system may have different attributes such as the CPU speed, available disk space, or network bandwidth [6][7]. The volumes of peers are also heterogeneous, which we defined as the allocated storage capacity for sharing objects. These heterogeneities eventually result in load imbalance. In a DHT-based P2P system, the random choices on the object IDs in resorting to the uniformity of a hash function do not guarantee a satisfactory load balance. It could result in \(O(\log N)\) load imbalance factor, where \(N\) is the number of peers in the system [8]. In order to resolve load imbalance, we distribute the load into the peers who have room for additional loads in such a way that a peer receives a load proportional to its capacity. A well-balanced P2P system is able to provide faster lookup services without network bottlenecks and could guarantee the scalability as well.

Several load balancing algorithms have been proposed, but they only work for a static environment in which peers are not supposed to join and leave a P2P system [1][8][9]. Recently a load balancing algorithm for a dynamic environment was proposed, but it did not consider the object lifetime for load balancing [10]. By the object lifetime we mean the difference in time between when an object appears and when it disappears in the P2P system.
Predicting the lifetimes of objects allows a P2P system to maintain a satisfactory load balancing among peers efficiently. A load balancing scheme should be designed to minimize the load imbalance as well as the cost of the load movement among the peers in the system as in [10]:

- Minimize the load imbalance: The load $i$ of a peer $i$ is the sum of the loads of the object keys stored on the peer and the capacity $i$ of a peer $i$ is fixed. A utilization of a peer $i$ is the fraction of its used capacity. ($\text{utilization}_i = \frac{\text{load}_i}{\text{capacity}_i}$)
- Minimize the cost of the load movement: Load movement consumes the network bandwidth and hence a large amount of load movements degrade the quality of services.

In this paper, we propose a load balancing algorithm for a DHT-based dynamic system. For designing our algorithm, we combine two new schemes along with basic schemes presented in [8]. These new schemes are the object lifetime-aware load balancing (OLLB) scheme which avoids unnecessary load movement using object lifetimes, and the peer activity-aware load balancing (PALB) scheme which efficiently handles the load movement occurred in peers’ join and leave. The experimental results show that the each proposed scheme effectively operates to achieve the load balancing and reduce the load movement.

The rest of this paper is organized as follows. Section 2 presents the related work. In Section 3, the proposed load balancing schemes are described in detail. Section 4 provides the experimental results of proposed load balancing schemes. Finally, in Section 5, the conclusion and future work are given.

2 Related Work

We first introduce the notion of the virtual servers presented in [11], since our proposed schemes work under the concept of virtual servers for load balancing. A virtual server in a P2P system behaves like a ‘real’ peer in the underlying DHT overlay. A peer owns one or more virtual servers and it covers discrete portions of the identifier space, where each portion is covered a virtual server it owns. The main advantage of using virtual servers for load balancing is that the load movement can be realized by only changing the ownership of virtual servers rather than changing the underlying DHT.

Chord [1] suggests that the virtual server concept is suitable for achieving the load balance. It allocates $\log N$ virtual servers per peer and ensures that the number of object keys per peer is within a constant factor of the optimal number, $O(\log N)$, with a high probability. However, Chord assumes that peers are homogeneous, so we need more sophisticated methods for load balancing in much complicated and realistic peer conditions.

Rao et al. [8] proposed three load balancing algorithms using virtual servers; one-to-one, one-to-many, and many-to-many schemes. The one-to-one scheme connects two randomly selected peers. If one peer is overloaded and the other is not, some of the virtual servers of the overloaded peer are transferred into the under-loaded one. The one-to-many scheme associates one overloaded peer with several peers through a directory, where a directory has the information on under-loaded peers. The many-to-many scheme associates multiple overloaded peers with multi- ple under-loaded peers. However, they were simulated only under a static environment.

Godfrey et al. [10] uses emergency load balancing to deal with dynamic peer participation between periodic load balancing. The emergency load balancing is performed immediately when the ratio of the peer’s load to its capacity gets larger than a certain threshold. However, the frequent participation of objects may occur a lot of emergency load balancing which weight the overhead to P2P system. Furthermore, they did not consider the volume of each peer.

Byers et al. [9] applied the “power of two choices” paradigm to load balancing. A fixed number $d$ of hash functions are used for each object to get $d$ different identifiers. Therefore, it obtains $d$ corresponding peers with the least amount of loads. It achieves load balance within $\log \log N$ factor of the optimal when $d = 2$. However, their algorithm was simulated only for homogeneous peer capacities in a static environment.

Karger and Ruhl [12] proposed two protocols, address space balancing and item balancing to balance the loads dynamically. They reassign lightly loaded peers to the neighbors of heavy loaded peers to achieve load balance dynamically. However, they did not consider the heterogeneities of both peers’ volumes and capacities.

Rieche and Petrak [13] introduced a load balancing scheme similar to heat dispersion. Objects are moved among peers similar to the process of heat expansion [14]. A local load balancing is performed continuously throughout the entire system and thus the whole system is balanced in a decentralized manner. Each peer has to interact only with some of its immediate neighbors. However, this approach also did not consider the heterogeneities of both peers’ volumes and capacities.

3 The Proposed Load Balancing Algorithm

3.1 Load Prediction Mechanism

In this section, we present how to predict the load variation of a peer. A peer’s load, the number of object keys, changes in proportion to the number of object identifiers which are mapped to the peer in a DHT-based P2P system. So, we turn our attention to the lifetime of an object to predict the load of each peer. The lifetime of an object is dependent on the volume status of the peer who actually holds the object. In a realistic P2P system, numerous objects join frequently. So a peer should replace existing objects with new ones when its volume is full. (We assume that the size of the volume of each peer is fixed and the FIFO replacement
strategy is used.) Consequently, the variation of the peer’s load is deeply related to its replacement history, which provides a good clue to the prediction in the lifetime of an object.

Let $\text{Life}_{T_p}$ be the average lifetime of all the objects stored at peer $p$’s volume. Then $\text{Life}_{T_p}$ can be estimated with the average object replacement time as in Equation (1), where $\text{Store}_{T_p^i}$ is the time when object $i$ was stored at peer $p$, $\text{Del}_{T_p^i}$ is the time when object $i$ is deleted, and $\text{NumDel}_p$ is the number of deleted objects over a certain period of time (we call this period $\text{History}_{T}$). To obtain $\text{Life}_{T_p}$, each peer $p$ should record the time stamps (the history) for each object both when it is stored and deleted. However, each peer uses only a limited space for storing the history.

$$\text{Life}_{T_p} = \frac{\sum_{i=1}^{\text{NumDel}_p}(\text{Del}_{T_p^i} - \text{Store}_{T_p^i})}{\text{NumDel}_p}$$ (1)

Now we can predict the disappearance time $\text{Disapp}_{T_p^i}$ of object $i$ at peer $p$ as follows.

$$\text{Disapp}_{T_p^i} = \text{Store}_{T_p^i} + \text{Life}_{T_p}$$ (2)

Let $\text{Load}_p$ and $\text{Capacity}_p$ be the load and the capacity of peer $p$, respectively. We call peer $p$ a heavy peer if $\text{Load}_p > \text{Capacity}_p$, and a light peer, otherwise. We can now estimate the remaining lifetime $\text{Rem}_{T_p}^i$ of object $i$ at peer $p$ with $\text{Disapp}_{T_p^i}$ as in Equation (3), where $\text{Current}_{T}$ denotes the current time.

$$\text{Rem}_{T_p}^i = \text{Disapp}_{T_p^i} - \text{Current}_{T}$$ (3)

We use a threshold $\delta_{\text{Leave}_{T}}$ for the expected time for an object to leave the P2P system such that if $\text{Rem}_{T_p}^i < \delta_{\text{Leave}_{T}}$, we regard object $i$ at peer $p$ as an object that will leave the system soon. $\delta_{\text{Leave}_{T}}$ adjusts the trade-off between the quantity of the load balance and the cost of the load movement. Longer $\delta_{\text{Leave}_{T}}$ reduces the load movement, yet it may make load imbalance worse. Hence, $\delta_{\text{Leave}_{T}}$ should be chosen properly to balance the trade-off. Let $\text{NumDisappKeys}_p$ be the number of the object keys that are expected to disappear in time $\delta_{\text{Leave}_{T}}$ at peer $p$. We call peer $p$ a "lightable" peer, if $\text{Load}_p - \text{NumDisappKeys}_p \leq \text{Capacity}_p$. That is, the size of the load at peer $p$ is expected to get smaller than its capacity in time $\delta_{\text{Leave}_{T}}$ and $p$ is a potential light peer. Note that a lightable peer may not always become a light peer.

### 3.2 The Object Lifetime-aware Load Balancing Scheme

In this section, we present the object lifetime-aware load balancing scheme (OLLB) that utilizes the predicted lifetime of the load in each peer. OLLB selects a proper virtual server $v$ in a heavy peer $p_h$ and transfers $v$ to a light peer $p_l$ with the following rules:

1. A virtual server $v$ must be selected in such way that transferring $v$ from $p_h$ to $p_l$ should not make $p_l$ heavy.
2. Among the virtual servers satisfying rule 1, a virtual server should be the lightest virtual server that makes $p_h$ a lightable peer.
3. If there is no virtual server whose transfer can convert $p_h$ into a lightable peer, then transfer the virtual server who has the longest average object lifetime.

The above rules have been made by modifying the rules in [8] in order to utilize object lifetimes in load balancing. Rule 1 prevents $p_l$ from becoming a heavy peer due to virtual server transferring. Rule 2 selects the lightest $v$ that may change $p_h$ to a lightable peer, not to a light peer. As described in Section 3.1, a lightable peer means that the load is expected to be light in time $\delta_{\text{Leave}_{T}}$. So, $p_h$ only transfers an excessive amount of $p_h$’s load to $p_l$ excluding the number of keys in $p_h$ that will disappear soon. Therefore, OLLB increases the chances to reduce the load movement significantly and to achieve load balance in time $\delta_{\text{Leave}_{T}}$. If there is no virtual server whose transfer can make $p_h$ lightable, we have to choose a virtual server in $p_h$ without violating rule 1. Rule 3 selects a virtual server which holds the object keys that will remain for longer period of time in the system. The reason why we select such a virtual server is that the more objects with short life remain at $p_h$, we get the higher possibility that $p_h$’s load decreases soon.

OLLB may be based on either the one-to-many or the many-to-many scheme. In this paper, we utilize both schemes that operate periodically for load balancing.

The OLLB scheme with the one-to-many scheme, called 1-M OLLB, distributes the overload of a heavy peer $p_h$ as in Figure 1. In the 1-M OLLB scheme, $p_h$ distributes its overload into light peers registered in $\text{Directory}_{p_h}$ sequentially, where $\text{Directory}_{p_h}$ denotes the directory with which $p_h$ is associated by the hash function. It terminates as soon as $p_h$ becomes lightable (rule 2) or $p_h$ contacts all $p_l$s in $\text{Directory}_{p_h}$. Lines 7–10 select a virtual server $v$ that has the minimum load satisfying rule 1

![Figure 1. The 1-M OLLB scheme](image-url)
The OLLB scheme with the many-to-many scheme, called M-M OLLB, distributes the overload of heavy peers in Directory$_i$, where Directory$_i$ denotes one of $d$ directories as in Figure 2. The many-to-many scheme uses two data structures, $d$ directories and one global pool. Unlike the directories in the one-to-many scheme, directories in the many-to-many scheme keep the load states for both heavy and light peers. Each peer can be associated to one of $d$ directories by the hash function and peers periodically report their load states to their corresponding directory. The global pool is a temporary space that keeps the information to transfer virtual servers while periodic load balancing is executed.

The M-M OLLB scheme initializes the global pool to NULL. And then each peer $p$ in Directory$_i$ predicts the life and disappearance time of each object key. If $p$ should lighten its own load, which means $\text{Load}_p - \text{NumDisappKeys}_p > \text{Capacity}_p$, the M-M OLLB scheme moves the lightest $v$ to the global pool until $p$ becomes a light peer as in Lines 4–5 (rule 2). Then, the overloads in the global pool are sequentially distributed in the order of their sizes. The lightest virtual server $v$ is transferred to the peer $p$ that makes $\text{PredictLoadRatio}_p$ minimum. So, it fills the largest free space with the smallest load to minimize the peers’ utilization. By considering $p$’s disappearing keys in the $\text{PredictLoadRatio}_p$, the M-M OLLB scheme may utilize more free spaces.

### 3.3 The Peer Activity-aware Load Balancing Scheme

The peer activity-aware load balancing scheme (PALB) scheme intelligently controls the number and the movement of virtual servers occurred by dynamic peer activities such as peers’ join and leave. When a peer joins or leaves a DHT-based P2P system, virtual servers are created or discarded inevitably. Hence, corresponding load movement should occur. Such load movement may intensify the load imbalance of the system. We made an experiment to confirm the effects with dynamic peer activities and the results are given in Section 4.3.1) In Chord [1], a leaving peer $p$ moves all loads to other peers and discards virtual servers from the system. If $p$ transfers the loads of a virtual server $v$ to peer $p'$ who is responsible for $v'$ that immediately succeeds $v$ in the identifier space, then $p'$ undertakes the load of $v$ and may become a heavy peer. Also, a joining peer creates $\log N$ virtual servers whose identifiers are randomly produced, so the $\log N$ virtual servers are distributed the load of succeeding virtual servers. However, theses simple virtual server movement may cause a heavy peer to become heavier and a light peer to be lighter, that is, it deteriorates the load balance further.

The PALB scheme tries to manage the virtual server operations such as create, discard, and transfer with the one-to-one scheme. We may prevent the occurrence of heavy peers due to a peer’s leave through the one-to-one load balancing. For example, a leaving peer does not discard its virtual servers, but distributes them into other light peers. We use the one-to-one scheme to minimize the overhead. However, the number of virtual servers increases if many peers join and leave continuously. A lot of virtual servers cause the lookup delay problem [1]. So, the PALB scheme tries to control the number of virtual servers in a P2P system while the operations are executed.

The PALB scheme with the one-to-one scheme, called 1-1PALB, is described in Figure 3. When a peer leaves a P2P system, the 1-1PALB scheme discards only the virtual servers that do not make other peers overloaded (heavy). Then a leaving peer selects a directory randomly and distributes its virtual servers to a light peer in the directory with the one-to-one scheme. We repeat the one-to-one
load balancing as many as the number of virtual servers, $log N$, because applying the one-to-one scheme just once is not enough for load balancing. In the pseudo code, \texttt{HeavyToLight}(p_h, p_l) transfers virtual servers from $p_h$ to $p_l$ according to the rules in Section 3.2. If there are remaining virtual servers in a leaving peer, it discards those virtual servers. Finally, the leaving peer records the number of virtual servers that are not discarded but transferred onto $NumExcessiveVS$, the number of excessive virtual servers, in the directory. We maintain a proper number of virtual servers in proportion to the number of peers in the system, when a joining peer creates its virtual servers.

When a peer joins, the 1-1PALB scheme obtains $NumExcessiveVS$ from a randomly selected directory. A joining peer creates virtual servers as many as $log N$ minus $NumExcessiveVS$ and enters those virtual servers to the system. Then a joining peer repeatedly takes load balancing with the one-to-one scheme to lighten the load of a heavy peer in the directory.

4 Performance Evaluation

4.1 The Simulation Environment

We simulated our proposed load balancing algorithm in a dynamic environment where objects and peers continuously join and leave in a DHT-based P2P system. Note that we used the DHT scheme in Chord. In the simulation environment, objects join the system by the Poisson distribution. When a peer joins, the 1-1PALB scheme obtains $NumExcessiveVS$ from a randomly selected directory. A joining peer creates virtual servers as many as $log N$ minus $NumExcessiveVS$ and enters those virtual servers to the system. Then a joining peer repeatedly takes load balancing with the one-to-one scheme to lighten the load of a heavy peer in the directory.

4.2 The Performance Metrics

To evaluate the load balancing performance, we use two evaluation metrics: the 99.9th percentile peer utilization and the load movement cost. The 99.9th percentile peer utilization is the 99.9th percentile of the utilization of the peers at time $t$. The cumulative 99.9th percentile peer utilization is the accumulation of the 99.9th percentile peer utilization during the entire simulation period. In the simulation, we set the target utilization of each peer to 1.0. The load movement is measured as the movement cost incurred due to load balancing at time $t$. The cumulative load movement cost is the accumulation of the load movement cost during the entire simulation period. We measure the results for 1,000 seconds after a stabilization time. All the experiment results are the average over ten independent trials.

4.3 The Simulation Results

4.3.1 Dynamic Peer Activity Problems

For a dynamic peer join/leave environment, we use query scenario data. The query scenario data consist of the sequence of time stamp, a peer ID and an object ID. We assume that a peer leaves the P2P system if the peer did not take any action such as querying an object for the peer activity threshold. We further assume that a peer who left the P2P system joins again the system when it takes any action. Figure 4 shows the cumulative peer activity for a scenario data excluding the initial join of peers in the P2P system. The smaller the peer activity threshold is, the more frequent peer join and leave occur. When the peer activity threshold is 45 seconds, almost no peer leaves or joins the system. Figure 5 compares the 99.9th percentile peer utilizations of

<table>
<thead>
<tr>
<th>Table 1. Simulation parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
</tr>
<tr>
<td>System Utilization</td>
</tr>
<tr>
<td>Number of peers</td>
</tr>
<tr>
<td>Peer capacity / volume</td>
</tr>
<tr>
<td>Object arrival rate</td>
</tr>
<tr>
<td>Average number of objects</td>
</tr>
<tr>
<td>Object movement cost</td>
</tr>
<tr>
<td>Object load / size</td>
</tr>
<tr>
<td>Periodic load balance interval $T$</td>
</tr>
<tr>
<td>Number of virtual servers per peer</td>
</tr>
<tr>
<td>Number of directories</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2. The list of load balancing schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Schemes</td>
</tr>
<tr>
<td>Proposed Schemes</td>
</tr>
<tr>
<td>Load Balancing Schemes</td>
</tr>
<tr>
<td>one-to-many</td>
</tr>
<tr>
<td>1-M OLLB</td>
</tr>
<tr>
<td>1-M + OLLB</td>
</tr>
<tr>
<td>1-1 PALB</td>
</tr>
<tr>
<td>many-to-many</td>
</tr>
<tr>
<td>M-M OLLB</td>
</tr>
<tr>
<td>M-M + OLLB</td>
</tr>
<tr>
<td>1-1 PALB</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>


Pure M-M between when the thresholds are 25 seconds and 45 seconds. The 99.9\textsuperscript{th} percentile peer utilization stays at 1.0 on each occasion of load balancing for both cases but it increases drastically until the next load balancing when the threshold is 45 seconds. This result implies that frequent join and leave of peers intensify the load unbalance of the P2P system.

Figure 6 shows the maximum 99.9\textsuperscript{th} percentile peer utilization and the load movements of Pure M-M for various peer activity thresholds.

4.3.2 Performance Results for the Hybrid Algorithms

In this section, we simulate the hybrid algorithms that combine the OLLB and PALB schemes; they are 1-M+OLLB+PALB and M-M+OLLB+PALB. They are tested under various environments with several scheme parameters. All the experiments are simulated in dynamic object and peer environments.

Figure 7 shows the cumulative 99.9\textsuperscript{th} percentile peer utilizations and the average load movements with various load balancing intervals for the basic and the hybrid algorithms.

Figures 8 shows the load balance performances with various object leave thresholds. As the object leave threshold increases, the cumulative 99.9\textsuperscript{th} percentile peer utilizations of hybrid algorithms increase while the cumulative load movements of hybrid algorithms decrease. If the object leave threshold is too large, an OLLB scheme assumes the object disappearance time to be long. Then the load movement during load balancing is quite small but the 99.9\textsuperscript{th} percentile peer utilization gets worse. In Figure 8 we could confirm the trade-off between the 99.9\textsuperscript{th} percentile peer utilization and the load movement.

Figure 9 shows the load balance performances with various peer activity thresholds. Even if peers frequently join and leave the P2P system, hybrid algorithms perform improved load balancing with greatly reduced load movements compared with Pure 1-M and Pure M-M.

To find out the effect of the distributions of capacities and volumes of peers in the P2P system, we simulate four different environments, G/G, G/P, P/G, and P/P, where X/Y denotes that the volumes and capacities have X and Y distributions, respectively, and G is the Gaussian distribution.
Figure 9. The cumulative 99.9th percentile peer utilizations and the cumulative load movements with various peer activity thresholds for the basic and the hybrid algorithms

and P is the Pareto distribution in Figure 10. The cumulative 99.9th percentile peer utilizations and the cumulative load movements in the P/P environment are higher than those in the G/G environments. It means that the heterogeneity of peers makes hard the load balancing.

In the G/G environment, M-M+OLLB+PALB shows better results than Pure M-M by 2.9% and 53.3% for the cumulative 99.9th percentile peer utilizations and the cumulative load movements, respectively. In the P/P environment, M-M+OLLB+PALB improves the performance by 8.8% and 41% compared with Pure M-M, respectively. These results show that M-M+OLLB+PALB outperforms load balancing with smaller load movements in any environments.

5 Conclusion and Future Work

In this paper, we introduced the object lifetimes of peers to a P2P system in order to improve load balancing and managed the virtual servers of active peers in a dynamic environment where objects and peers continuously join and leave the system. The lifetimes of objects in the system are utilized for predicting the changes in loads at peers. The proposed OLLB scheme uses the predicted lifetimes to deal with the trade-off between load balancing and the load movement. And the PALB scheme intelligently manages the virtual server operations to reduce the load unbalance occurred by dynamic peer activities. The experimental results show that the hybrid load balancing algorithms are quite effective under various conditions and environments and especially, M-M+OLLB+PALB outperforms other load balancing schemes with lower cumulative 99.9th percentile peer utilizations and smaller cumulative load movements. We plan to expand the peers’ conditions such as varying the sizes of loads of peers and the sizes of objects for more realistic P2P environments.

References