Using Diffusive Load Balancing to Improve Performance of Peer-to-Peer Systems for Hosting Services

Ying Qiao, Gregor Bochmann

Using diffusive load balancing to improve performance of peer-to-peer systems for hosting services

Ying Qiao, Gregor v. Bochmann

School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada
{yqiao074, bochmann}@site.uottawa.ca

Abstract. This paper presents a diffusive load balancing algorithm for peer-to-peer systems. The algorithm reduces the differences of the available capacities of the nodes in the system using service migrations between nodes in order to obtain similar performance for all nodes. We propose algorithms for handling homogeneous services, i.e., services with equal resource requirements, and for heterogeneous services, i.e., services with diverse resource requirements. We have investigated the effect of load balancing in a simulated peer-to-peer system with a skip-list overlay network. Our simulation results indicate that in case that the churn (nodes joining or leaving) is negligible, a system that hosts services with small resource requirements can maintain equal performance for all nodes with a small variance. In case that churn is high, a system that hosts homogeneous services with large resource requirements can maintain equal node performance within a reasonable variance requiring only few service migrations.

Keywords. Load balancing; diffusive load balancing; peer-to-peer systems; distributed resource management.

1. Introduction

Peer-to-peer nodes are different by their resource capacities, geographic region, or on-line time (i.e., time of being part of the peer-to-peer system). This diversity could cause performance issues. For example, some peer-to-peer requests are delayed or even lost by some nodes while other nodes are idle. Load balancing schemes, such as [2], [5], [7] and [8], are proposed to dynamically reallocate nodes or shared objects in the system. Therefore, the services accessing shared object could have a short mean response time.

We propose two different algorithms in this paper. They are for a diffusive load balancing scheme to decide load movements between nodes in a peer-to-peer system. These two algorithms are the variations of an algorithm proposed in [9], where tasks with the same unit of resource requirements are considered. Our algorithms

implement a directory-initiated policy; they consider the amount of resource requirements of services instead of the number of services on 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 with different resource requirements), respectively. Simulation results indicate that, with the diffusive load balancing, a system with heterogeneous services is able to maintain a small variance of node performance when churn (i.e., node joining or leaving) is negligible. However, when churn is large, a system hosting homogeneous service is able to maintain the performance of the nodes within a reasonable variance and induces fewer service migrations than a system hosting heterogeneous service.

Our scheme is different from other load balancing schemes for peer-to-peer systems from two points of view. First, dynamic schemes like [5] and [8] reduce the variance of the utilizations of nodes. In case that the capacities of the nodes are different, this may lead to a large variation of the response times, even though the node utilizations are equalized. Our scheme reduces the variance of node performance by reducing the difference of the available capacities of nodes. The available capacity of a node is the processing power remains on a node after the node serves all the requests of its services. When a service migrates, the resource requirement of the service is transferred from the load sender to the load receiver, and both of the nodes have their available capacity changed by the same amount. The scheme in [7] decides node movement with different formulas for peer-to-peer systems, where the load of a cluster is shared among all the nodes of a cluster and the number of nodes in the sender and receiver cluster may be different. Second, our scheme does not rely on a specific overlay network structure to aggregate the load status of the system. Hence, our scheme can be adopted in any peer-to-peer system. Like the research in [8], we assume that the overlay network would update the destination of a shared object or a virtual server during a service migration.

We organize the rest of the paper as follows: Section 2 briefly reviews existing load balancing schemes for peer-to-peer systems and the diffusive scheme proposed for parallel computing systems. Section 3 presents our proposed diffusive scheme and its algorithms. Section 4 discusses the results of several simulation experiments, including the speed of load balancing and the number of service migrations involved. We conclude our paper in Section 5.

2 Peer-to-peer load balancing and diffusive load balancing

2.1 Peer-to-peer load balancing

Peer-to-peer load balancing techniques can be distinguished by their different ways of performing load balancing operations. Some techniques perform load balancing operations when an object is inserted into a system or when a node joins a system. For example, a newly inserted object is placed on a node with the lowest load among the nodes randomly probed at that time [10], or a newly joined node hosts virtual servers that are taken from overloaded nodes [4]. Some techniques dynamically relocate

objects between nodes that are consecutively connected in a ring or in a list. In order to further improve the load balancing speed, these techniques relocate an under-loaded node by making the node leave its original place and rejoin the network as a consecutive neighbor of an overload node [13] and [14]. Some load balancing techniques relocate virtual servers for load balancing [2], [5], and [8]. This kind of load balancing does not split or merge virtual servers, and adds less overhead to the overlay network than the other techniques. Our diffusive load balancing scheme relocates services like the methods in the last category. Load balancing techniques for peer-to-peer systems can be differentiated by the ways they collect status information and the ways that they select load senders and receivers. We will further compare our scheme with other methods in Section 4.

2.2 Diffusive load balancing

Synchronous and asynchronous diffusive load balancing can be distinguished. In the case of synchronous load balancing, all nodes run a load balancing operation at the same time. As shown in [11], this synchronized operation results in load exchanges that are similar to the diffusion of heat through a solid body. Asynchronous load balancing does not require synchronous operations on all nodes. The scheme in [3] specifies that any overloaded node (called a sender) should take the initiative to send part of its workload to under-loaded nodes (called receivers), and after such a load migration, the workload of the sender should still be larger than that of the receiver(s). Experiments in real systems have shown that diffusive load balancing with immediate neighbors deals well with dynamic changes of the workload [1].

Diffusive load balancing schemes can also be distinguished by how they deal with workloads of various sizes. Several papers assume fine-grain tasks, that is, the sizes of the workloads are very small compared with the resource capacity of a node. In this case, load balancing lets the workloads of all nodes converge to a global average [11]. Papers of [15], [6], and [9] considered that the resource requirement of each task is one (fixed sized) unit. The load balancing operations eventually lead to a global system state which is stable (that is, no further load exchanges occur), although this state is not completely balanced. In fact, the difference of workloads between any two nodes in a neighborhood could be as large as one unit (without leading to a load exchange). Therefore, the global load imbalance in the stable state, defined by the maximal difference of workloads between any two nodes in the network, is bounded by $\left\lceil \frac{D}{2} \right\rceil$ load units, where D is the diameter of the network.

3 Diffusive load balancing scheme for peer-to-peer systems

The proposed diffusive load balancing scheme allows load balancing operations periodically run on each node. An operation undergoes three phases. At first, the operation collects the load status of its neighbors in the information phase. Then, it makes decisions on service migrations in the decision phase. At the end, services are

transferred from load senders to load receivers in the service migration phase. Operations executing on different nodes are not globally 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 describe in the following two algorithms that are used in the decision phase. One algorithm, named DIHomoService, is for a system hosting homogeneous services with the identical resource requirements. Another algorithm, named DIHeteroService, is used for systems hosting heterogeneous services with diverse resource requirements. These algorithms are derived from a directory-initiated (DI) algorithm where the running node works as a directory for locating senders and receivers. Our results in [7] show that the DI algorithm is superior to a sender-initiated or a receiver-initiated algorithm for load balancing.

We use here the following notations. The node that initiates a load balancing operation is called the running node of the operation and denoted as node i, the neighborhood of the operation is the overlay-network of the running node and is denoted as A_i. A node in the neighborhood is identified as node j. The number of nodes in A_i is denoted $|A_i|$. In case that a service, for example m, with resource requirement l_m migrates from node x to y, after the migration, we have the available capacity on nodes as $avc'_y = avc_y - l_m$ and $avc'_x = avc_x + l_m$ where avc_x and avc'_x are the available capacities of x before and after the service migration, respectively.

3.1 DIHomoService: DI algorithm for homogeneous services

The DI algorithm decides service migrations between possibly several pairs of overloaded and under-loaded nodes within the neighborhood of the load balancing operation. The algorithm calculates the average available capacity of nodes in the neighborhood using the formula $\overline{avc_i} = \frac{\sum_{j \in A_i} avc_j}{|A_i|}$. Based on the average available node capacity $\overline{avc_i}$, it classifies the nodes j in the neighborhood as either overloaded (if its available capacity is smaller than $\overline{avc_i}$), under-loaded (if its available capacity is larger than $\overline{avc_i}$), or average loaded. The algorithm stores the overloaded nodes in vector SVect and the under-loaded nodes in vector RVect. Then the algorithm decides service migrations using the decision procedure shown in Fig. 1. A service has its resource requirement equal to l.

Decision procedure: Fig. 1.(a) shows the procedure. For a pair of nodes that have the largest difference of their available capacities among all of the nodes in the two vectors (line 3 and 4), the procedure resolves the load imbalance by calling the selection function shown in Fig. 1.(b) (line 5). The selection function returns the number of services to be transferred. In the case that no service can be transferred, the procedure stops since it will not be able to schedule any other service migration in this operation. In the other case the procedure decides the service migration. Then the procedure goes back to line 2 to find another node pair.

Selection function: The function is shown in Fig. 1.(b). It calculates the required available capacity for the sender and the provided available capacity for the receiver according to the differences between their available capacity and $\overline{avc_i}$ (line 1 and 2). The minimum of the provided and the required available capacities is the load difference that the algorithm should resolve (line 3). In case that the minimum is larger than the resource requirement of a single service, the function returns the integer part of the ratio of the minimum to the resource requirement of a service (line 5). Otherwise, it returns 1 in case that the available capacity of the sender could be still less than that of the receiver right after the service migration. In this way, the algorithm keeps the available capacities of nodes closest to the average.

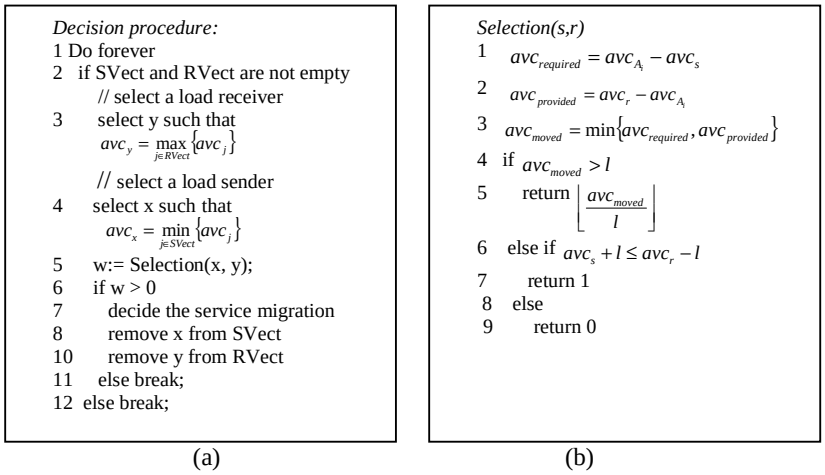

Fig. 1. The DIHomoService algorithm: (a) the Decision procedure, (b) the Selection function

We can see that, when following the above procedure, the diffusive load balancing eventually stops. We assume that the system has a static workload. This means that no new service joins or leaves the system, and the request rates of existing services do not change. We also assume now that the peer-to-peer system has no churn. Research in [15] presents assumptions for a general model of a partial asynchronous load balancing scheme. These assumptions assure that a scheme conforming to the general model is able to converge or stop in a system whose tasks have the same load size. We show that our scheme with the DIHomoService algorithm has stronger assumptions than the general model. First, the proposed scheme serializes the running of its operations in neighborhoods with common nodes. Compared with the assumption of partial asynchronous message passing of the general model, the local serialization guarantees that the load status of a neighborhood is fresh and correct during each operation. Second, the general model assumes a sender-initiated load movement. Since the scheme with DIHomoService decides service migrations for multiple pairs of senders and receivers, the scheme has a stronger assumption by invoking multiple sender-initiated service migration in its operations. Third,

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 with DIHomoService will eventually stop service migrations (in a system with static workload) and the system enters a global stable state.

We further claim that after the system enters a global stable state, the local load imbalance of the system (i.e., the maximum difference of available capacities on nodes in a neighborhood) is $2l$. When the decision algorithm of an operation does not find any service migration to be done between two nodes, for example, between the sender $s1$ of S and the receiver $r1$ of R, either $\overline{avc_i} - avc_{s1} < l$ or $avc_{r1} - \overline{avc_i} < l$ holds as well as $avc_{r1} - avc_{s1} < 2l$. In the case that there are p nodes in R, and $avc(rp) \le \ldots \le avc(r2) \le avc(r1)$, then no receiver could be located as a receiver for $s1$. In the case that there are q nodes in S, and $avc(s1) \le avc(s2) \le \ldots \le avc(sq)$, then no sender could be found for $r1$. Hence, in the global stable state, the local imbalance is the difference of the available node capacities between $s1$ and $r1$ which is at most $2l$.

Because of the local load imbalance, a maximal global load imbalance (i.e., the maximum difference of the available capacities of nodes in the system) can reach the value $2lD$ where D is the diameter (i.e., maximum of the minimum hop distance between any two nodes) of the overlay network. We use an example to derive the global load imbalance. We consider that a node $s1$ in neighborhood A_1 sends services to r_D in the neighborhood A_D in at most D hops. We construct a path connecting the nodes according to the service migrations in the form $s_1 \xrightarrow{A_1} r_1 / s_2 \xrightarrow{A_2} r_2 / s_3 \rightarrow \cdots \xrightarrow{A_D} r_D$ where r_i / s_{i+1} is a receiver in A_i and a sender in A_{i+1}. Since the local load imbalance is bound by $2l$, the global load imbalance between s_1 and r_D is bound by $2lD$.

3.2 DIHeteroService algorithm: DI algorithm for heterogeneous services

The DIHeteroService algorithm deals with heterogeneous services. Compared with the DIHomoService algorithm, the algorithm has a different selection function. The function returns a vector containing the services selected for service migration (Fig. 2.(a)). The function selects services with the minimal resource requirements (line 4) in order to assure that the total resource requirement of the selected services will not lead to a sender available capacity larger than the available capacity of the receiver after the load transfer. After one migration pair has been selected, the procedure removes the sender and receiver from the SVect and RVect and continues the decision phase until no further migration pair can be identified.

Following similar arguments as for the DIHomoService algorithm, we can show that the load balancing with the DIHeteroService algorithm will stop in the case of static workload. However, when the system reaches a global stable state, the local load imbalance might not be the smallest. For example, for the pair $s1$ and $r1$, even when there is no service of $s1$ that could be selected for a service migration, it is possible that there are still some services in other senders that could be migrated to $r1$ to reduce their available capacities. In order to improve the decision procedure, we replace the line 11 and 12 of the decision procedure in Fig. 1. (a) by the segment

shown in Fig. 2. (b). The Decision procedure of the operation stops when there is no sender in S or no receiver in R.

Like the DIHomoService algorithm, the DIHeteroService algorithm reduces the variance of the available capacities of the nodes in each operation. Also, when the distribution of services in the system is unknown, at a global stable state, the local imbalance is bounded by $2l_{\max}$ where $l_{\max}$ is the request rate of a service with the highest request rate, and the global load imbalance is bounded by $2l_{\max}D$.

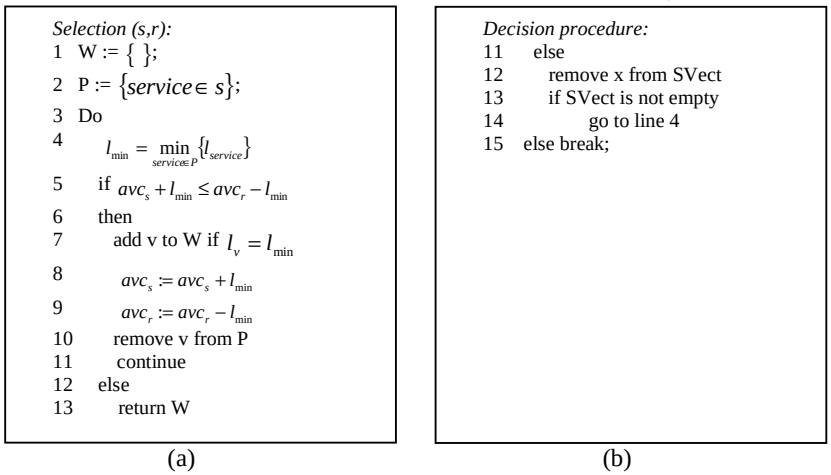

Fig. 2. The DIHomoService algorithm: (a) the Selection function, (b) the segment replacing line 11 and 12 of the Decision procedure in Fig. 1.(a).

4. Experiments

The experimental measurements discussed in this section are obtained from a simulated peer-to-peer system. The system has a skip-list structured overlay network, as described for some classic peer-to-peer systems, like Chord and Pastry. In our simulated overlay network, nodes are connected into a ring, and each node is assigned a position numbered from 0 to $N-1$. Node i at position i will take nodes at positions $(i+2^0) \bmod N, (i+2^1) \bmod N, \cdots (i+2^{\lfloor \log_2 (N-1) \rfloor}) \bmod N$ as its neighbors. In the overlay network, a node has $\lfloor \log_2 (N-1) \rfloor$ out-degree connections and $\lfloor \log_2 (N-1) \rfloor$ in-degree connections, and the diameter of the overlay graph is $O(\log N)$. We have built the overlay network with the simulator of the eQuus system [12] using one node per cluster.

In the simulation, load balancing operations are scheduled by a discrete event simulation library called “Ssim”. During each operation, the running node collects the load status of its neighbors in the skip-list and uses the DIHomoService or DIHeteroService algorithm to decide service migrations. Service migrations are

realized by updating the location of sevices to the nodes in the simulation. The time elapsed during an operation is not simulated since we assume that the neighborhood does not change in such a short time. We call a **round** a simulated time period in which each node runs one load balancing operation, and the standard deviation of the available node capacities and the number of service migrations are collected at the end of each round.

We investigate the effectivness of the diffusive load balancing agorithms in systems hosting homogeneous services or heterogeneous services; the impact of the resouce requirements of services is also examined by configuring small services (i.e., services with small resource requirements) or large services (i.e., services with large resource requirements) to these systems. The effectiveness of the load balancing is evaluated from three points of view. First, the speed that the system approaches the global stable state at is evaluated according to convergence ratios. The convergence ratio γ_τ of round τ is equal to the ratio of $\sigma(avc^\tau)$ to $\sigma(avc^{\tau-1})$, where $\sigma(avc^\tau)$ is the standard deviation of the available node capacities at the end of round τ. The ratio indicates the degree of reduction of the standard deviation of the available node capacities during one round. A smaller convergence ratio indicates a higher load balancing speed. Second, the number of service migrations that occurred for load balancing is concerned. We assume that each service migration spends the same amount of resources, such as CPU and bandwith, even though they may contain different numbers of services. A large number of service migrations indicates a high cost of load balancing. Third, the standard deviation of available node capacities when the system is in a stable state is concerned. This is the degree of load balancing that can be obtained; as we will see, it depends on the degree of churn (as can be expected).

The simulated peer-to-peer system is configured with 1000 nodes. In case of homogeneous nodes, the capacities of nodes (i.e., C) is 10 requests/second. In a simulated system, the sum of the resource requirements of services is equal to half of the total capacity of the system; therefore, the average available capacity of nodes is 5 requests/second (and the average utilization of the system is 50%). The simulated systems install either a workload of large services or of small services. These services are randomly distributed to the nodes at the beginning of the experiments. For example, for a system with large-sized homogeneous services, l is set as 2.5 requests/second for a service, which is in the same order as the node capacity. Therefore, a node can host at most 4 services in the system. For a system with small-sized homogeneous services, l is set to 0.25 requests/second, which is one tenth of that of a large service. A node can host at most 40 services in the system. For the systems hosting heterogeneous services, services have their resource requirements uniformly distributed between 0 and a preconfigured maximum, e.g., 2.5 requests/second for a system with large-sized services, and 0.25 requests/second for a system with small-sized services.

Table 1 shows the results collected from 20 runs of experiements. The mean value and the 90% confidence interval (CI) for the mean of each item are given. The convergence ratios of the first rounds γ_1 of all systems are smaller than the convergence ratios of the seconds rounds. This indicates that, when load balancing first starts, the balancing operations largely resolve the differences of available node

capacities. Furthermore, γ_1 for systems hosting small services is smaller than that for systems hosting large services. This indicates that small services help the load balancing to reduce the differences of the available capacities between nodes. Since load balancing operations could select the small services in heterogeenous systems, the load balancing in these systems is able to further resolve the load unbalance and achieve a smaller standard deviation of available node capacities in subsequent rounds. However, moving services in the heterogeneous systems introduce more service migrations. The number of service migrations in a heterogeneous system is about three times of the number in a homogeneous system hosting the same services. From the Table 1, we also observe that the global load imbalance of the system in the stable state is much smaller than we expected in Section 3. The predicated global load imbalance is bound to $2lD$, but the experiment shows a value around l or $2l$.

Table 1. DIHomoService decision algorithm with skip-list overlay neighborhood and random neighborhood

		Homogeneous system		Heterogeneous system	
		Small services	Large services	Small services	Large services
number of service migrations	Mean	1825.9	617.65	5414.6	1608.05
	90% CI	25.44	5.86	64.46	17.17
γ_1	Mean	0.034	0.141	0.013	0.124
	90% CI	0.002	0.009	0.001	0.002
γ_2	Mean	0.88	0.99	0.324	0.99
	90% CI	0.039	0.005	0.016	0.001
Standard deviation	Mean	0.09	0.49	0.012	0.355
	90% CI	0.01	0.032	0.001	0.006
Global load imbalance	Mean	0.36	4.5	0.139	2.64
	90% CI	0.047	0.377	0.014	0.116

In the simulated system, churn is realized with the occurrences of nodes' joining and leaving. A newly joined node could be positioned at a random place in the ring, and an existing node could be randomly picked to leave the ring. At the beginning of an operation, the running node searches for its neighbors in the skip-list by their positions in the ring. We use a Poisson arrival model to simulate the churn occurrence in the system. We define a relative churn rate, i.e. the churn occurrence rate of the arrival model, as the number of node joining and leaving within a round. For example, when a system with 1000 nodes has a churn rate of 10%, the system would have a total of 100 occurrences of leaving or joining per round, and the mean time interval between two consecutive node joining or leaving is T over 50 where T is the duration of a round. In this way, the changes of available node capacities induced by churn and the deduction of the differences on available node capacities caused by the load balancing are evaluated in the same time duration. A joining node takes over half of the services of its successor after it locates its position in the ring, and a leaving node hands over its services to its successor. We assume that a node leaves and another node joins at the same time, so that, neither the total number of nodes nor the system's average available capacity changes. Without load balancing, the standard deviation of available node capacities always increases, and the degree of the increase

depends on a churn rate. For example, in case the system has a churn rate of 10%, when the system has run for 50 rounds, the standard deviation of the available node capacities is increased by a factor of three. In case the churn rate is 90%, the standard deviation is increased by a factor of 7 after 50 rounds.

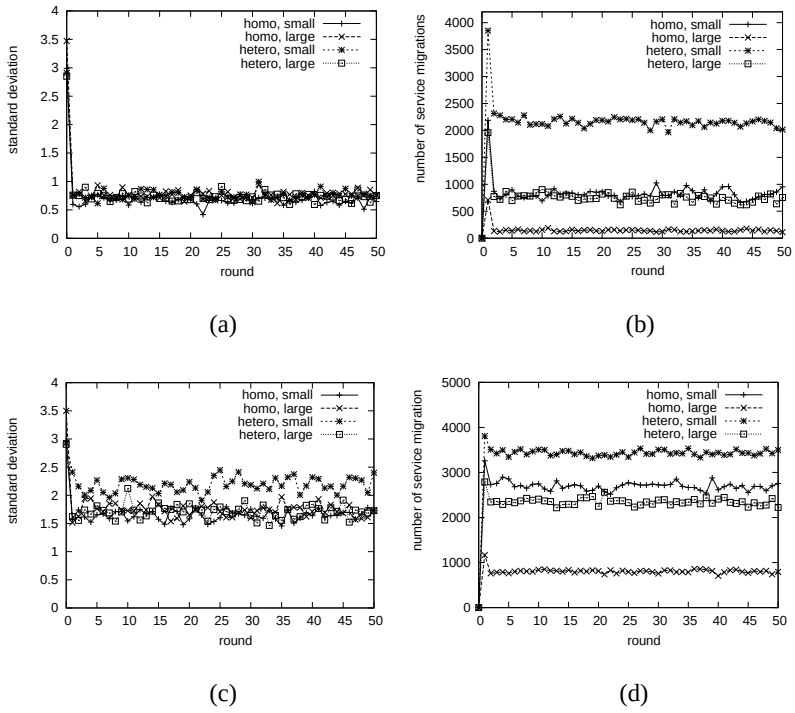

Fig. 3. Load balancing in a system with churn: (a) the standard deviation of available node capacities when churn rate is 10%; (b) the number of service migrations when churn rate is 10%; (c) the standard deviation of available node capacities when churn rate is 90%; (d) the number of service migrations when churn rate is 90%. (Note: "homo" is for homogeneous services, "hetero" is for heterogeneous services, "small" is for services with small resource usage, and "large" is for services with large resource usage)

We compare the effect of the load balancing to the performance of the systems when they experience churn. When the systems have negligible churn, for example, one node joins or leaves in every 2 rounds, load balancing can quickly resolve the load unbalance, and its effectiveness is close to that shown in the previous experiments. Therefore, in this part, the systems with two different churn rates are investigated individually: a low rate as 10% and a high rate as 90%. Fig. 3 shows the standard deviation of available node capacities and the number of service migrations in the first 50 simulation rounds. The standard deviation slightly varies around a certain value as the system evolves, and we say that the system enters a steady state. At a steady state, when the churn rate is 10%, the standard deviations for the four systems are around

0.75 with no significant difference (Fig. 3.(a)). However, the number of service migrations are largely diverse (Fig. 3.(b)). The homogeneous system hosting large services has the fewest number of service migration, and the heterogeneous system hosting small services has the largest number of service migrations. This observation indicates that the systems hosting large services are favored by the load balancing operations with fewer number of service migrations. Fig. 3.(c) shows the standard deviation of available node capacities when the churn rate is 90%. Compared with Fig. 3.(a), the standard deviation is increased. A heterogeneous system hosting small services has a distinct standard deviation of available node capacities around 2.2, and other systems have the standard deviation around 1.6. Fig. 3.(d) shows that a homogeneous system hosting large services has the fewest number of service migrations, and this further confirms our intuition based on Fig. 3.(b).

We compare in the following our scheme with others proposed for peer-to-peer systems. We differentiate them in terms of load balancing policies. Similar to our scheme, these schemes have information, decision and load migration phases. However, some schemes have these phases run separately. For example, the schemes proposed in [13] and [14] require a global load distribution map for their decision phase. Their information and decision phases are separated. Research on dynamic load balancing has shown that this kind of separation could cause the load status information to become stale and thus reduce the effectiveness of load balancing. Also, aggregating a global map induces message overhead. Some schemes, such as [5], use random walks in their information phases, and they normally have a sender-intitiated policy in their decision phases. However, random walks cost extra messages, and a scheme with a sender-initiated policy converges slower than a scheme with a directory-initiated policy. Our work is similar to load balancing with a fixed number of directories [8].

We further compare our research with others in terms of the parameters collected from experiments. The research in the literature consider the maximum difference of loads among nodes [13] or the portion of failed requests [8] in the steady state of a dynamic system. We investigate the standard deviation of load distribution at systems' steady state, the convergence speed of the load balancing, and the number of service migrations during the load balancing. This approach allows us to analyze the effectiveness of load balancing from different perspectives.

5 Conclusions

We proposed a diffusive load balancing algorithm for peer-to-peer systems. The scheme reallocates shared objects on nodes and balances the available node capacities on nodes. Therefore, the performance of nodes is similar. The load balancing operations use the DIHomoService algorithm, i.e., directory-initiated algortihm for systems hosting homogeneous services, or the DIHeteroService algorithm, i.e., directory-initiated algorithm for systems hosting heterogeneous services. The results of the simulation experiments show that, when the churn is negligible, the small services hosted by a heterogeneous system facilitate load balancing. Hence, the node performance of a heterogeneous system has a smaller variance than that of a

homogeneous system for the same services. The results also show that, when the systems have noticeable churn, the variances of the node performance of the systems are not significantly different. However, a churn with a higher rate brings a larger variance to a system. For example, when the churn rate is 90%, the variance of node performance is almost two times larger than that when the churn rate is 10%. The numbers of service migrations are also increased. A system hosting large services with homogeneous capacities always introduces the fewest service migrations.

Our load balancing scheme could be used to improve the performance of a large-scale distributed systems that have characteristics like those of a peer-to-peer system. In such systems, the variance of the delay of service requests is imperative. Also, our research indicates that, in order to have efficient load balancing, the system should have large-sized services in case that the system has heterogeneous services.

References

1. Corradi, A., Leonardi, L., and Zambonelli, F. 1999. Diffusive Load-Balancing Policies for Dynamic Applications. *IEEE Concurrency* 7, 1 (Jan. 1999), 22-31.
2. Zhu, Y. and Hu, Y. 2005. Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems. *IEEE Trans. Parallel Distrib. Syst.* 16, 4 (Apr. 2005), 349-361.
3. Bertsekas, D.P. and Tsitsiklis, J.N., Parallel and distributed computation: Numerical Methods, Englewood Cliffs, NJ, 1999
4. Ledlie, J., Seltzer, M., "Distributed, secure load balancing with skew, heterogeneity and churn," in *Proceedings of INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* vol.2, no., pp. 1419-1430 vol. 2, 13-17 March 2005
5. Shen, H. and Xu, C., 2007. Locality-Aware and Churn-Resilient Load-Balancing Algorithms in Structured Peer-to-Peer Networks. IEEE Transactions on. Parallel Distributed Systems. 18, 6 (June 2007), 849-862.
6. Song, J. 1994. A partially asynchronous and iterative algorithm for distributed load balancing. *Parallel Comput.* 20, 6 (Jun. 1994), 853-868.
7. Qiao, Y. and Bochmann, G. v. 2009. A Diffusive Load Balancing Scheme for Clustered Peer-to-Peer Systems. In *Proceedings of 15th* ICPADS. IEEE Computer Society, 842-847.
8. Surana, S., Godfrey, B., Lakshminarayanan, K., Karp, R., and Stoica, I. 2006. Load balancing in dynamic structured peer-to-peer systems. Perform. Eval. 63, 3 (March 2006), 217-240.
9. Cortés, A., Ripoll, A., Cedó, F., Senar, M. A., and Luque, E. 2002. An asynchronous and iterative load balancing algorithm for discrete load model. *J. Parallel Distrib. Comput.* 62, 12 (Dec. 2002), 1729-1746.
10. David R. Karger and Matthias Ruhl. 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* (SPAA '04). ACM, New York, NY, USA, 36-43.
11. Cybenko, G. 1989. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.* 7, 2 (Oct. 1989), 279-301.
12. Locher, T., Schmid S., and Wattenhofer R., 2006. eQuus: A Provably Robust and Locality-Aware Peer-to-Peer System. In *Proceedings of* P2P '06, (Sept. 2006), 3-11.
13. Vu, Q. H.; Ooi, B. C.; Rinard, M.; Tan, Kian-Lee, "Histogram-Based Global Load Balancing in Structured Peer-to-Peer Systems," *Knowledge and Data Engineering, IEEE Transactions on* , vol.21, no.4, pp.595-608, April 2009
14. Li, M.; Lee, W.-C.; Sivasubramaniam, A.; , "DPTree: A Balanced Tree Based Indexing Framework for Peer-to-Peer Systems," *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on* , vol., no., pp.12-21, 12-15 Nov. 2006
15. F. Cedo, A. Cortes, A. Ripoll, M. A. Senar, and E. Luque. 2007. The Convergence of Realistic Distributed Load-Balancing Algorithms. Theory of Computing Systems 41, 4 (December 2007), 609-618.