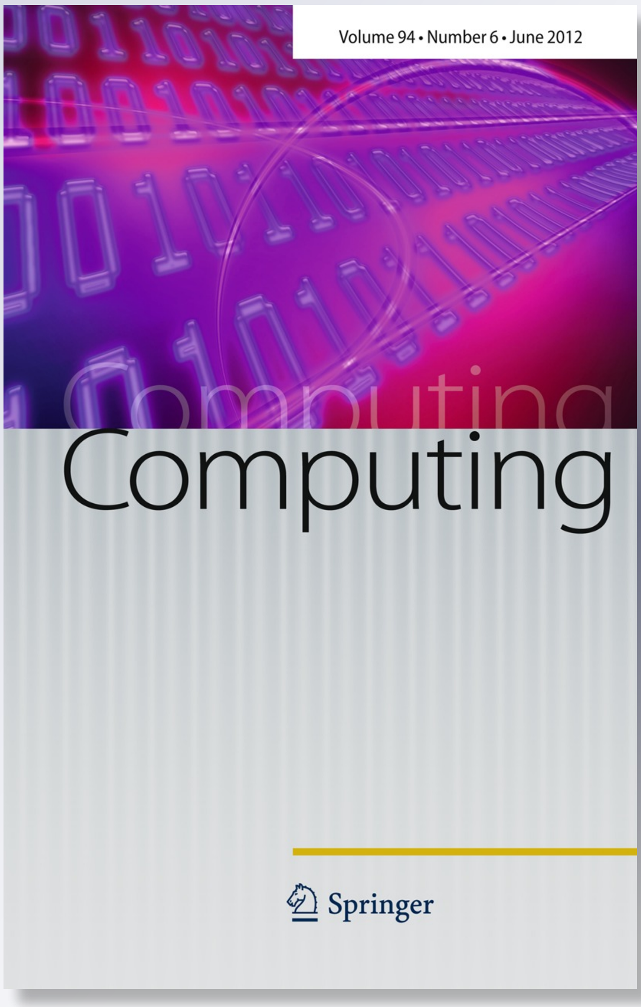
Load balancing in peer-to-peer systems using a diffusive approach

Ying Qiao & Gregor v. Bochmann

Computing
Archives for Scientific Computing

ISSN 0010-485X
Volume 94
Combined 8-10

Computing (2012) 94:649-678
DOI 10.1007/s00607-012-0196-x



Computing (2012) 94:649–678
DOI 10.1007/s00607-012-0196-x

Load balancing in peer-to-peer systems using a diffusive approach

Ying Qiao · Gregor v. Bochmann

Received: 15 November 2011 / Accepted: 5 June 2012 / Published online: 22 June 2012

Abstract We developed a diffusive load balancing technique for P2P systems. This technique uses the overlay network of a P2P system and results in the nodes of the network having similar available capacities; therefore the services hosted on these nodes are expected to have similar mean response times. In this paper, the technique is presented, including the policies, stages of operation, and decision algorithms. The convergence of the available capacities to the global average is demonstrated. The convergence speed depends on the decision algorithm, the neighborhood structure of the underlying overlay network, and the workload distribution. When used in a system with churn, the technique keeps the standard deviation of available capacities in the system within a bound. This bound depends on the amount of churn and the frequency of the load balancing operations, as well as on the distribution of node capacities. However, the sizes of services have little impact on this bound. The paper presents the results of analytical analysis and simulation studies.

Keywords Load balancing · Diffusive load balancing · Peer-to-peer systems · Distributed algorithms · Performance management component

Mathematics Subject Classification 68W15, Distributed algorithms · 68M20, Performance evaluation, queueing, scheduling · 68M14, Distributed systems

Y. Qiao · G. v. Bochmann (✉)
School of Electrical Engineering and Computer Science,
University of Ottawa, Ottawa, ON K1N 6N5, Canada
e-mail: bochmann@eecs.uottawa.ca
URL: http://www.eecs.uottawa.ca/~bochmann/

Y. Qiao
e-mail: yqiao074@site.uottawa.ca

1 Introduction

The performance issues of peer-to-peer (P2P) systems come from the characteristics of these systems. The nodes in these systems may be heterogeneous in terms of their resource capacities, geographic locations, or on-line periods (i.e. time of participation in the system). The services for accessing the shared objects on the P2P nodes also have different resource requirements and the number of user requests, and the locations of users may change. In these systems, some user requests may be delayed or even lost by some nodes, while other nodes are idle. Also, the mean response times of the requests that access the same object on the same node could vary over time because of changing request rates.

Load balancing techniques are proposed to solve these issues. These techniques dynamically reallocate nodes or shared objects in the P2P system so that they could provide a more uniform quality of service for their services. Using distributed approaches, these techniques are scalable to large sizes of P2P systems. However, these techniques either rely on some specific structures of the overlay networks or construct their own structures therefore inducing extra messages. For example, some techniques, such as the one in the BATON system [26] or in the DPTree system [16], use the tree structure of an overlay network for their operations. These techniques can not be deployed in an overlay network using another kind of structure. Some techniques require the system to construct a structure based on their P2P overlay networks (e.g. [10,11]) only for load balancing. Or, some techniques use random walks (e.g. [7,14]) in their load balancing operations. These random walks add extra messages to the P2P system.

We developed a diffusive load balancing technique for P2P systems. It uses a diffusive load balancing scheme, originally proposed for parallel computing systems that have a massive number of processors. In order to deal with the characteristics of P2P systems, this diffusive scheme specifies that the P2P nodes run the load balancing operations asynchronously. During each operation, a node collects the load statuses (i.e. the available capacities) of its neighbors in the overlay network, and decides load transfers (i.e. object movements) between these neighbors so that the neighbors could have similar mean response times for their services. A global balanced state, where all nodes have the same mean response time, can be achieved by this kind of local load balancing. Compared with other techniques proposed for P2P systems, the diffusive technique neither sets up extra connections between nodes nor spends a large number of messages on random walkers in the system.

The following sections are organized for presenting different aspects of the proposed scheme. Section 2 reviews peer-to-peer load-balancing techniques and diffusive load-balancing schemes studied for parallel computing systems. Section 3 presents the design of the proposed diffusive scheme for systems with fine-grained services, including its policies, the stages of operation, and the decision algorithms. This section examines the convergence of the scheme and compares the effectiveness of the decision algorithms in systems with a skip-list overlay structure. It further discusses the impact of the characteristics of P2P systems on the effectiveness of load balancing, including the structure of the overlay network, workload distribution, churn (node joining or leaving), and nodes with heterogeneous capacities. Section 4 presents the decision algorithms for the diffusive scheme to deal with large-sized services. The section further studies the impact of the service size on the effectiveness of load balancing. Section 5 compares the diffusive scheme with other schemes for P2P systems proposed in the literature, discusses the practical realization of load transfers in a real P2P system, and concludes the paper.

2 Background

2.1 Load balancing techniques for P2P systems

The load balancing techniques proposed for P2P systems either allocate or relocate service objects onto nodes in these systems, or relocate nodes such that the users accessing these objects see a uniform response time. These techniques have policies to specify when and where to perform load balancing operations, and how to decide object locations/relocations during these operations. Since P2P systems have large sizes, these techniques also specify various structures and algorithms to detect the load status of the system and to decide load transfers between nodes.

These load balancing techniques differ in the structures by which they organize the nodes. This results in differences of the effectiveness of load balancing. A linked-list structure is the simplest structure used by techniques such as [3,11,12]. After a node conducted a load balancing operation (which moves objects between nodes), all the nodes in its neighborhood (e.g. including itself and its direct neighbors in the ring) have the same load. Vu et al. [11] shows that a simple load balancing scheme like this is not effective in balancing the load variations in systems with churn. Normally, another kind of scheme is used to further reduce the load variations.

Other structures, such as tree, distributed directory, or neighborhoods with randomly probed nodes, are also used for load balancing. For example, in the tree structure of BATON [26], a parent node works as a decision component (or directory) for balancing the loads for its sub-trees. The system is load balanced when the sub-trees of the root have equal loads. However, this load balancing can not be applied to systems with other kinds of structure. The distributed directories proposed in [8] is another structure used for load balancing. A node registers to a directory at random and stays there for some time, while the directory balances the loads of its registered nodes. However, the directory (central or distributed) scheme can not deal well with the dynamics of a P2P system and the running period of the directory must be engineered [7]. Some schemes, such as those in [5,7,13,14], construct neighborhoods by using random walks. A node probes some other random nodes for sharing their excess loads.

Some schemes use structures to aggregate the global load information for load balancing. These schemes are criticized because of the cost of collecting the global information and the limited freshness of this information. For example, the *k-ary* tree scheme [10] constructs a tree structure based on Chord for aggregating the global load information from the leaf-nodes to the root node, and for disseminating this information from the root to the leaves. The global information is used by the leaves to identify their state (either overloaded or under-loaded). Shen et al. [7] showed that the system using the scheme does not deal well with load variations due to churn (i.e. the loads of nodes have a large variance). Shen also revealed that, compared to other schemes, the *k-ary* tree scheme induces a larger number of messages. To avoid the cost of constructing a tree, the scheme for the DPTree system [16] generates a global map that is circulated among nodes. The map is updated during the circulation. However, the map itself reflects a tree structure; this prevents the scheme from being used in systems with another kind of structure. The *Histogram* scheme [11] constructs a structure for aggregating the global load information in a structured P2P system. Although the scheme keeps a smaller load imbalance for a P2P system than a scheme using a sender-initiated policy, the aggregation introduces message overhead. Random walks are sometimes used in estimating global load information for load balancing, such as in Mercury [12]. However, in order to precisely estimate the global load distribution, the scheme has to use $O(\log N)$ random walks for an operation to estimate the load status of a system that has N nodes. These random walks add extra overhead to the system.

2.2 Diffusive load balancing schemes

Diffusive load balancing schemes (also called diffusive schemes in this paper) are studied for balancing the computations of parallel computing programs over the nodes in parallel computer systems. These schemes are classified as synchronous or asynchronous schemes. A synchronous scheme specifies that all nodes, which are coordinated by a global clock, run load balancing operations at the same time. These operations conduct the activities such as reporting load statuses of nodes to neighbors, deciding load transfers, and transferring loads. Asynchronous diffusive schemes do not require this kind of synchrony while nodes periodically conduct individual load balancing operations. However, the delay for transmitting a message or transferring a load must be bounded. Therefore, these schemes are also called partially asynchronous schemes.

Diffusive schemes normally use a sender-initiated policy, where a sender node (that is, an overloaded node) decides and invokes the load transfers to receivers (that are under-loaded nodes). These schemes differ in their decision algorithms which calculate the amount of load to be transferred. Assuming that the objects (e.g. tasks, calculations or data items) on nodes are fine grained, Boillat [17] derived an algorithm for a synchronous scheme from the Poisson diffusion equation. Cybenko [18] derived an algorithm in a similar way. Bertsekas formalized a partial asynchronous scheme in [20] by specifying that an overloaded node should send its load to under-loaded nodes, especially the lightest loaded one, and, after a load transfer, the sender should still have more load than those having less loads before. The partially asynchronous schemes in [21,24] also specified the function for calculating the amount of workload to be transferred from load senders to receivers.

A diffusive load balancing scheme converges if the workload of all nodes (asymptotically) reaches the global average as time proceeds. It has been shown that the convergence speed of a synchronous diffusive scheme could be optimal by choosing the portion of workload for load exchanges between two neighbors according to the topology of a system [23]. If the load for an exchange is equal to a proportion of the difference between the loads of two neighbors [24], a scheme converges faster in a system with a hypercube or torus topology, where, compared with a ring or a linear network, the network has a symmetric graph and a smaller diameter.

Because of these properties, we propose an asynchronous diffusive scheme for load balancing in P2P systems.

3 Design of the diffusive load balancing scheme

The proposed diffusive scheme is different from the other load balancing schemes for P2P systems in terms of the load index (i.e. the measure used for indicating the load statuses of nodes), the policies for applying load balancing operations, and the stages of these operations. In this section, we describe the scheme in terms of these aspects and analyze its convergence speed. We also analyze the impact of the neighborhood structures, workload distributions, churn, and the heterogeneity of the node capacities on the effectiveness of load balancing.

3.1 Load index

P2P nodes are expected to provide services with a uniform mean response time, like the servers in a client/server system [9]. But, the load balancing techniques proposed for P2P systems in the literature can not achieve this purpose. For example, some load balancing schemes that equalize the amount of data or number of virtual servers on the nodes [2,3] do not deal with nodes with heterogeneous capacities. Some other schemes bring the utilizations of nodes to the average of the system [8,10]. It can be shown that, when two server nodes with different capacities have the same utilization, their mean response times might not be the same. That is, the requests arriving at the node with the higher capacity would experience a smaller mean response time. Therefore, these schemes do not equalize the response times of services on different nodes.

The proposed scheme considers the available capacities of the nodes as load index that must be equalized. The performance of a server in a client/server system is normally modeled as an M/M/1 queuing system. According to [19], we have

$$\mathrm{E}\left[r\right] = \frac{\frac{1}{u}}{1-\frac{\lambda}{\mu}} = \frac{1}{\mu - \lambda} \tag{3.1}$$

which indicates that the mean response time for requests $\mathrm{E}[r]$ is the inverse of the difference between the service rate μ and the arrival rate λ of requests on the server. We take the number of user request that can be processed per time unit as the measure of the (total) capacity of a node, normally written μ. Using the same units, the used capacity, normally written λ, is the number of user requests arriving per time unit. Then the available capacity is the difference between μ and λ. Equation 3.1 shows that, in the case that two server nodes have the same available capacity, the mean response times of their requests are the same. The equation also applies to the case where the two servers have different capacities. We conclude that, in the case that the nodes that could be modeled as M/M/1 queues in a P2P system have the same available capacity, the mean response times of the requests of the system are the same. Therefore, the purpose of the diffusive load balancing is to obtain similar available capacities for all nodes so that the services provided by the system could have a more uniformed mean response times or quality of service for their services.

In order to show that equalizing the available capacity leads to similar response times also in situations where the response time of each node is not accurately modeled by an M/M/1 queue, we compared in [27] two load balancing techniques for systems with nodes modeled as GI/G/1 queues: one equalizes the available capacities of nodes, the other their utilizations. This study showed that the first technique leads to more similar response times than the second. Using the first technique, the system has a smaller expected value and a smaller variance for the mean response times of nodes. Also, the variance of the mean response times is bounded by a fixed value. In a system using the second technique, this variance is bounded by $\frac{1}{1-\rho}$ where ρ is the equalized utilization.

The diffusive schemes for parallel computing systems have different load indexes for equalizing the amounts of computation on the nodes. Clearly, those load measures do not reflect the dynamic arrival and departure of requests on the nodes. Therefore, they are not appropriate for load balancing in P2P systems.

3.2 Load balancing operations and their decision algorithms

The proposed scheme specifies load balancing policies for its operations that periodically run on the nodes of a P2P system. The *information* policy specifies that a node periodically runs an operation to collect load information from its neighbors. The *transfer* and *location* policies specify the selection of the senders and receivers for load transfers within a neighborhood. A load balancing operation realizes these policies. We call the node that is executing a load balancing operation the operating node, and its neighborhood includes the operating node itself and its direct neighbors within the overlay network.

An operation goes through the following three stages:

- *Load determination*: In this stage, the operating node collects the available capacities of its neighbors by sending probing messages. A probed neighbor responds with its available capacity if it is not involved in another balancing operation. The operating node waits for these responses.
- *Decision:* First, the operating node calculates the average available capacity for the neighborhood. A node is identified as a candidate receiver (sender) of load or load-receiver (load-sender) if its available capacity is larger (smaller) than the average. Then, a decision procedure identifies one or several receiver-sender pairs and sends a load transfer request to the sender of each pair, including the ID of the selected receiver (which is the target of the load transfer) and the amount of load to be transferred (called required capacity). The detail of the decision procedure depends on the decision algorithm. We will discuss different decision algorithms in the following sections.
- *Load transfer*: During this stage, loads are transferred between the determined senders and receivers.

After having performed an operation, the operating node will go back to process the normal service requests until the time has come for another load balancing operation.

Operations on different nodes are not synchronized. These operations may run concurrently on different nodes, however, a node involved in one such operation will refuse the participation in another load balancing operation initiated by one of its neighbors. In this way, the load status information collected from a neighbor during an operation is always correct.

We consider the following algorithms that could be used in the decision stage of a load balancing operation: the *Proportional*, *Complete Balancing* (*CB*), *Directory-Initiated* (*DI*), *Sender-initiated* (*SI*) and *Receiver-initiated* (*RI*) algorithms. We assume that objects have sizes of fine granularity, which means that workloads of arbitrary sizes can be transferred; we also assume that they can be moved to any neighbor in the system. We will discuss the interaction of these load transfers with the search algorithm in the P2P system in the last section.

We introduce the following notations. The operating node of a load balancing operation is called node i. The neighborhood of the operating node is denoted as A_i, and the number of nodes in the neighborhood is $|A_i|$. A node in A_i is identified as a node j. A node x has a node capacity equal to C_x. If a node has services with a total resource requirements of l_x, its available capacity is $avc_x = C_x - l_x$. We write avc_x and avc'_x to represent the load status of node x at the beginning and at the end of an operation, respectively. For example, when services with resource requirements l have been transferred from node x to y at the end of an operation, we have $avc'_x = avc_x + l$ and $avc'_y = avc_y - l$.

3.2.1 Proportional algorithm

The *Proportional* algorithm (*Prop.*) has been discussed in [23], and we assume that the algorithm uses the available capacities of nodes as load index. Here, the decision algorithm determines the following load exchanges between node i and all other nodes j in its neighborhood: load equal to $k(avc_i - avc_j)$ will be transferred from node j to node i (if the value is negative, the exchange proceeds in the opposite direction), where k is a constant between zero and one. At the end of the operation, when all exchanges have been performed, the new available capacities are as follows: $avc'_i = (1-dk)avc_i + k\sum_j avc_j$ for i where $d = |A_i| - 1$, and $avc'_j = (1-k)avc_j + kavc_i$ for any neighbor j other than i.

3.2.2 Complete balancing algorithm

The *Complete Balancing* (*CB*) algorithm (also described in [23]) equalizes the available capacities of all nodes in the neighborhood of node i during an operation. The average available capacity of the nodes in the neighborhood of node i (including node i) is

$$avc_{A_i} = \frac{\sum_{j\in A_i} avc_j}{|A_i|} \tag{3.2}$$

The *CB* algorithm determines load exchanges such that at the end of the operation all nodes in the neighborhood have the same average available capacity.

```
Decision Procedure
1  Do forever
2      if SVect and RVect are not empty
3          $s = \min_{j \in SVect} \{avc_j\}$
4          $r = \max_{j \in SVect} \{avc_j\}$
5          $tr = \min\{avc_{A_i} - avc_s, avc_r - avc_{A_i}\}$
6          Send instruction to s and r with load equal to tr for the transfer;
7          remove s from SVect and r from RVect
8      else break;
9  End of Do
```

Fig. 1 The decision procedure of the *DI* algorithm

3.2.3 Directory-initiated algorithm

The *Directory-Initiated* (*DI*) algorithm is similar to the algorithm proposed in [4] for parallel computer programs. This algorithm also calculates the average available capacity of the neighborhood by using Eq. (3.2). Based on the value of the average, the algorithm identifies nodes as overloaded (if its available capacity is smaller than the average), under-loaded (if its available capacity is larger than the average), or equalized. The overloaded nodes are kept in a vector *SVect*, and the under-loaded nodes in a vector *RVect*. The algorithm uses the procedure shown in Fig. 1 to decide service migrations.

In the case that none of the vectors is empty, the procedure selects a pair of a sender s and a receiver r such that the two nodes have the largest difference between their loads among all the nodes remaining in the two vectors (line 3 and 4). Otherwise, the procedure stops (line 2). Line 5 decides the load that should be moved between s and r so that the sender s would not be under-loaded and the receiver r would not be over-loaded after the load transfer. The procedure continues after removing s from *SVect* and r from *RVect*.

3.2.4 Sender-initiated and receiver-initiated algorithms

Like the *DI* algorithm, the *Sender-Initiated* (*SI*) and *Receiver-Initiated* (*RI*) algorithms identify overloaded and under-loaded nodes according to the average. However, in the *SI* (*RI*) algorithm, node i is identified as a sender s (a receiver r) if its available capacity is smaller (larger) than the average; otherwise, no load transfer will take place. Similar algorithms have been proposed for parallel computing systems in [6].

The procedure for deciding the load transfer is shown in Fig. 2. The load to be transferred out from node s (called $avc_{required}$) is the difference between the average and the available capacity of s (line 1). The total providable available capacity (called $avc_{providable}$) is obtained from the available capacities of all under-loaded nodes. The load to be transferred into a receiver is proportional to its providable available capacity (line 6). The under-loaded nodes in *RVect* are considered one by one for deciding a load transfers.

```
    Decision Procedure
1   $avc_{required} = avc_{A_i} - avc_s$
2   $avc_{providable} = \sum_{r \in RVect} (avc_r - avc_{A_i})$
3   Do forever
4       if RVect is not empty
5           for a node r in RVect
6               $tr = avc_{required}(avc_r - avc_{A_i}) / avc_{providable}$
                send instruction to r with load equal to tr for the transfer
7               remove r from RVect
8       else break;
9   End of Do
```

Fig. 2 The decision procedure of the *SI* algorithm

The *RI* algorithm has a similar procedure where the receiver takes the role of the sender in Fig. 2, and its exceeding available capacity (i.e. providable available capacity) will be distributed to all the overloaded nodes.

3.3 Analysis of the scheme

In this section, we consider a P2P system with a static workload, where the workloads of the services performed on the nodes of the P2P system do not change over time. We will study here how the asynchronous, diffusive load balancing scheme leads the system to change from any initial state (e.g. where the loads of the nodes are uniformly distributed) to a globally balanced state (where all nodes have the same available capacity). First, we discuss the convergence of the diffusive scheme from an analytical viewpoint, and then we present some simulation studies which provide a more detailed comparison of the different decision algorithms.

3.3.1 Analytical considerations

The function that the *Proportional* algorithm uses to calculate the new available capacities of nodes is the function of an asynchronous diffusion scheme presented in [23] where workload is replaced by available capacity. The proof in [23] shows that, after an operation, the variance of the workload of all nodes in the system is decreased by a given factor a (smaller than 1). This means that the variance follows a geometric series of values which converges to zero. Hence, the *Proportional* algorithm converges when it uses the available capacity as load index. We can provide similar proofs of convergence for the other decision algorithms as follows.

We first discuss the *CB* algorithm in detail. We assume that there is a P2P system that consists of N nodes, and the global average of the available capacities of its nodes is $\overline{AVC} = \frac{\sum_j avc_j}{N}$. We write $\sigma^2(avc) = \frac{\sum_j (\overline{AVC} - avc_j)^2}{N}$ for the variance and $\sigma\ (avc)$ for the standard deviation of the available capacities in the system at the time before a node i starts its load balancing operation. Now we want to calculate the variance of the available capacities after this operation, written $\sigma^2(avc')$. We have Eq. (3.3) to calculate the variance of available capacities:

$$\begin{aligned} N\sigma^2\left(avc'\right) &= \sum_j \left(\overline{AVC} - avc'_j\right)^2 \\ &= \sum_{j\in A_i}\left(\overline{AVC} - avc'_j\right)^2 + \sum_{j\notin A_i}\left(\overline{AVC} - avc'_j\right)^2 \end{aligned} \tag{3.3}$$

At the right side of the equation, there are two terms. The first term is a sum over all the nodes j that are within the neighborhood A_i (including i), and it is equal to $|A_i|\left(\overline{AVC} - avc_{A_i}\right)^2$, where avc_{A_i} is the average of the available capacity of the nodes within i's neighborhood and $|A_i|$ is the number of nodes in this neighborhood. Then, we write

$$\mathrm{E}\left[N\sigma^2\left(avc'\right)\right] = \mathrm{E}\left[|A_i|\left(\overline{AVC} - avc_{A_i}\right)^2\right] + \mathrm{E}\left[\sum_{j\notin A_i}\left(\overline{AVC} - avc'_j\right)^2\right] \tag{3.4}$$

where the notation $\mathrm{E}[X]$ represents the mean of the random variable X. Since the local average avc_{A_i} is obtained over a set of $|A_i|$ nodes, the mean of avc_{A_i} is $\overline{AVC}$, and in the first term of Eq. (3.4), $\mathrm{E}\left[(\overline{AVC} - avc_{A_i})^2\right]$ is the variance of avc_{A_i} which is equal to $\frac{1}{|A_i|}\sigma^2(avc)$. We assume that $|A_i|$ is the same for all nodes i. Since, in Eq. (3.4), $\mathrm{E}[N\sigma^2(avc')] = N\sigma^2(avc')$for all available capacities in the system, and the second term at the left of the equation evaluates to $(N - |A|)\sigma^2(avc)$, we obtain $N\sigma^2\left(avc'\right) = (N - |A| + 1)\,\sigma^2(avc)$, or

$$\sigma^2(avc') = \left(1 - \frac{|A| - 1}{N}\right)\sigma^2(avc) \tag{3.5}$$

Since Eq. (3.5) holds for any local load balancing operation that is performed by any node in the system, we see that the value of the variance follows a geometric series that converges to zero.

Now we are interested in estimating by which factor the variance decreases over a period of one round. A round is the time interval within which each node of the overlay network is supposed to have performed exactly one load balancing operation. We denote the variance of available capacities at the end of round t as $\sigma^2(avc^t)$ and the standard deviation as $\sigma(avc^t)$. Since there will be N load balancing operations within this period, we obtain the following equation:

$$\sigma^2\left(avc^t\right) = \prod_i \left(1 - \frac{|A| - 1}{N}\right)\sigma^2\left(avc^{t-1}\right) = \left(1 - \frac{|A| - 1}{N}\right)^N \sigma^2\left(avc^{t-1}\right) \tag{3.6}$$

Since $|A|$ is much smaller than N, we can use the approximation $(1+\frac{x}{n})^n = e^x$ for large integers n, and obtain the following equation:

$$\sigma^2\left(avc^t\right) = e^{-(|A|-1)}\sigma^2(avc^{t-1}) \tag{3.7}$$

and

$$\sigma\left(avc^t\right) = e^{\frac{-(|A|-1)}{2}}\sigma(avc^{t-1}) \tag{3.8}$$

Therefore, for the period of one round, the variance of available capacities reduces by a factor of $e^{|A|-1}$, and the standard deviation reduces by a factor of $e^{\frac{|A|-1}{2}}$ (derived from Eq. (3.7)). We call the multiplier that indicates the change of the standard deviation in Eq. (3.8) the convergence ratio. Then, we can say that diffusive load balancing has a convergence ratio equal to $e^{\frac{-(|A|-1)}{2}}$ (or $e^{\frac{1-|A|}{2}}$) when it uses the *DI* algorithm.

In the following, we discuss the convergence speed for the other algorithms: the *DI*, *SI* and *RI* decision algorithms. These algorithms are more practical for real systems compared to the *CB* algorithm. Using the *CB* algorithm, an operating node works as a receiver for some neighbors and a sender for the others at the same time. This does not occur for the algorithms considered now. However, they are expected to provide slower convergence than the *CB* algorithm, because at the end of a load balancing operation by a node i, the available capacities of the nodes within its neighborhood would be less uniform than in the case of the *CB* algorithm. For example, in the case of the *SI* algorithm, half of the times, there is no change in the load distribution, namely when node i is under-loaded and we have $\sigma^2(avc') = \sigma^2(avc)$. If node i is overloaded, its available capacity will reach the neighborhood average and the available capacity of each under-loaded node will be increased by smaller amounts. If we consider the load change on the overloaded node and ignore the changes of the under-loaded nodes, we obtain the formula $N\sigma^2\left(avc'\right) = \frac{1}{|A_i|}\sigma^2\left(avc\right) + (N-1)\sigma^2\left(avc\right)$. We combine the above two formulas into the equation $N\sigma^2\left(avc'\right) = \frac{1}{2}N\sigma^2\left(avc\right) + \frac{1}{2}\left(\frac{1}{|A_i|}\sigma^2\left(avc\right) + (N-1)\sigma^2\left(avc\right)\right)$– note that we have ignored here the difference between the global average and the average within the neighborhood. Then, we obtain $\sigma^2\left(avc'\right) = \left(1 - \frac{|A_i|-1}{2|A_i|N}\right)\sigma^2\left(avc\right)$. If we assume $|A_i|-1 \approx |A_i|$, then, we obtain $\sigma^2\left(avc'\right) \approx \left(1-\frac{1}{2N}\right)\sigma^2\left(avc\right)$, and $\sigma^2\left(avc^t\right) \approx e^{-\frac{1}{2}}\sigma^2\left(avc^{t-1}\right)$. Therefore we expect that the standard deviation of available capacities is reduced over the period of one round by a factor of $e^{0.25}$ approximately. This would be similar for the *RI* algorithm.

The convergence speed of the *DI* algorithm is more difficult to estimate, since during a single load balancing operation, several sender-receiver pairs exchange parts of their load. For each of the resulting load transfers, one of the partners will reach the neighborhood average, but it is difficult to estimate how many pairs will be identified and how much the load change of the other partner contributes to the reduction of the variance. However, since the *DI* algorithm drives more nodes to reach the average of the neighborhood during one operation than the *SI* or *RI* algorithms, it is clear that the convergence speed of this algorithm is expected to lie between the speeds of the *CB* and *SI* algorithms.

3.3.2 Simulation studies

Next, we study the convergence of load balancing with simulation experiments. The convergence of a classical synchronous diffusive scheme, such as the one in [18, 23], was evaluated by a convergence factor γ, which is the smallest reduction of the variance of loads during one operation. Therefore, the convergence time derived from the factor (i.e. $t \approx \frac{1}{\ln \gamma}$) is the maximum time that the scheme uses. We use a different method to investigate the convergence of an asynchronous scheme. In our experiments, the convergence of the scheme is measured by the convergence ratio r_t during round $t . r_t$ is the ratio of $\sigma(avc^t)$ to $\sigma(avc^{t-1})$ where $\sigma(avc^t)$ is the standard deviation of available capacities at the end of round t. Therefore, r_t is also the reduction ratio of the standard deviation of available capacities. Using this method, the progress of load balancing can be displayed.

These experiments use a simulated P2P system which is the modified version of a clustered P2P system called "eQuus" [28], where each cluster has only one node. This system has a skip-list structured overlay network. For example, for a system with N nodes, all nodes are first connected into a ring according to the ascending order of their IDs. In addition, each node has fingers pointing to the nodes at 2^k positions further down on the ring ($k = 0, 1, \ldots \lfloor \log_2 N \rfloor$). This structure is similar to those used by many P2P systems (e.g. Chord, Pastry, or DPTree), where the time and message complexities of a search can be maintained at $O(\log N)$. In the following experiments, the simulated systems have 1,000 nodes. All 1,000 nodes have the same capacity of 10 requests per second, and initially, the available capacities of nodes are uniformly distributed in the range of [0, 10], with a mean of 5 which leads to a standard deviation of 2.88.

We observed that, for a given decision algorithm, the convergence ratios in different rounds are different. During the first round, the algorithm has the smallest convergence ratio with the largest proportion of loads transferred, and the standard deviation of available capacities drops very rapidly. In the following rounds, the algorithm has slower convergence ratios with fewer loads transferred. We also observe that, after the experiment runs for 10 rounds, there are very few loads transferred, and the standard deviation of available capacities approaches zero. We say that the system is in the globally balanced state. Figure 3 shows the effectiveness of these decision algorithms during the first five rounds. The measurements in the figure are averaged over 20 simulation runs, each. For each measurement, the mean and the 95 % confidence interval of the mean are displayed.

Figure 3 shows that these decision algorithms converge at different speeds. This confirms the predictions of our analysis above. Among these algorithms, the *CB* algorithm converges most rapidly. It has a convergence ratio r_1 close to 0.002, which indicates that the standard deviation of available capacities drops by 99.8 % during the first round. Figure 3a further shows that the standard deviation of the normalized available capacities drops from 0.573 to 0.001 in this round. The normalized available capacities are the available capacities at the end of that round divided by the average workload of the system. In the following rounds, the convergence ratios of the *CB* algorithm remain as small as 0.01. The *SI* and *RI* algorithms converge much slower than the *CB* algorithms (e.g. the standard deviation of available capacities drops by

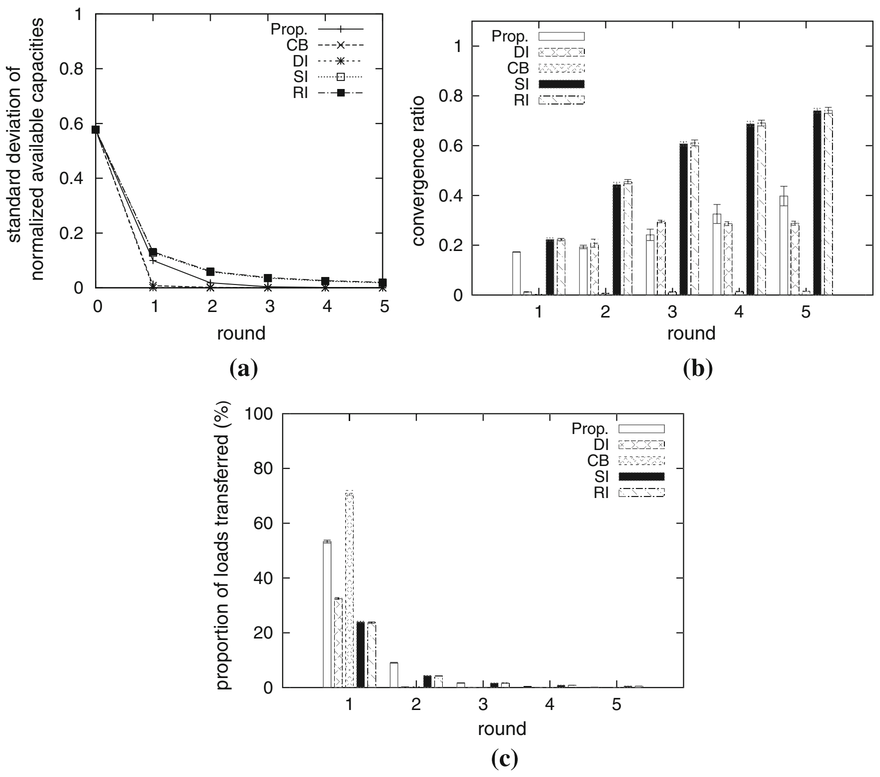


Fig. 3 The progress of the diffusive load balancing with various decision algorithms: **a** standard deviation of normalized available capacities, **b** convergence ratio, and **c** proportion of loads transferred

78 % in the first round with r_1 around 0.22). Among the practical algorithms (i.e. *DI*, *SI* and *RI*), the *DI* algorithm has the strongest average convergence ratio which is close to the *CB* algorithm (with r_1 around 0.02). This observation indicates that resolving multiple pairs of senders and receivers, as done by the *DI* algorithm, improves the effectiveness of diffusive load balancing. The *Proportional* algorithm converges faster than the *SI* and *RI* algorithm but slower than the *DI* algorithm. The data in the figure confirms the predictions of our analysis given above.

The amount of loads transferred between nodes is also collected for evaluating the cost of load balancing. For transferring loads, a system has to spend some processing power of nodes for packing and unpacking objects and some bandwidth of its network links for transmitting the packed objects. A decision algorithm has a higher cost if it decides more transferred loads. The *Proportional* algorithm requires transferring more loads than the *DI*, *SI* or *RI* algorithms. The *DI*, *SI* and *RI* algorithms result in about 35 % of the total loads to be transferred between nodes for the standard deviation of available capacities to drop by 99 % from the beginning.

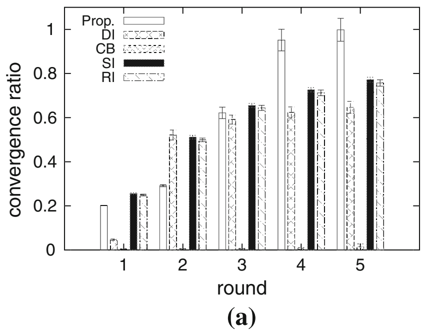
From the perspective of the convergence speed, the diffusive scheme is scalable for P2P systems. We note that, using the diffusive scheme with the *CB* algorithm, the system has a convergence ratio close to $e^{\frac{1-|A|}{2}}$ during one round (where $|A|$is the number of nodes in a neighborhood). In a P2P system, the value of $|A|$ grows along with the increase of the system size, normally logarithmically. Therefore, the larger the system size, the smaller is the factor, and the faster the scheme converges. For example, for a system with a skip-list overlay network, in the case that the system size is doubled from N to $2N$, the convergence ratio of the *CB* algorithm would reduce from $e^{\frac{1-\lfloor \log_2 N \rfloor}{2}}$ to $e^{\frac{1-\lfloor \log_2 N+1 \rfloor}{2}} = e^{\frac{-\lfloor \log_2 N \rfloor}{2}}$ (i.e. by 39.3 %). We investigated the change of convergence ratios in systems with different sizes (e.g. $N = 128$, 256, 512, or 1,024) through simulation experiments. The *DI* algorithm is used. We observed that the convergence ratio r_1 slightly drops when the system size is doubled. After the system has its size increased by a factor of 8 (for example, N increases from 128 to 1,024), the reduction of r_1 becomes close to 42.8 % ($r_1 = 0.021$ for $N = 128$, and 0.012 for 1,024). The reduction is smaller than that calculated for the *CB* algorithm (which is 77.6 % in this case). During the following rounds, the convergence ratios for these systems gradually increase to be as large as 0.3. We can further observe that the proportions of loads transferred between nodes are almost the same for the different systems. Also, the effectiveness of the scheme is not different when these systems are under the same situation of churn (see Sect. 3.6). These results show that the diffusive scheme is scalable to the large-sized systems.

3.4 Network structure

The impact of the network structure on the effectiveness of diffusive load balancing has been intensively studied by considering ring, hypercube, or star topologies (see [17,18,24]). In this section, we consider a structure of random neighborhoods for load balancing operations. We consider two kinds of random neighborhoods. One is called random-graph neighborhoods, which are the neighborhoods in a network with a random-graph topology. Another is called random-walk neighborhoods, which are the neighborhoods that are obtained by dynamic random walks, a different neighborhood for each load balancing operation. A node with a neighborhood of either kind has $\lfloor \log_2 N \rfloor$ neighbors.

We observed that the network structure of a system has an impact on the convergence speed of an algorithm. It has little impact on the *CB* algorithm since we see that the convergence ratio r_1 is around 0.003 for either kind of random neighborhoods. This value is also close to that for the skip-list neighborhoods. When the random-graph neighborhoods, instead of the skip-list neighborhoods, are used, the other algorithms have their convergence speeds degraded. The *Proportional*, *SI* and *RI* algorithms have their convergence ratios increased by 10 % (see Fig. 4a to be compared with Fig. 3b). The *DI* algorithm has its convergence ratio increased by a factor of about 2–4 (for the skip-list neighborhoods, r_1 is around 0.012, and r_5 around 0.30, in Fig. 3b; for the random-graph neighborhoods, r_1 is around 0.046 and r_5 around 0.6, in Fig. 4a).

The situation is different when random-walk neighborhoods are used, as shown in Fig. 4b; only the classic *Proportional* algorithm does not perform well. The standard deviation of available capacities at a level of around 0.05 can not be further reduced even after the experiment runs for 50 rounds. The *SI* and *RI* algorithms perform better with random-walk neighborhoods. Starting from round 2, their convergence ratios



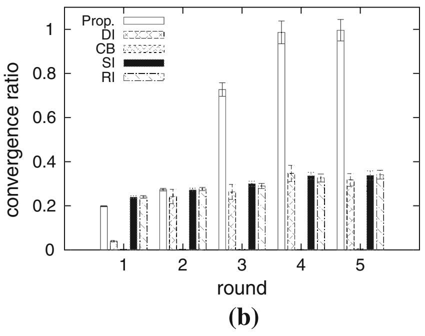


Fig. 4 The convergence ratios of diffusive load balancing with different neighborhoods: **a** in an overlay network with a random-graph topology, and **b** with neighborhoods collected by random walks

become much smaller than those for static overlay network neighborhoods (either the skip-list neighborhoods or random-graph neighborhoods). Using random-walk neighborhoods, a node has more chance to be a sender or a receiver when it runs operations with different neighborhoods each time, and the reduction of the load differences is also larger than in the case of overlay network neighborhoods. Using random-walk neighborhoods, the *DI* algorithm has similar convergence ratios as those for skip-list neighborhoods. Meanwhile, we observe that the proportion of loads transferred between nodes in a system using the *DI*, *SI* or *RI* algorithm is very similar to that in the previous case (see Fig. 3c).

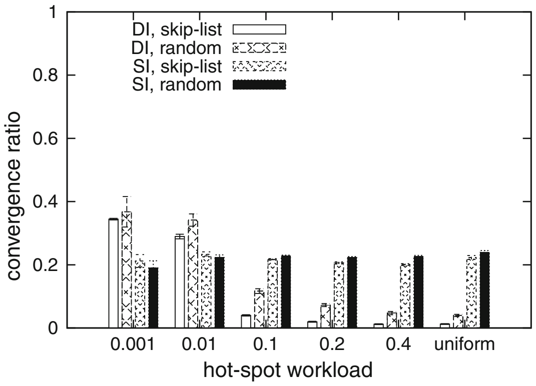
3.5 Workloads with highly skewed distributions

In this section, we investigate the decision algorithm in situations with highly skewed distributions of workloads, like the Zipf distribution in [22]. The simulated systems have different workload distributions. We observed that, in these systems, the convergence speeds of a decision algorithm are different at the beginning of load balancing. Afterwards (e.g. after one or two rounds), the speeds do not change much. We also observed that the *SI* algorithm handles workloads with extremely skewed distributions better than the *DI* algorithm.

To simulate a skewed workload distribution, we considered a situation where a certain proportion of the nodes are hot spots (e.g. 0.001, 0.01, 0.1, 0.2, or 0.4 of all nodes). At the beginning of an experiment, the workload is evenly distributed over all the hot-spots. The *DI* and *SI* algorithms are used for the following experiments. One of two kinds of neighborhoods is used: either the skip-list or random-walks neighborhoods. Figure 5 only displays the r_1 for each of the cases since we observed that, along with the progress of load balancing, the convergence ratios (e.g. r_5) depend on the decision algorithm and the kind of neighborhoods rather than the number of hot spots.

The *SI* algorithm outperforms the *DI* algorithm when the system has an extremely small proportion of hot spots, like 0.001. However, the advantage of the *SI* algorithm drops when the convergence speed of the *DI* algorithm increases with the increase of the proportion of hot spots. For the 0.001 hot-spots case, Figure 5 shows that the

Fig. 5 Comparison of the DI and SI algorithms in systems with hot spots



SI algorithm has an r_1 of 0.19, but the *RI* algorithm has 0.35. For the 0.1 hot-spots case, the convergence ratios of the *DI* algorithm are close to those observed from the previous experiments where the system initially has a uniform workload distribution. However, the speed of the *DI* algorithm will not increase much even if the system further increases the proportion of hot-spots. One reason for this is that, in the 0.001 case, a neighborhood could have one or zero sender with a larger probability at the beginning of the experiment. Since the *DI* algorithm only selects one receiver for a sender, the *DI* algorithm resolves fewer differences between the available capacities of the nodes than the *SI* algorithm which can select many receivers for one sender. Therefore, we conclude that the workload distribution has little impact on the convergence speed of the *SI* algorithm (with convergence ratios of r_1 around 0.2), but, it has an impact on the *DI* algorithm, which reduces the convergence speed when the workload distribution is extremely skewed.

3.6 Churn

In this subsection, we investigate the effectiveness of diffusive load balancing in a system that experiences churn. Churn occurs when nodes join or leave the system. This involves changes of the overlay network and the distribution of workloads. We analyzed the variance of the available capacities in a system with churn.

In the simulated system, churn is realized by adding or deleting nodes from the network. We use the functions provided by the SSim library [29] (i.e. a library used for discrete-event simulation) to schedule the events for joining and leaving nodes. The events of nodes joining or leaving are modeled by a Poisson arrival process where the inter-arrival times of these events follow an exponential distribution. In the following, we use the term churn rate to measure the intensity of churn; it is defined as the fraction of nodes that join or leave the system during one round of load balancing. In this way, the changes of available capacities of nodes caused by churn and the reduction of the differences between these available capacities produced by the diffusive load balancing are measured within the same time period. We assume that, for each node that leaves, there is a node that joins so neither the total number of nodes nor the system's average available capacity changes. For example, when the churn rate is 0.1

in a system with 1,000 nodes, 50 nodes will join and 50 nodes will leave during one round. If the duration of a round is T, the mean inter-arrival time of joining or leaving events is $\frac{T}{50}$. Without load balancing, the standard deviation of the available capacities will increase as time proceeds. This increase depends on the churn rate in the network.

Using the proposed diffusive load balancing scheme in a P2P system with churn, we observed in our simulation experiments that the standard deviation of available capacities depends linearly on the average workload of the system. According to the analysis of Cybenko in [18], when a system with dynamic workload uses a synchronous scheme, the variance of loads on nodes after a load balancing operation can be calculated as $\sigma^2\left(w'\right) = \frac{\sigma_0^2(w)}{1-\gamma^2}$ where $\sigma_0^2\left(w\right)$ is the variance of the dynamic workload introduced during one round, and γ is the convergence factor of the scheme. In a P2P system like Chord, a leaving node passes its workload to its predecessor, and a newly joined node takes over half of the workload of its successor. We assume that all nodes have the same capacity C. The workloads of the nodes are represented by a random variable l with a mean value $\bar{l}$. The system is load-balanced before a change of the network (i.e. a single join or leave event). Therefore, the variance of available capacities introduced by the change is:

$$\begin{aligned}\sigma_0^2(avc) &= \sigma_0^2(C-l) = \sigma_0^2(l)\\ &= \frac{(2\bar{l}-\bar{l})^2 + (\frac{\bar{l}}{2}-\bar{l})^2}{N} = \frac{5\bar{l}^2}{4N}\end{aligned} \tag{3.9}$$

Therefore, according to Cybenko's equation, we suggest that the standard deviation of available capacities is a linear function of the average workload in the system. In our experiments, we observed that the larger the average workload, the larger is the standard deviation. Moreover, for systems that differ only on their average workloads, the ratios of the standard deviation of available capacities to the average workload of the systems (i.e. those with homogeneous nodes) are always the same. Therefore, we define this ratio as the **standard deviation of normalized available capacities** and use it as a parameter for comparing the performance of different decision algorithms.

The proposed scheme controls the standard deviation of available capacities (or normalized available capacities) within a bound. In the following experiments, a system uses the *DI* or *SI* algorithm with the skip-list or random-walk neighborhoods. Figure 6 shows the data for the first 20 rounds of load balancing. We observe that, after a few rounds (e.g. 2 or 3 rounds), the system is in a steady state where the standard deviation of available capacities (or normalized available capacities) and the proportion of loads transferred are steady. We see that the average of the standard deviation of available capacities in the steady state is **bounded**.

Also the size of the **bound** for the standard deviation of available capacities (or normalized available capacities) depends on the decision algorithm. An algorithm with a faster convergence speed can maintain a smaller bound for the system. For example, for the case of the *SI* algorithm, the bound is about 30 % larger than for the *DI* algorithm (0.15 for the *DI* algorithm and 0.2 for the *SI* algorithm). Furthermore, we note that the bounds, for a given algorithm using different kinds of neighborhoods, are not significantly different. Figure 6b indicates that the costs of load balancing are similar

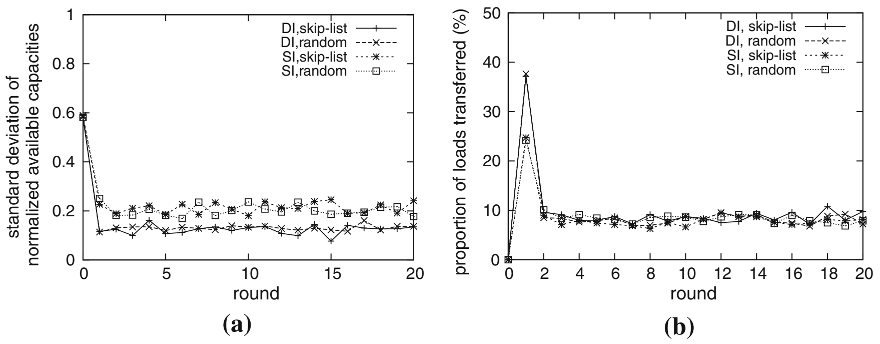


Fig. 6 Effectiveness of diffusive load balancing in a system with churn at a rate of 0.1: **a** standard deviation of normalized available capacities, and **b** proportion of loads transferred

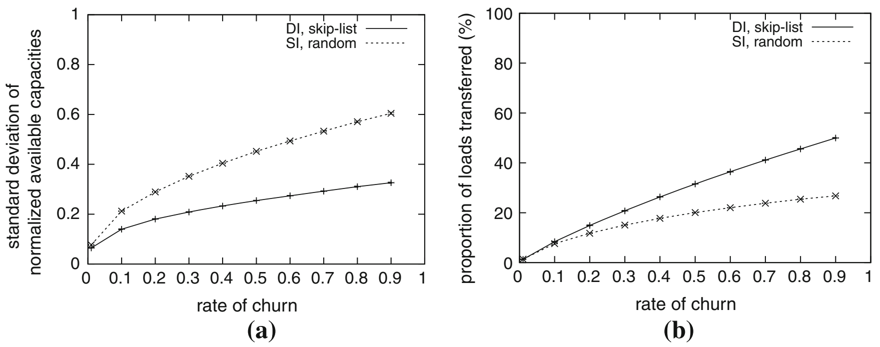


Fig. 7 The bound of the standard deviation of normalized available capacities in systems with varying churn: **a** the standard deviation of normalized available capacities, and **b** proportion of loads transferred

for both decision algorithms. Both these algorithms invoke almost the same proportions of loads transferred when the system is in the steady state. The proportion for a decision algorithm does not depend on what neighbors are used: a skip-list overlay network, or probed by random walks.

The system has its available capacities bounded by a higher value when the churn has a larger rate. Also, the *DI* algorithm brings smaller bounds to the system than the *SI* algorithm; the larger the churn rate, the larger is the difference between the two algorithms. In the experiments of Fig. 7, the churn rates vary from 0.1 to 0.9 with an increment of 0.1. A rate of 0.01 is also used as an exception. The bounds and their 95 % confidence intervals were collected from the experiments which were run 20 times. The average workload of a simulated system is equal to 5 requests/s. Figure 7a shows the bounds when a system has different churn rates. In a system that has a churn rate of 0.1, the bound is around 0.7 (with a bound for normalized available capacity of 0.139 in the figure) for the *DI* algorithm with the overlay network neighborhoods. In this system, few nodes would have an available capacity less than zero and be overloaded. In the case that the system has churn with a rate of 0.9, the bound becomes 1.5 (with a bound of normalized available capacity of 0.3 in the figure), and there are less than 10 % of nodes overloaded. We further observed that the bound is not a linear function of the churn rate. We observed that the size of the bound also depends on whether the *DI* algorithm or *SI* algorithm is used. When the churn rate is as small as 0.01, the bounds for the two algorithms are not significantly different. The difference between these bounds increases when the rate increases. When the churn rate is as large as 0.9, a system using the *SI* algorithm would have a bound twice as large as a system using the *DI* algorithm (0.6 for the *SI* algorithm, and 0.3 for the *DI* algorithm). However, the system has fewer loads transferred when it uses the *SI* algorithm (Fig. 7b).

Although the size of the system does not change in the above experiments, the bounds obtained in these experiments are realistic also when the system size may change. In a real P2P system, the rates of node joining and leaving may be different. In this case, the size of the system will change, e.g. the size will increase if the joining rate is higher than the leaving rate. As long as the rate difference is a small fraction of the leaving (or joining) rate, we expect that the bound of the standard deviation of the normalized available capacity would be very close to the bound obtained for the stationary system state in our simulations. When the system size changes, the expected bound may therefore be obtained by interpolating the bounds obtained by our simulations for the different fixed system sizes.

3.7 Heterogeneous node capacities

Since the proposed load balancing scheme equalizes the available capacities of nodes, systems with heterogeneous nodes capacities can have similar mean response time for their services. In a system without churn, the convergence of the scheme does not depend on whether the nodes are homogeneous or heterogeneous. Therefore, our experiments that use systems without churn in the previous sections are all valid. However, for a system with churn, the variation of workloads introduced by the leaving or joining of nodes depends on the capacities of nodes. The larger-capacity nodes which have larger workloads would induce larger variations than the smaller-capacity nodes (assuming that all nodes have the same available capacity).

In the following experiments, the system has two types of nodes: small-capacity nodes with a capacity of 10 requests/s, and large-capacity nodes with a capacity of 1,000 request/second. There are 1,000 nodes among which 0.1 % are large capacity nodes, and the others are small capacity nodes. The churn rate is 0.1. In Fig. 8, there are several extraordinary points with standard deviations much higher than other points. These points are caused by the leaving or joining of the high capacity nodes. As we observed in Sect. 3.5, the *DI* algorithm has larger convergence ratios in the one-hot-spot case than the *SI* or *RI* algorithm. We propose to modify the *DI* algorithm and allow the sender (or receiver) to distribute its excess workload (or available capacity) to all the under-loaded (or overloaded) nodes in the neighborhood in the case that there is only one overloaded (or under-loaded) node in the neighborhood. This modification is expected to improve the performance of the *DI* algorithm to deal with hot-spots. The solution in [15] that partitions a node into several virtual nodes and locates these virtual nodes in different places of the overlay network is an alternative approach.

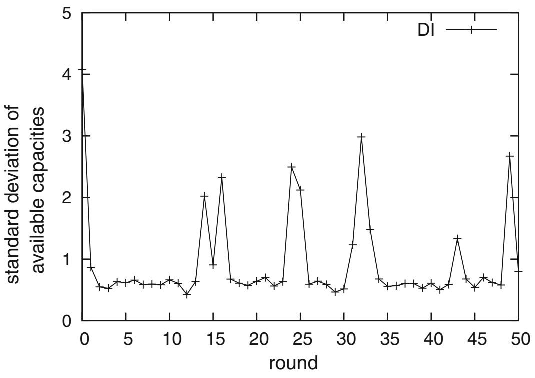


Fig. 8 Effectiveness of the diffusive load balancing in the heterogeneous node system with a churn rate of 0.1

4 Decision algorithms dealing with large-size services

The sizes of services of P2P systems could be fine-grained or large. The size of a service represents the resource requirements of the service, and it is equal to the amount of resources that the service uses to respond a request. In the previous section, we saw that the diffusive scheme can completely equalize the available capacities for a system with fine-grained services. However, working in systems with large-size services, the load balancing scheme can not exactly equalize the available capacities, because the differences between available capacities can not be completely resolved if only large-size services could be exchanged. In this subsection, we propose two decision algorithms for the diffusive scheme to deal with large-size services.

The two algorithms are variations of an algorithm proposed in [19], which used a sender-initiated policy and considered tasks with equal amounts of resource requirements. Our algorithms use a directory-initiated policy and consider the amount of resource requirements of services instead of the number of services on the nodes. They are intended for systems with homogeneous services (i.e., all services have the same resource requirements) and for systems with heterogeneous services (i.e., services have different resource requirements), respectively. We investigate the remaining standard deviation of available capacities and the impact of the sizes of services on this standard deviation. We use the same notations as in Sect. 3 to describe the algorithms. As earlier, we assume that the overlay network would update the destination of a shared object or a virtual server during a load transfer.

4.1 Decision algorithms for large sized services

4.1.1 Homogeneous services

Here we describe a decision algorithm called *DIHomoService* for the diffusive scheme to deal with services of homogeneous size. Similar to the *DI* algorithm, this algorithm calculates the loads for load transfers between pairs of senders and receivers.

```
5        $tr = \left\lfloor \frac{\min\{avc_{A_i} - avc_s, avc_r - avc_{A_i}\}}{L} \right\rfloor$
6        if tr==0 and $avc_s + L < avc_r - L$
7          tr=1
8        if tr > 0
9           decide the transfer with the number of services as tr
10          remove x from SVect
11          remove y from RVect
12       else break
13     else break
14 End of Do
```

Fig. 9 The segment of *DIHomoService* algorithm replacing lines 5–9 of the DI algorithm shown in Fig. 1

It shares with the *DI* the process for identifying the states of nodes as overloaded or under-loaded and selecting a pair of sender and receiver. However, it calculates the number of services to be transferred rather than the amount of load to be transferred. Figure 9 shows a segment of this algorithm. This segment replaces the lines 5–9 of the *DI* algorithm shown in Fig. 1. We assume that the resource requirements of services are equal to L. The number of services for a load transfer is calculated at line 5 where the symbol $\lfloor x \rfloor$ is the floor function of a real number x, equal to the integer part of x.

Following the above procedure, the diffusive load balancing procedure eventually stops. We assume that the system has a static workload and no churn. Cedo et al. [1] presented assumptions for a general model of a partially asynchronous load balancing scheme to converge or stop in a system with the equal-sized tasks. We show that our scheme with the *DIHomoService* algorithm has stronger properties compared with this general model. First, the proposed scheme allows only one node to run an operation at a time in a neighborhood. This guarantees that the load status of a neighborhood is always fresh and correct during an operation. Second, since the *DIHomoService* decides load transfers for multiple pairs of senders and receivers, the proposed scheme has a stronger load balancing power than the general model which uses a sender-initiated policy for load transfers. Third, the *DIHomoService* also guarantees that avc'_s is less than avc'_r for a pair of sender and receiver. Hence, like the general model, our scheme using the *DIHomoService* will eventually stop (in a system with a static workload), and the system enters a globally stable state thereafter.

We further claim that after the system enters a globally stable state, the local load imbalance (i.e. the maximum difference between the available capacities in a neighborhood) is bounded to $2L$. In the case that the decision algorithm of an operation decides no load transfer to be done between two nodes, for example, between the sender $s1$ of *SVect* and the receiver $r1$ of *RVect*, either $avc_{A_i} - avc_{s1} < L$ or $avc_{r1} - avc < L$holds. Then, the inequality $avc_{r1} - avc_{s1} < 2L$ holds. In the neighborhood, there is no receiver that could be located as a receiver for $s1$, and no sender that could be found for $r1$. Hence, in the globally stable state, the local load imbalance (i.e. the difference between the available node capacities between $s1$ and $r1$) is at most $2L$. Therefore, the global load imbalance of the system (i.e. the maximum difference between the

```
5        $W = \{\ \}$
6        $P = \{service \in s\}$
7        Do forever
8            $tr = \min\{avc_{A_i} - avc_s, avc_r - avc_{A_i}\}$
9            $l_{select} = 0$
10           if $l > 0$
11               $l_{select} = \max_{service \in P} \{l_{service} \le tr\}$
12               if $l_{select} > 0$
13                   add v to W if $l_v = l_{select}$
14                   $avc_s = avc_s + l_{select}$
15                   $avc_r = avc_r - l_{select}$
16                   remove v from P
17               else break
18           else break
19       End of Do
20       if W is not empty
21           send instruction to s and r for the load transfer containing the services in W
22           from s from SVect and r from RVect
23       else
24           remove s from SVect
25           if SVect is not empty
26               go to line 4
27           else break
28   End of Do
```

Fig. 10 The segment of *DIHeteroService* algorithm which replaces the lines 5–9 of the *DI* algorithm shown in Fig. 1

available capacities) is bounded by *2LD* where *D* is the diameter of the system (i.e. the maximum of the minimum hop distance between any two nodes).

4.1.2 Heterogeneous services

The *DIHeteroService* algorithm deals with heterogeneous services; its segment that replaces the line 5–9 of the *DI* algorithm is shown in Fig. 10. This segment mainly selects the services from the sender s. This selection prevents a sender (or a receiver) from becoming a receiver (or a sender) after the load transfer (see lines 7–19). In the case that a sender has no services for a receiver (line 23), another sender is selected for the same receiver, and the decision procedure continues (lines 24 through 26).

Using the arguments we used for the *DIHomoService* algorithm, we can show that the load balancing with the *DIHeteroService* algorithm will stop in a system with a static workload. Like the *DIHomoService* algorithm, the *DIHeteroService* algorithm reduces the differences between the available capacities of the nodes in each operation. Also, when the distribution of the loads in the system is unknown, in a globally stable state, the local imbalance is bounded by $2l_{\max}$ where $l_{\max}$ is the maximum resource requirement of the services, and the global load imbalance is bounded by $2l_{\max}D$.

4.2 Analysis of the algorithms

Using simulation experiments, we investigated the two above decision algorithms in terms of their convergence speeds, numbers of load transfers, and the remaining standard deviations of available capacities. We assume that each load transfer requires the same amount of resources, such as CPU or bandwith, even though they may include multiple services. Therefore, a larger number of load transfers indicates a higher cost of load balancing. The impact of the resource requirements of services (i.e. the sizes of services) is also investigated. The simulated system has a configuration similar to the previous experiments. In the following experiments, the system installs large-sized services or small-sized services. These services are randomly distributed over the nodes at the beginning of an experiment, and the average available capacity is 5 requests/s. For example, for a system with large-sized homogeneous services, L is set to 2.5 requests/s for a service, which is of the same order as the node capacity. Therefore, a node can host at most 4 services. For a system with small-sized homogeneous services, L is set to 0.25 requests/s, which is one tenth of that of a large service. A node can host at most 40 such services. For the systems hosting heterogeneous services, services have their resource requirements uniformly distributed between 0 and a preconfigured maximum, e.g., 2.5 requests/s for a system with large-sized services, and 0.25 requests/s for a system with small-sized services.

The decision algorithms converge faster in the systems with smaller services, and the standard deviation of available capacities are smaller in these systems. Table 1 shows the mean value and the 90 % confidence interval (CI) for values collected from 20 runs of experiments. In all of the experiments, the diffusive load balancing stops after a small number of rounds (e.g. at most 4 rounds). The r_1 of the decision algorithms are smaller than their r_2. Furthermore, r_1 for a system hosting small services is smaller than for a system hosting large services. This indicates that small services facilitate load balancing. Since the load balancing operations could select the small services in the heterogeneous systems for further resolving load unbalances, the available capacities of the nodes in these systems can have a smaller standard deviation in subsequent rounds. However, moving services for load balancing in these heterogeneous systems introduces more load transfers. The number of load transfers in a heterogeneous system is about three times larger than in a homogeneous system hosting only the maximum-sized services. From Table 1, we also see that the global load imbalance of a system in the stable state is much smaller than the bound calculated in Sect. 4.1. The predicated global load imbalance is bounded by $2LD$, but the experiments show a value of around L or $2L$.

We observed that, in a system using the diffusive scheme, the sizes of services have little impact on the standard deviation of the available capacities when churn is noticeable. In the following experiments, the churn rate is 0.1 or 0.9. The four systems described above are used. Table 2 shows the bounds of the standard deviation of available capacities and the average numbers of load transfers in each round for

Table 1 Results for the DIHomoService and DIHeteroService decision algorithms with a skip-list overlay neighborhood

	Small services		Large services	
	Homo	Hetero	Homo	Hetero
Number of load transfers				
Mean	1825.9	5414.6	617.65	1608.05
90 % CI	25.44	64.46	5.86	17.17
r_1				
Mean	0.034	0.013	0.141	0.124
90 % CI	0.002	0.001	0.009	0.002
r_2				
Mean	0.88	0.324	0.99	0.99
90 % CI	0.039	0.016	0.005	0.001
Standard deviation of available capacities				
Mean	0.09	0.012	0.49	0.355
90 % CI	0.01	0.001	0.032	0.006
Maximum difference of available capacity				
Mean	0.36	0.139	4.5	2.64
90 % CI	0.047	0.014	0.377	0.116

Table 2 The effectiveness of the scheme in systems with churn

	Small services		Large services	
	Homo	Hetero	Homo	Hetero
Rate for churn: 0.1				
Bound*	0.75	0.805	0.83	0.79
Number of load transferred	841	2,048	122	708
Rate for churn: 0.9				
Bound*	1.68	2.29	1.82	1.68
Number of load transferred	2,609	3,326	817	2,229

* The bound of the standard deviation of available capacities

the four simulated systems. When the churn rate is 0.1, the four systems have their bounds close to 0.8 in their steady states; these bounds are not significantly different (see Table 2). A system has its bound increased when the churn rate increases. When the rate is 0.9, the heterogeneous system hosting small services has a bound of around 2.2, and the other systems have a bound of around 1.6 (see Table 2). These bounds are close to those in systems with fine-grained services (see Sect. 3.6). This indicates that, in a system with churn, the size of services has little impact on the bound of the standard deviation of available capacities. However, the numbers of load transfers are largely different (see Table 2). The systems hosting large services are favored; the load balancing operations introduce fewer load transfers.

Table 3 Comparison of load balancing schemes for P2P systems

Structure	Decision component	Information policy	Transfer policy	Location policy
Distributed directory (d-directory) [8]	Directories	A directory collects load statuses of registered nodes	Directory-initiated	Nodes registered in each directory
k-ary tree [10]	Inner nodes of the tree	Tree-root aggregates load statuses of nodes through the tree structure, and the average load status of the system is disseminated to leaves	Directory-initiated	Nodes in the sub-trees of a decision component
Random probing [12]	Each node	A node collects the load statuses of a set of randomly selected nodes	Sender-initiated	Nodes that were probed
Diffusive scheme	Each node	A node collects the load statuses of nodes in its neighborhood	Directory-initiated	Nodes in the neighborhood

5 Discussion and conclusion

5.1 Comparison of load balancing algorithms

The proposed diffusive load balancing technique is different from other techniques proposed for P2P systems. First, the proposed technique equalizes the available capacities of nodes so that the mean response times of services could be similar. Other techniques equalize other kinds of performance aspect. Some of them (e.g. [2,3,12]) equalize the amount of data or the numbers of virtual servers on nodes, and some others (e.g. [8,10]) equalize the utilizations of nodes. With those techniques, a system that has nodes with heterogeneous capacities may have largely varying mean response times for its services. Second, we analyzed the effectiveness of the proposed scheme in terms of convergence speed and the remaining standard deviation of available capacities in the case of churn. These kinds of analysis display the statistical properties of load balancing more clearly than an analysis based on load imbalance (i.e. the difference between the maximum and the minimum loads of nodes) as in [3,11,12], or the proportion of requests failed or succeeded as in [5,7,8].

Third, the proposed technique uses a diffusive scheme. In order to differentiate the diffusive scheme from the other schemes, we compare in the following the proposed diffusive scheme with three other typical schemes for P2P systems: the *distributed directory*, *k-ary tree*, and *random probing* (see Table 3). These schemes are different in the way they decide load transfers. In the *distributed directory* scheme, the number of directories is pre-configured. The effectiveness of the scheme depends on the number of directories and the interval between the two consecutive load balancing operations of a directory. For example, for a large-size system, the scheme with a small number of directories is similar to a scheme with a central directory. For a system of small size, the scheme with a large number of directories is similar to a distributed scheme using random probing. Different from the *distributed directory* scheme, the *random probing* scheme and the *diffusive* scheme let every node in the system make these decisions. Therefore, the two schemes are more scalable compared with the distributed directory scheme with a fixed number of directories. The *k-ary tree* scheme uses the inner tree-nodes to make decisions for load transfers. Therefore, the number of decision components is a fraction of the number of nodes in the system, and the scheme is scalable.

In terms of the *Information* policy, among the four schemes, only the *k-ary tree* scheme uses a tree structure to aggregate the global load status information and to disseminate this information to each node. As we reviewed in Sect. 2.1, the global information of a system easily becomes stale if the system has a dynamic workload. Moreover, churn in a P2P system induces fluctuations of the tree-structure in addition to variations of workload. This further degrades the accuracy of the global load status information. Furthermore, Shen et al. [7] showed that the *k-ary tree* needs more messages to implement its *Information* policy. The other three schemes all use the load status information collected from a subset of nodes. Also, without using the global information, they are more effective in dealing with the dynamics of the P2P system.

Among these schemes, only the *random probing* scheme uses a sender-initiated *Transfer* policy for a load transfer. The other schemes use a directory-initiated policy where a decision component selects a sender and a receiver for a load transfer. Our previous experiments showed that the directory-initiated policy is superior to the sender-initiated policy.

Among these schemes, the *distributed directory* and *k-ary* schemes spend fewer messages on load balancing operations, but they spend extra messages on constructing the load balancing structure. For example, in a system using the *distributed directory* scheme, the load balancing operations use $2N$ messages in one round in the case that the system has N nodes, including those for load status reports and load transfers. However, these nodes also use $O(N \log N)$ messages to register in the directories, and they change their registrations from time to time for achieving global load-balance. In a system using the *k-ary* scheme, an information operation uses $2(N-1)$ messages, including the messages for load status aggregation and dissemination. This system uses $N \log_k N$ messages in total to resolve load unbalance in the worst case, when half of the nodes are overloaded and all located in the same sub-tree of the root, and the load balancing requests are individually sent to different nodes. However, the procedure that constructs a tree based on a DHT uses $O(N \log N)$ messages. This procedure calls a lookup procedure to map a tree node to a real node in the DHT. Moreover, the message cost on maintaining the tree in a system depends on the churn in the system. The larger the churn rate, the larger the message cost.

The message cost of the diffusive scheme or the random probing scheme) is not impacted by the churn of the system since they do not construct control structures. In a system using diffusive load balancing, the load balancing operations use $O(N \log N)$ messages in one round, including those for collecting load statuses of neighbors and for load transfers. The message cost of the *random probing* scheme depends on the accuracy requirement for the estimation of the load status in the system. The scheme in [12] uses $O(\log N)$ messages (or steps) for each probe, and $O(\log N)$ probes for an operation. Then, in a round, the message cost is $O(N \log^2 N)$ in total. We conclude

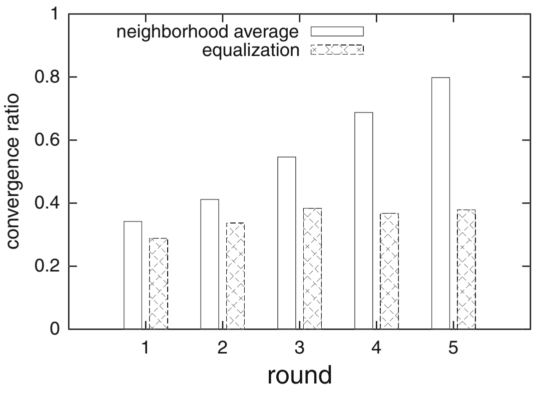


Fig. 11 The convergence ratios of the random probing scheme

that the *diffusive scheme* requires fewer messages than the *random probing* scheme in a round.

To further distinguish the *random probing* scheme proposed in the literature and the *diffusive* scheme, we implemented a simulation of the *random probing* scheme and investigated its convergence speed. Similar to the experiment in Sect. 3.4, the operating node of an operation randomly picks $\lfloor \log_2 N \rfloor$ nodes in the system as the neighborhood for load balancing. The *random probing* scheme uses a sender-initiated policy. In the case that the running node turns out to be a sender (its available capacity is larger than the average available capacity of the nodes in the neighborhood), the running node locates the node with the smallest available capacity as a receiver. We compared two decision algorithms that are popular in *random probing* schemes. One algorithm lets the sender equalize its available capacity with one receiver, in the following called *equalization* algorithm. Another algorithm lets the sender and the receiver have their available capacities equal to the neighorhood average, in the following called *neighborhood average* algorithm. The other parameters of the simulated system are configured as for the experiments in Sect. 3.

Figure 11 shows that the two decision algorithms are different in their convergence ratios. The *equalization* algorithm converges faster than the *neighborhood average* algorithm. However, the *equalization* algorithm induces 15 % more load transfer than the *neighborhood average* algorithm (the *equalization* algorithm moves 45 % and the other moves 30 % of the total workload). Compared with the data shown in Fig. 4b, the convergence speed of the *equalization* algorithm is close to that of the *SI* algorithm using random neighborhood, which is slower than the *DI* algorithm. The experiment further indicates that *random probing* schemes with a sender-initated policy converge much slower than the diffusive scheme with a directory-initiated policy.

5.2 Searching for services that are moved

Load balancing normally implies the movement of services from an overloaded node to an under-loaded node. Such transfer will in general make the searching for these services more difficult. P2P systems normally establish their overlay networks in such a way that it provides efficient searching, possibly through a distributed hash table (DHT). In the following we mention two design alternatives to implement service transfers without interference with the searching algorithm.

One alternative is to foresee for each service (or object) two nodes: (1) the owner node (which is the destination of the searching algorithm for this service), and (2) the host node which stores and executes the service. The service is accessed indirectly; first the owner node is reached. This node forwards the request to the host node. When the service is moved, the address of the host node in the owner node is updated.

Another alternative is the use of virtual servers. Each node contains a certain number of virtual servers, and the virtual servers are the units to be transferred between the nodes. The virtual servers maintain an overlay network among themselves which can be used for searching. When a virtual server is moved, all its neighbor virtual servers will have to update their routing table with the address of the node where the moved virtual server will be located. Therefore the searching is not affected by the load movement. The load balancing algorithm running on a physical node may use the routing tables of its virtual servers to select the neighbors to be used for load balancing. In a sense, this leads to random neighborhoods for load balancing without using random walks; the overhead of random walks is avoided.

5.3 Conclusion

We proposed a diffusive load balancing scheme for P2P systems in this paper. This scheme focuses on equalizing the available capacities of the nodes in the system so that the services on these nodes could have similar mean response times. Nodes in a P2P system asynchronously run the load balancing operations. Since these operations use the structure of an overlay network to identify their neighborhoods, they do not introduce extra overheads for maintaining these neighborhoods that are required for collecting the load information.

The effectiveness of decision algorithms for the diffusive scheme is discussed in detail. The *Complete Balancing* (*CB*) algorithm exactly equalizes the loads on the nodes in a neighborhood, and therefore converges fastest. Also, its convergence speed does not depend on what kind of neighborhood is used: the skip-list neighborhood or the random neighborhood. The *Directory-initiated* (*DI*) algorithm is superior to the *Sender-initiated* (*SI*) or *Receiver-initiated* (*RI*) algorithm since its convergence speed is close to the *CB* algorithm. The convergence speed of the *SI* (or *RI*) algorithm improves when it uses the neighborhoods constructed by random-walks. Especially, in situations with a small number of hot spots (for example, 0.001 or 0.01 of nodes are hot spots) these algorithms perform better than *DI*. This advantage disappears when the proportion of hot spots becomes larger (e.g. 0.1). The diffusive scheme is very scalable; its convergence speed increases when the size of a system increases. The message complexity of the load balancing operation performed by a node is close to $O(\log N)$ in a system with N nodes.

When churn occurs, the diffusive scheme is able to keep the standard deviation of available capacities within a bound. The bound is larger when the system has a larger churn rate. The *DI* algorithm brings the available capacity into a smaller bound than the *SI* (or *RI*) algorithm. We note that in a system with homogeneous nodes, the bound of the available capacities is a linear function of the average workload in the system.

We also designed load balancing decision algorithms for systems with large-size services; however, the sizes of services have little impact on the effectiveness of load balancing. The scheme converges faster for smaller-sized services since its operations invoke more services to be transferred, and the reduction of the differences between the available capacities is larger. However, when churn occurs, the bound of the standard deviation of the available capacities is not affected by the sizes of services. For a specific churn rate, the systems with different large-size services have similar bounds. The impact of the churn rate on the bound is much larger than the impact of the sizes of services.

The proposed diffusive load balancing scheme could be augmented with other mechanisms to guarantee the quality of services. One quality of service requirement is the mean response times of services. When a system uses the proposed diffusive load balancing, the services that the system provides have similar mean response times. From the perspective of performance, services may have different requirements on their mean response times. When a mechanism that controls the mean response times for individual services is combined with the proposed load balancing scheme, the services in a system could have different mean response times while the nodes have the same available capacity.

References

1. Cedo F, Cortes A, Ripoll A, Senar MA, Luque E (2007) The Convergence of Realistic Distributed Load-Balancing Algorithms. Theory Comput Syst 41(4):609–618
2. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. SIGCOMM Comput Commun Rev 31(4):149–160
3. Ganesan P, Bawa M, Garcia-Molina H (2004) Online balancing of range-partitioned data with applications to peer-to-peer systems. In: Proceedings of the thirtieth international conference on very large data bases, vol 30, Toronto, Canada, August 31–September 03, 2004
4. Corradi A, Leonardi L, Zambonelli F (1999) Diffusive load-balancing policies for dynamic applications. IEEE Concurr 7(1):22–31
5. Ledlie J, Seltzer M (2005) Distributed, secure load balancing with skew, heterogeneity and churn. In: Proceedings of 24th INFOCOM 2005, pp 1419–1430, March 2005
6. Willebeek-LeMair MH, Reeves AP (1993) Strategies for dynamic load balancing on highly parallel computers. IEEE Trans Parallel Distributed Syst 4(9):979–993
7. Shen H, Xu C (2007) Locality-aware and churn-resilient load-balancing algorithms in structured peer-to-peer networks. IEEE Trans Parallel Distributed Syst 18(6):849–862
8. Surana S, Godfrey B, Lakshminarayanan K, Karp R, Stoica I (2006) Load balancing in dynamic structured peer-to-peer systems. Perform Eval 63(3):217–240
9. Mohamed-Salem M-V, v Bochmann G, Wong JW (2003) Wide-area server selection using a multi-broker architecture. In: Proceedings of international workshop on new advances of web server and proxy technologies, Providence, USA, May 19
10. Zhu Y, Hu Y (2005) Efficient, proximity-aware load balancing for DHT-based P2P systems. IEEE Trans Parallel Distributed Syst 16(4):349–361
11. Vu QH, Ooi BC, Rinard M, Tan K (2009) Histogram-based global load balancing in structured peer-to-peer systems. IEEE Trans Knowl Data Eng 21:4–595608
12. Bharambe AR, Agrawal M, Seshan S (2004) Mercury: supporting scalable multi-attribute range queries. In: Proceedings of the SIGCOMM '04. ACM, New York, NY, pp 353–366

13. Karger DR, Ruhl M (2004) Simple efficient load balancing algorithms for peer-to-peer systems. In: Proceedings of the sixteenth annual ACM symposium on parallelism in algorithms and architectures, Barcelona, Spain, June 27–30, 2004. SPAA '04. ACM, New York, NY, pp 36–43
14. Zhong M, Shen K, Seiferas J (2008) The convergence-guaranteed random walk and its applications in peer-to-peer networks. IEEE Trans Comput 57(5):619–633
15. Qiao Y, von Bochmann G (2009) A diffusive load balancing scheme for clustered peer-to-peer systems. In: Proceedings of the 2009 15th ICPADS. IEEE Computer Society, Washington, DC, pp 842–847
16. Li M, Lee W, Sivasubramaniam A (2006) DPTree: a balanced tree based indexing framework for peer-to-peer systems. In: Proceedings of ICNP 2006. IEEE Computer Society, Washington, DC, pp 12–21
17. Boillat JE (1990) Load balancing and Poisson equation in a graph. Concurr Comput Pract Exp 2(4):289–313
18. Cybenko G (1989) Dynamic load balancing for distributed memory multiprocessors. J Parallel Distributed Comput 7(2):279–301
19. Jain R (1991) The art of computer systems performance analysis. Wiley, New York
20. Bertsekas DP, Tsitsiklis JN (1999) Parallel and distributed computation: numerical methods. Prentice-Hall, Englewood Cliffs
21. Song J (1994) A partially asynchronous and iterative algorithm for distributed load balancing. Parallel Comput 20(6):853–868
22. Zhao S, Stutzbach D, Rejaie R (2006) Characterizing files in the Modern Gnutella network: a measurement study. In: Proceedings of SPIE/ACM Multimedia Computing and Networking, 2006
23. Xu C, Lau FC (1997) Load balancing in parallel computers: theory and practice. Kluwer Academic Publishers, Boston
24. Hui CC (1996) A hydro-dynamic approach to heterogeneous dynamic load balancing in a network of computers. In: Proceedings of ICPP 1996. IEEE Computer Society, Washington, DC, p 140
25. Cortés A, Ripoll A, Cedó F, Senar MA, Luque E (2002) An asynchronous and iterative load balancing algorithm for discrete load model. J Parallel Distributed Comput 62(12):1729–1746
26. Jagadish HV, Ooi B Ch, Vu QH (2005) BATON: a balanced tree structure for peer-to-peer networks. In: Proceedings of the 31st international conference on very large data bases (VLDB '05), Endowment, pp 661–672
27. Qiao Y, (2012) Using a diffusive approach for load balancing in peer-to-peer systems. Chapter 4, Section 4.2.1.2, University of Ottawa, Dissertation, 2012, pp 65–74
28. Locher T, Schmid S, Wattenhofer R (2006) eQuus: a provably robust and locality-aware peer-to-peer system. In: Proceeding of sixth IEEE international conference on peer-to-peer computing (P2P'06), pp 3–11
29. SSim library for discrete event simulation. http://www.inf.usi.ch/carzaniga/ssim/index.html