# Applying a diffusive load balancing in a clustered P2P system

Ying Qiao
University of Ottawa
Ottawa, Canada
1-613-562-5800
yqiao074@site.uottawa.ca

Gregor v. Bochmann
University of Ottawa
Ottawa, Canada
1-613-562-5800
bochmann@site.uottawa.ca

## ABSTRACT
Although a large number of users are using P2P systems, the ability of these systems to provide services with quality is questioned. A load balanced P2P system can provide services with smaller failure rate and better performance; hence, service quality of the system can be improved. Cluster systems have been adopted for services which are tolerant to faults. Although a cluster structure improves the reliability and robustness of a P2P system, the load unbalancing problem still remains because of the heterogeneities of nodes and requests. Existing dynamic load balancing mechanisms in P2P systems require extra connections and overhead on aggregating the load status from the nodes. We propose diffusive load balancing in a clustered P2P systems, where a global balance is achieved through balancing the neighbourhoods of all clusters within the existing overlay network. We simulated three load balancing schemes: directory-initiated, sender-initiated, and receiver-initiated; from an initially unbalanced situation, the results show that diffusive load balancing can achieve a global balance comparable to a centralized directory scheme, and the distributed directory-initiated scheme provides better results than the sender- or receiver-initiated schemes.

## Categories and Subject Descriptors
C.4 [**Performance of Systems**]: design studies; C.2.4 [**Distributed Systems**]: Distributed application.

## General Terms
Design, Management, Performance

## Keywords
Load balancing, diffusive load balancing, peer-to-peer systems, distributed algorithms, dynamic resource allocation, performance management, server clusters, clustered peer-to-peer systems

## 1. INTRODUCTION
A Peer-to-Peer (P2P) system is a form of distributed computing system with computer nodes located in the Internet. Normally, a P2P system can be decomposed into two layers: the overlay network layer and the application layer. The former connects all nodes and provides lookup function to locate nodes; the latter performs the actions of an application, for example, file downloading in a file system, or stream delivery in a video system. Figure1 shows a P2P file downloading path between two peers in a file sharing application with its overlay layer network and the Internet below.

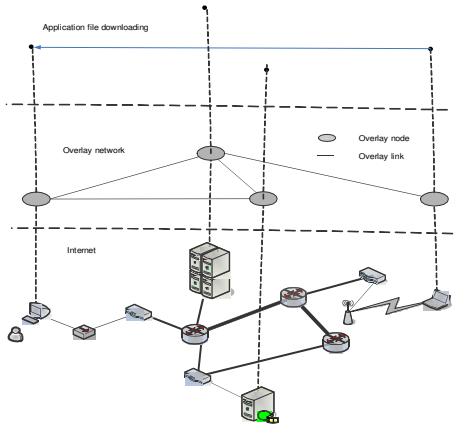



**Figure 1. An example of a file downloading path in a file sharing P2P application**

The overlay network of a P2P system can be constructed using a structured or unstructured architecture. An unstructured architecture randomly connects nodes, and the network is resilient and robust to node failures; but, lookup messages for finding objects are broadcast which takes a large amount of message transmissions and time. A structured architecture associates nodes according to the associations of objects stored in them; lookup messages are routed among nodes according to their associations, and objects are located with a relatively low message and time complexity, e.g., *O(logN)* in Pastry and Chord, where N is the number of nodes; $O(dN^{1/d})$ in CAN, where *N* is the number of nodes, and *d* is dimension for the torus of a CAN; but, the network takes some time to reach a stable state when nodes leave and join the network.

The characteristics of P2P systems are studied mainly for better understanding them and further improving their performance. Measurements from systems in the real world, e.g. KaZza, Gnutella, and PPLive, show that, millions of nodes could participate in an overlay network at a time, and they randomly join and leave the network. These nodes have heterogeneous

capacities of resources: processing power, storage space, and bandwidth; also, the shared objects in the system, i.e., files and videos, have different popularities, and the popularity of a shared object changes with time. These heterogeneities cause load unbalancing among nodes in a P2P application, which results in some requests experiencing long latency while some nodes are idle.

One major issue in a P2P system is the dynamics caused by nodes leaving and joining the system without any notice. Because of this dynamics, called churn, the quality of service provided by overlay networks and applications is not guaranteed; they could experience long delays, or even failures.

Cluster systems have been adopted for services which are tolerant to faults. When a node fails, the service is still available at other nodes in the cluster. A cluster structure improves the reliability and robustness of a P2P system, e.g., eQuus system [1]. Also, in a cluster system, a consistency protocol can be integrated with a group membership protocol, such that it maintains consistency of replicas while the cluster is prone to node failures and recoveries [2], [3]. From the point of view of replica management, in a clustered P2P system, all objects stored in the nodes of a cluster can share one group membership mechanism for managing their replicas; this limits the overhead required for maintaining consistency of replicas.

We study load balancing in such a clustered P2P system. Although, nodes have been clustered in this system, the heterogeneities of nodes and requests still exist. We proposed that, loads are balanced among clusters through moving nodes from lightly loaded clusters to heavily loaded clusters. Installing a load balancing mechanism, the system will have better overall performance. Furthermore, with a load balancing mechanism, clusters can be dynamically resized according to their load statuses.

We propose to apply diffusive load balancing in a clustered P2P systems. Similar to an energy diffusion procedure, with diffusive load balancing, a node balances its load only with nodes directly connected to it. These directly connected nodes compose a local domain for the node. A global balance can be achieved through the balance at all of the local domains. As a diffusive load balancing policy has a small amount of overhead, and it is scalable when the size of a system increases, it is suitable to be applied in a P2P system. To apply diffusive load balancing, a network must cover all nodes, and local domains must overlap. A cluster P2P system satisfies these conditions. Our diffusive load balancing moves nodes from heavily loaded clusters to lightly loaded clusters in the clustered P2P system; loads at these clusters related to the total capacity of their resources are changed. In this way, heavily loaded clusters would have more capacity to satisfy their requests, and service quality are improved.

We present general background regarding load balancing schemes in Session 2. In Session 3, we introduce a clustered P2P system: eQuus, on which we base our simulation. In Session 4, we describe our diffusive load balancing procedure according to their policies and four phases; also, we propose a design of components running this procedure. In Session 5, simulation results are discussed from the comparison of loads distribution before and after the running of a load balancing procedure; the differences of these schemes under different policies and system parameters are discussed as well. In Session 6, we conclude the paper.

# 2. Background

Load balancing is "*the problem of allocation: of mapping and remapping"* workload to "*the physical system"* [4]. On one side, a load balancing scheme determines when and where to move the load; on the other side, the architecture of a node organization in a load balancing scheme determines how nodes communicate and migrate loads for the purpose of load balancing.

## 2.1 Load balancing schemes

According to its load balancing scheme, a distributed system moves workload from heavily loaded nodes to lightly loaded nodes to improve its overall performance [5]. The heavily loaded node is a sender of load, and the lightly loaded node is a receiver. A load balancing scheme is a combination of policies. The policies specify when and where to migrate load for the purpose of load balancing or sharing. Policies can be classified as follows [6]:

- *Transfer* policy: decides whether a node is suitable to initiate a load movement; either as a sender or as a receiver.
- *Location* policy: determines another participant in the load movement after the *Transfer* policy has decided a movement.
- *Information* policy: specifies when and how to collect system state information.
- *Selection* policy: specifies which load should be transferred in a load movement.

Static load balancing scheme: With a static scheme, loads are distributed from senders to receivers through deterministic splits in a random portion or cyclic manner. A static scheme is simple to implement with no effort in collecting system state information and easy to achieve with little overhead. However, this scheme works perfectly only in a homogenous system, where all nodes are almost the same, and all loads are the same as well. A static scheme can hardly catch up and react to the dynamics caused by heterogeneity.

Dynamic load balancing scheme: A dynamic scheme makes decisions based on the system status at the current or recent moment. According to system status information, a node can decide to be a sender or a receiver through a *Transfer* policy, and can decide the peer through a *Location* policy. A *Selection* policy selects a load to be transferred, i.e. small tasks vs. large tasks, or tasks in waiting state vs. those in running state. Dynamic schemes result in better performance when its nodes of the system have heterogeneous capacities of resources, and loads come to the nodes in a random manner.

### *2.1.1 Architecture of dynamic load balancing schemes*

Nodes can be organized in different manners for collecting load information and making load balancing decisions. The typical architectures can be classified into centralized, distributed, and topological.

In a centralized architecture, a central server receives load status reports from the other nodes, and senders ask the server to find receivers for them [7]. Although, systems perform best with this scheme: tasks obtain the lowest mean response time within a narrow range, it is not scalable because the management workload for reporting system status information to the central point

increases with the size of the system. Furthermore, a central information center could be a performance bottleneck and a single point of failure of the system.

In a distributed scheme, each node has a global or a partial knowledge of the system status, and it can locally decide to start transferring a load either into it or out from it. A node could broadcast its node status periodically through out the system [7], [8], or, only when its state is changed [8]. In a distributed scheme with probing policy, when its local status is changed, a node probes part of the nodes in the system and makes decisions based on the received responses. S*ender-initiated* or *receiver-initiated* are the two major schemes. With a probing scheme, a sender or receiver could find its peer through probing a limited number of nodes [9].

Schemes with topological architectures are proposed for systems with a large number of nodes. In schemes with group partitioning [10], [11], and [12], nodes can be partitioned into groups, and load balancing will be performed in each group first, then, a global balancing will be performed when loads are unbalanced among groups. In a scheme with hierarchical architecture [13], nodes are organized into a tree hierarchy, and inner nodes will aggregate the status information of its sub-trees; load balancing is performed from the leaves to the root of the tree through the indication of aggregated status information at inner nodes.

## 2.2 P2P load balancing

Load balancing techniques in P2P systems are facing challenges coming from the characteristics of these systems. First of all, the size of a P2P system is large, which means that a load balancing technique applied to it must be scalable. Second, unlike traditional systems, nodes of a P2P system are not replicas and requests can not be executed in any node. Alternatively, P2P systems place or re-place shared objects optimally among nodes, and overlay routing tables would redirect requests of these shared objects to the right nodes; as a result, the load of the P2P system can be balanced. Combined with techniques of dynamic load balancing, object placement and node placement are two types of load balancing techniques used in P2P systems.

In object placement techniques, objects are placed at lightly loaded nodes either when they are inserted into the system [14] or through dynamic load balancing. In the latter, objects can be stored in virtual servers and moved from nodes to nodes. [15, 16 and 17] adopted a distributed directory approach similar to a load balancing scheme with partitioned group architecture. Each node reports its node status to a directory, and load is balanced in each directory. In order to globally balance the system, a node registers to one of the directories of the system; after it stays there for a duration, it will leave the directory and register another one in turn. [18] proposed a $k-ary$ tree architecture for load balancing; where the inner nodes and root of the tree aggregate load statuses of their sub-trees, and the root disperses the average load status of the system to all nodes down the tree. Accordingly, each node can dynamically identify its relative load situation. In this kind of hierarchical architecture, load can be balanced from the leaves to the root according to the aggregated load information at inner nodes.

In node placement techniques, nodes can be placed or replaced to locations with heavy load. For example, the Mercury load balancing mechanism moves nodes from lightly loaded data ranges to heavily loaded ranges [19]. Nodes are connected into a ring, and each node periodically samples the ring with a random walk, which selects nodes from the routing tables as next hops. According to an estimation value based on samples, a node is able to detect a lightly loaded range, and move there if it is overloaded.

[20] proposed a load balancing mechanism that combines both object placement and node placement in a P2P system. Nodes are connected with a linear link, and each node balances its load with its two consecutive neighbours. If a node has balanced its load with its neighbours already, and it is still overloaded, it will select a lightly loaded node among all nodes in the system to hand over some of its loads. Before moving, the lightly loaded node will shed all its loads to its own consecutive neighbours. The load balancing operations occur when a data object is inserted or deleted, and nodes are connected through an extra skip list according to their load information on top of the overlay; this requires frequent updates of the skip list when the load situation changes.

## 2.3 Diffusive load balancing

In a diffusive load balancing, a heavily loaded component sheds portion of its load to any of less loaded components in its local domain; including the portion left itself, the total portions can not exceed 1. A diffusive load balancing policy is a policy having three aspects [21]: each component individually performs load balancing; load balancing is achieved locally in the domain of a component; each local domain partially overlaps with other local domains, and, components of the whole system must be covered by domains. From these aspects, we can see that diffusive load balancing policies are simple, where messages for collecting statuses and load migration are only transferred in a local network; also, they are efficient on achieving global balancing with a small amount of message overheads.

Diffusive load balancing policies can be classified according to their specifications in two aspects: decision and load migration. While making a decision, the components evaluates its local state through collecting load statuses from other components in its domain; with a sender-initiated policy, after evaluating itself as overloaded, it initiates a load migration to a receiver in its local domain; with a receiver-initiated policy, the component will initiate a load migration if it is under-loaded. Also, a component could decide senders and receivers in its domain and initiate load migrations among them [distributed]. Load can be migrated from an overloaded component to less loaded components in its local domain, or to components in the global domain. In the latter case, a path is first located from a sender to a receiver, and then load is forwarded along the path through the intermediate nodes. While load is migrated, a component is only allowed to participate in one action, either sending or receiving, which prevents it from receiving or shedding loads multiple times at the same time. After one round running of decision and load migration, the component will reach a local balancing state.

In research of diffusive load balancing, balancing is measured through the difference between loads of each component and the average load of the whole system. When the difference is a small value, e.g. 0.01, the system is said to be balanced. The research on convergence studies whether the given load balancing policy can finally balance a system by a limited number of rounds of local

balancing, and how fast this convergence can be, i.e. the rate of convergence. It has been proved that a diffusion load balancing policy can converge in a homogeneous system [22]; this was generalized to heterogeneous systems in [23]. After each run of the iteration, the difference becomes smaller; a boundary of number of iterations exists for the difference reaching the limit. Networks with different topologies were studied, such as: torus, grid, and hypercube. With a *d*-dimensional hypercube, a policy can converge in *d+1* iterations. These load balancing policies are also studied under the environment where loads dynamically arrive to nodes.

These policies are mainly studied for massively parallel systems, e.g., distributed memory multiprocessor system, or parallel processing system, whose processors are tightly connected to provide high speed computing power. In these systems, the number of components could be as large as thousands; however, the domain of neighbours for each component is small. These systems adopt diffusive load balancing policies to fully use the capacity of their resources and further speed up computations without the managing of a central controller.

# 3. eQuus

As our load balancing will be based on an eQuus system, we introduce it here. eQuus is a structured P2P system based on clusters, where its nodes are organized into clusters according to a proximity metric, and its DHT is constructed among these clusters. The proximity metric could be the geographic distance, or the network distance in the Internet which is measured in number of hops. Unlike other DHT systems, each cluster is identified by a unique ID. Also, the routing tables are constructed based on these IDs. There are up to *k* nodes belonging to the same cluster pointed to by an entry of a routing table. A node can select a node from these *k* nodes to route a lookup message. Meanwhile, the shared objects belonging to a cluster are replicated among all nodes in that cluster.

eQuus has a routing algorithm similar to Pastry, which forwards a lookup message according to prefix matching. Unlike Pastry, a node resolves a lookup message by checking if the hash key of the lookup is located in the range between the ID of itself and its successor. If this is true, the node returns itself as the final results. Otherwise, the node will forward the lookup message to the next hop according to its routing table. The number of steps of a lookup procedure is bound by *O(logN),* where *N* is the number of clusters in the eQuus.

Nodes in an eQuus system have two levels of connections: intra-cluster and inter-cluster. At the intra-cluster level, a node connects with all other nodes in the same cluster. At the inter-cluster level, a node has connections with $k$ nodes in each neighbour cluster included in the routing table, which provides $k$ redundancy for the lookup forwarding, as well as for the application services on top of the DHT overlay. Each node also stores connections to up to $k$ nodes in its predecessor and in its successor clusters. During a lookup procedure, the probability that all $k$ nodes of an entry would disappear at the same time is very low.

In addition to the operations in regular DHT systems, an eQuus system provides two extra operations dedicated to clusters, one is splitting and another is merging.

- Splitting: When a new node joins the eQuus system, it joins a cluster which is the closest to it on the chosen proximity metric. Its joining only changes the membership of this cluster. When the size of the cluster reaches an upper limit, the cluster will start a splitting operation, where half of the nodes will be in a new cluster, and another half will stay in the original cluster. The new cluster takes over half of the hashed keys which are close to the predecessor of the original cluster on the ring. Also, it is identified by an identifier in the middle of two original identifiers. The new cluster updates entries of its routing table by searching for them in the overlay.
- Merging: When the size of an eQuus cluster reaches a lower bound, the cluster will start to merge with its predecessor, where all of its nodes join its predecessor, and its own cluster ID disappears. After merging, the resulting cluster takes over all of hash keys of both clusters. Also, the clusters having the merged cluster as an entry in their routing tables will be notified for its departure.

With this architecture, only when node joining or leaving accumulate to a certain degree, clusters will experience merging or split, and connections associated with these clusters will be updated. From this point of view, an eQuus is robust and resilient to churn.

# 4. Diffusive load balancing for a clustered P2P networks

The load balancing in a clustered P2P system has two levels: intra-cluster, i.e., loads among nodes in a given cluster are balanced, and inter-cluster, i.e., loads among different clusters are balanced. As research has already intensively studied intra-cluster load balancing, we propose to apply diffusive load balancing in the system at the inter-cluster level based on the assumption that intra-cluster load balancing has been implemented inside each cluster.

We adopt node movements instead of object movement for load balancing. Without virtual servers, load balancing through moving objects can only be realized between consecutive clusters in the clustered P2P system; in this way, diffusive load balancing would converge only slowly. With load balancing through moving node, the overhead of maintaining data consistency among a large number of nodes in different clusters for moving objects, and the updating of routing tables in the network is avoided.

## 4.1 Choice of the load index: available capacity

A dynamic load balancing scheme identifies the system status according to a load index at each node; a load index should correctly reflect the amount of loads at a node, and from this index, the performance of a node could be evaluated. It has been reported that the choice of a load index has a large effect on the performance of a system [24]. CPU queue length is generally preferred as a load index [25, 7, and 24] because it has a strong correlation with the mean response time of tasks at the node. Other load indexes include utilization, request-response time, available capacity.

The utilization of nodes can be used as load index in a homogeneous system where the maximum capacities of the nodes are the same: when two nodes have the same utilization, their request-response times are the same. However, this is not the case in heterogeneous systems. CPU queue length can be used in heterogeneous systems; but it is particularly suitable for load sharing which uses static thresholds to determine whether load should be exchanged. Request-response time is used on dispatching requests among nodes by load sharing scheme as well.

We adopt available capacity as the load index of our load balancing scheme. It has been proposed in [27] for balancing CPU and disk storage usage in a digital library. Also, [28] uses a metric derived from available capacity to balance bandwidth usage in a network during the routing of service requests.

We use a M/M/1 queuing model to show that, the average response time at a node is the reverse of the available capacity of a node; this means that, when two nodes have the same available capacity, even if they have different maximum capacities, the mean response times for a given request will be the same at those two nodes. We use the notations in [26] to derive the equation (1), where we have the average response time $E[r]$, the service rate of a node $\mu$, the arrival rate of a node $\lambda$, and the utilization of a node $\rho$. As the service rate of a node is the maximum number of requests it can process per time unit, and the arrival rate is the number of requests that are processed per time unit, we can say that the maximum capacity of a node is its service rate, and the used capacity is its arrival rate; as a result, the utilization of a node, which is the ratio of $\lambda$ and $\mu$, becomes the ratio of its used capacity to its maximum capacity. We have

$$E[r] = 1 / (\mu - \lambda) = 1 / (\text{MaximumCapacity} - \text{UsedCapacity}) = 1 / \text{AvailableCapacity} \qquad (1)$$

Because of this directly relationship between the response time and the available capacity, we use the latter as load index. Since we assume that the nodes in a cluster are load-balanced, we may also assume that they have all the same available capacity within a given cluster. With inter-cluster load balancing, the available capacities of the nodes in all clusters will be close to their average. We do not differentiate requests into multiple classes here.

We can calculate the available capacity of a node with equation (2) after determining its maximum capacity by benchmark tools and its utilization by performance measurements. Then, the available capacity of all nodes in a cluster can be combined into the load index of the cluster: the available capacity of the nodes within the cluster; by this load index, load balancing procedure will make decision on node movement, and consequently, the load indexes among clusters will be changed in the direction to approach the system average.

$$\text{AvailableCapacity} = \text{MaximumCapacity} * (1 - \text{utlization}) \qquad (2)$$

## 4.2 Diffusive load balancing mechanisms among clusters

Using available capacity as load index, each cluster iteratively runs a diffusive load balancing procedure which identifies the state of its own as well of its neighbours, and makes decisions concerning possible load movements with these neighbours. We consider three schemes here: directory-initiated, sender-initiated, and receiver-initiated; they differ in their *Location* policy. We use the traditional meaning of sender and receiver here: a sender is a node will transfer its load out, and a receiver will transfer load in. In the directory-initiated scheme, when a cluster runs the procedure, it locates the senders and receivers among its neighbours; with sender-initiation or receiver-initiation, it locates a receiver only when it is sender or a sender only when it is receiver, respectively.

We describe in the following the diffusive load balancing procedure in terms of four phases:

- *LB triggering*: the execution of a load balancing procedure is triggered by a timeout event or a state change event in a cluster. There is a time duration between two consecutively runs of the procedure, and this duration is pre-configured. The procedure is also activated when a cluster changes its state, either to be a sender or to be a receiver.

- *Load determination*: First, the cluster determines its own load status as well as the load status of its neighourhood through sending probing messages to its neighbours, and waits for responses from them; a probed cluster responds with its load index.

- *Decision:* A parameter, called bound, is used to determine whether a cluster is considered overloaded or under-loaded. First the load average is calculated for all the clusters in the neighbourhood. Then the upper and lower load thresholds are calculated by the formula: threshold = average-load-index * (1 +/- bound). The bound is given in percentage of the average load. A cluster is a candidate receiver (sender) of load if its load index is smaller (larger) than the lower threshold. The purpose of the decision procedure is to identify one or several receiver-sender pairs and send a load transfer request to the receiver of each pair, including as parameters the ID of the selected sender (which is the target for the node movement) and the amount of load it requires to reach the load average (called required capacity). The details of the decision procedures depends on the *Location* policy:

  *Directory-initiated:* the cluster identifies one or several receiver-sender pairs, as appropriate.

  *Sender-initiated*: if the cluster is a sender, then it tries to identify a corresponding receiver in its neighbourhood.

  *Receiver-initiated:* If the cluster is a receiver, then it tries to identify a corresponding sender in its neighbourhood.

- *Load transfer*: After a receiver cluster receives an instruction of node movement, it will select nodes from its own, delete them from its membership list, and let them join the sender cluster. It is important that the node movement should not cause the state of these clusters to be changed to the opposite, e.g., an under-loaded cluster becomes overloaded, or, an overloaded cluster becomes under-loaded. A receiver can only

transfer out the portion which is over the mean, and we call it transferable capacity; in order to avoid this situation, the transferred portion should be close to the smaller one of the required capacity and the transferable capacity.

## 5. Simulation and Discussion

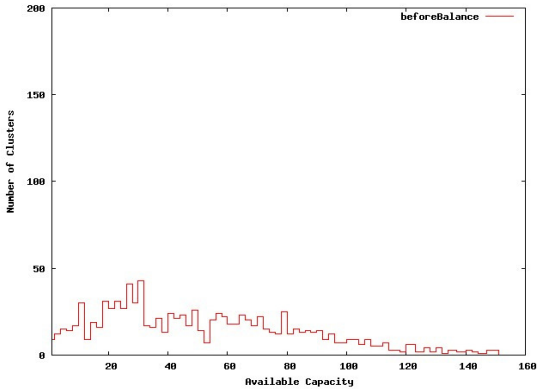
We have built a simulation program for evaluating the performance of the diffusive load balancing procedures described above. Also, we compared them with a central directory scheme, where a single directory collects status information of all clusters in the system and makes inter-cluster load balancing decisions using the same kind of decision criteria based on mean and threshold values. As we are interested in the differences between these different policies, such as their rates of convergence, the effect of the threshold parameters and the impact of churn, we have not taken into account the time and cost of message transmission and node movement (at the current stage of our studies); in our simulation, the LB procedures of the different clusters work sequentially in a random order. We also assume that the capacity lost during node movement is negligible.

In our simulation study, we assume that a cluster has its own intra-cluster load balancing and the load indexes are the same at all of its nodes. To show the speed of load balancing convergence, we assume in our simulation an initially unbalanced load situation, were the load index of the different clusters is uniformly distributed from the lowest value: 0 to the maximum of a cluster. We study the system with fixed (but heterogeneous) traffic loads for the different clusters and we assume that the nodes within a cluster have the same maximum capacity here; heterogeneous node capacities are considered below. Figure 2(a) is an example of the histogram of load among clusters before load balancing, where the load indexes of the clusters are distributed between 0 and 160.

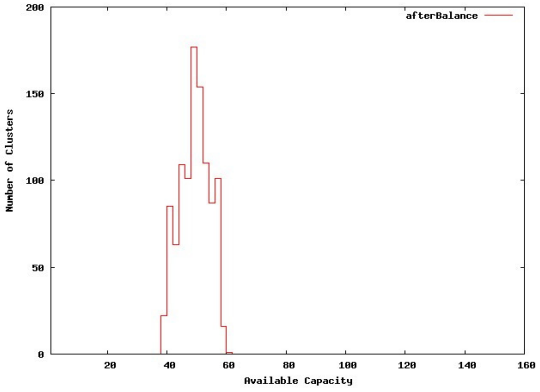
For the purpose of measurement, we insert time points into the simulation, and the time duration between two consecutive points is a measurement round; in one round, each node runs the load balancing. In a *LB decision* phase, a node could identify its neighbourhood as balanced according to the criteria listed in previous section; in this case, no node movement would occur. When there is no node movement among all neighbourhoods in the last round, the system is said to be globally balanced and the simulation will be stopped. In the real system, the LB procedure will run from time to time to handle the possible unbalancing of the system.

### 5.1 Load balancing with homogeneous nodes

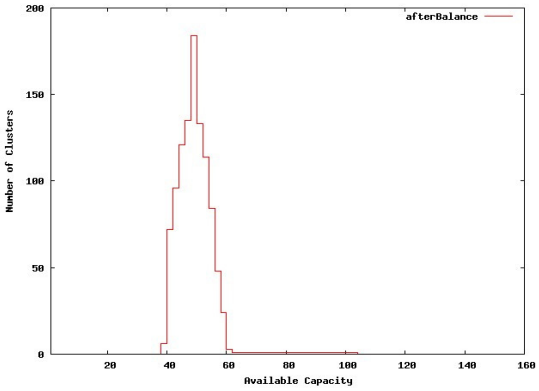
We display the histogram of the load index among clusters after running of LB procedure in Figure 2. The system has 10,000 nodes; the average size of a cluster is configured as 8 (the number of nodes in a cluster then ranges between 4 and 16 under the organization of eQuus), and the load balancing *bound* is plus/minus 20% of the mean. Figure 2 (b) – (e) show the results of these four schemes. The figures show the number of clusters as a function of the load index when the system got into the balanced state (as defined above). Except the receiver-initiate scheme, these schemes balance the load tightly around the mean, and especially, there is no heavily loaded cluster in the system. The directory-initiated scheme has a histogram similar to the central directory scheme, where there is a spike existing near the mean; with the sender-initiated and the receiver-initiated schemes the load index of the clusters is more spread between the lower and upper thresholds. In the receiver-initiated scheme, a cluster makes decision on node movement only when it identifies itself as a receiver. In the case that a cluster is not a receiver, node movements will not occur in its local domain, even when there are overloaded clusters in the domain. This is the reason why with the receiver-initiated scheme some under-loaded clusters remain.



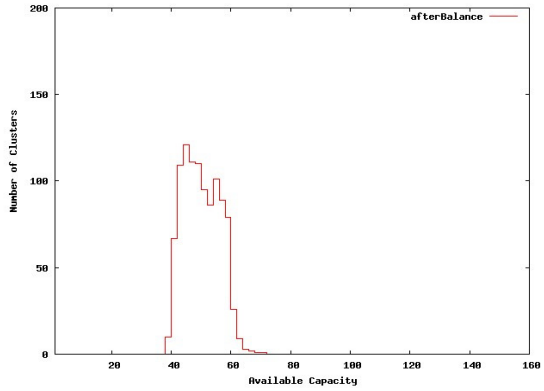
(a)



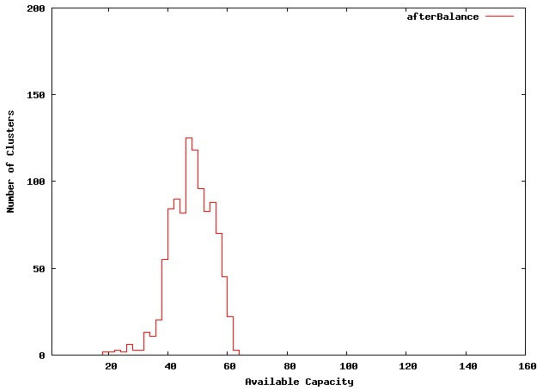
(b)



(c)




(d)




(e)

**Figure 2. The histogram of loads among clusters before load balancing (a), and after load balancing, (b) central directory, (c) directory-initiated, (d) sender-initiated, (e) receiver-initiated**

From Figure 2, we can see that all these schemes can globally balance the load indexes among clusters in the system through balancing their neighbourhood. Next, we give some numerical results about these schemes. Table 1 provides a comparison for the standard deviation of the load index, the delta of the load index which is defined as: maximum – minimum, the rounds of running a scheme to achieve global balancing, and the total number of node movements that occurred during the balancing procedure. These values are the average taken from ten experimental runs for different bounds and different decision policies.

**Table 1. Comparison of load balancing results**

| | 20% | | | | 50% | | | |
|---|---|---|---|---|---|---|---|---|
| | CD | DI | SI | RI | CD | DI | SI | RI |
| std. | 4.88 | 4.54 | 5.78 | 4.42 | 10.87 | 11.2 | 12.25 | 15.84 |
| delta | 20.76 | 29.89 | 36.94 | 57.84 | 50.84 | 80.95 | 72.99 | 95.4 |
| rounds | 3.08 | 1.08 | 3.25 | 4.42 | 2 | 1 | 2.08 | 2.33 |
| node movements | 1667 | 1901 | 1704 | 1556 | 1247 | 1254 | 1160 | 937 |

CD: central directory, DI: directory-initiated, SI: sender-initiated, RI: receiver-initiated

With different bounds configured in the *LB decision* phase, they are all able to balance the load indexes to the mean; however, they have different results. The central directory scheme is an ideal scheme where the delta of available capacity at clusters is the smallest for both bounds; meanwhile, it reaches the balanced state with the smallest number of node movements. A larger bound causes a larger delta of load indexes; in this case, schemes can approach the balancing state fast, i.e., the number of rounds to be balanced with a bound of 50% is less than that for a bound of 20%. Compared with other schemes, the directory-initiated scheme spends less rounds but has more node movements for balancing, which indicates that its fast convergence is based on more load balancing decisions and node movements. The values of the table confirm the exception of the receiver-initiated scheme, already indicated in Figure 1, for which the minimum load index is further away from the maximum; furthermore, the receiver-initiated scheme uses more rounds.

## 5.2 Load balancing with heterogeneous nodes

As the nodes in a P2P system have heterogeneous capacity, the *Selection* policies of the load balancing schemes should be modified: when a scheme selects a node for a receiver, it should considers the required capacity for the sender to reach the mean load index, and pick a node to be transferred having a maximum capacity close to it; we call this a policy with capacity consideration. Through such a policy, the number of node movements to achieve the balanced state would be reduced.

We have configured the simulator with nodes having capacities in the range [10, 5000] with a Pareto distribution shape as 2, and scale as 100 [16]; the other parameters are the same as for Figure 1. Table 2 compares the load balancing results for the different the schemes for two cases: (a) policy with capacity consideration and (b) with random selection of nodes.

We see from Table 2 that the trends observed in a homogeneous system remain present in heterogeneous systems, e.g., the directory-initiated scheme uses a smaller number of rounds to reach a balanced state with more node movements; the receiver-initiated scheme is left with some overloaded clusters when the load balancing procedure stops. Furthermore, the number of node movements is reduced with capacity consideration, as compared with random node selection. For instance, in the directory-initiated scheme, the movements are reduced by almost 15%, and in the sender-initiated scheme, 16%. This indicates that selecting a node with its maximum capacity matching to the required capacity is superior to randomly selection.

**Table 2. Comparison of load balancing results with random and capacity consideration policy**

| | random | | | | capacity | | | |
|---|---|---|---|---|---|---|---|---|
| | CD | DI | SI | RI | CD | DI | SI | RI |
| std. | 30.37 | 31.67 | 37.33 | 49.19 | 23.98 | 30.54 | 35.59 | 54.62 |
| delta | 129.22 | 283.83 | 254.54 | 418.07 | 119.08 | 277.93 | 291.14 | 396.11 |
| rounds | 4.67 | 1.17 | 3.25 | 3.5 | 3.08 | 1.08 | 3.17 | 3.17 |
| node movements | 1702 | 1876 | 1733 | 1531 | 1454 | 1614 | 1498 | 1288 |

CD: central directory, DI: directory-initiated, SI: sender-initiated, RI: receiver-initiated

## 5.3 Impact to Churn

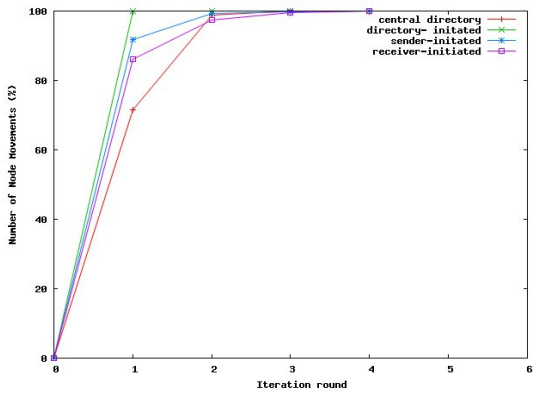
The load balancing procedure causes node movements and introduces extra churn into the system. Churn impacts this peer-

to-peer system at two levels: intra-cluster and inter-cluster. At the intra-cluster level, the departure of an existing node or the arrival of a new one only impacts the cluster membership management. At the inter-cluster level, such changes may cause the split and merge of some clusters; in which case the routing tables of the clusters must be updated. The frequency of splits and merges is related to the frequency of arrival and departure of nodes, and also the average size of the clusters. The node movements due to load balancing have a similar impact at the intra- and inter-cluster level.

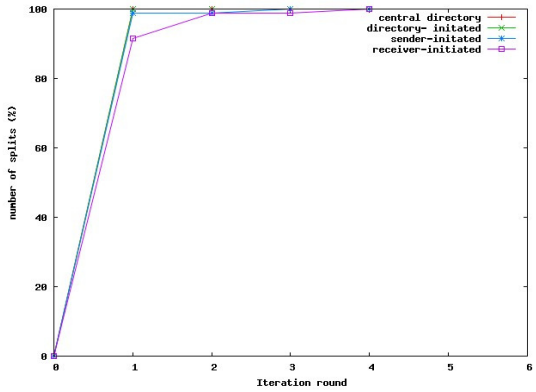
In order to understand the dynamics of our load balancing schemes, we display in Figure 3 the number of node movements (a), the number of splits (b), the drop ratio of the standard deviations of the load index (c), and the drop ratio of the delta between minimum and maximum load index (d) as a function of the number of rounds. The simulator is configured with the same parameters used in Figure 2.

Initially, the load indexes at clusters are dispersed in the range from (0, 160) as seen in Figure 2(a) with large standard deviations and deltas. During each round of the load balancing procedure, the standard deviation and delta of the load indexes become smaller; in fact, most of the changes occur during the first round. The directory-initiated scheme makes 99% of its node movements during the first round; while the sender-initiated and the receiver-initiated schemes only move about 90%. Also, the directory-initiated scheme reduces the delta most quickly during the first round (Figure 3(d)). Most splits of clusters occur during the first round, corresponding to the large portion of node movements (Figure 3(b)). After the first round, there are still some node movements, and changes of the delta of load indexes can be observed from Figure 3(b) and 3(d).

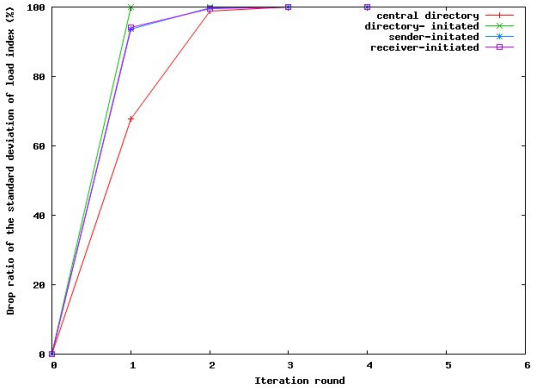
If we associate the load status at each round in Figure 3 with the load state of the system, the initial status could be seen as an extremely unbalanced state and the final status as balanced. Figure 3 is also a picture showing the migration of system from the unbalanced to the balanced state.



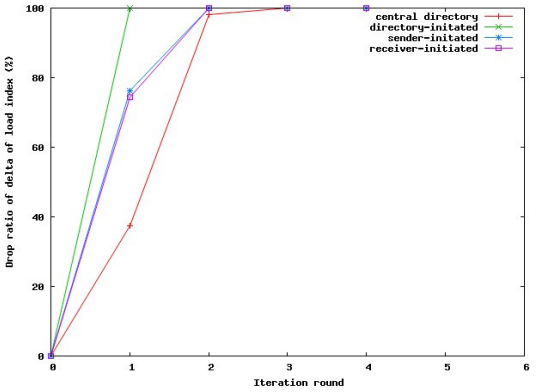

(a)




(b)




(c)




(d)

**Figure 3. The status of the system at the end of each iteration round: (a) number of node movements (%), (b) number of splits (%), (c) drop ratio of standard deviation of load index (%), (d) drop ratio of delta between minimum and maximum load index (%)**

## 5.4 Impact of cluster size to LB results on churn

As the load balancing procedure causes churn (i.e. node movements) in the system, we compare in Table 3 the performance of the load balancing procedure in systems with different average cluster sizes. In a system with large cluster sizes, the probability that a given node movement leads to a split or merge of clusters is smaller than in systems with smaller cluster sizes. Therefore we expect that load balancing is more effective in systems with larger average cluster sizes.

**Table 3. Comparison of network dynamics for different average sizes of cliques**

| | 8 | 16 | 32 |
|---|---|---|---|
| **Central Directory** | | | |
| rounds | 3.25 | 2.92 | 3 |
| node movements | 1438.42 | 1187.92 | 1055.08 |
| Split (%) | 7.33 | 5.77 | 5.37 |
| Merge (%) | 0.03 | 0 | 0 |
| **Diffusive Directory** | | | |
| rounds | 1.58 | 1.25 | 1.42 |
| node movements | 1608.17 | 1331.42 | 1154.42 |
| Split (%) | 8.25 | 6.31 | 5.55 |
| Merge (%) | 0.04 | 0 | 0 |
| **Sender-Initiate** | | | |
| rounds | 5.25 | 5.17 | 4.18 |
| node movements | 1461.58 | 1230.08 | 1081.36 |
| Split (%) | 7.22 | 6.21 | 5.49 |
| Merge (%) | 0.08 | 0 | 0 |
| **Receiver-Initiate** | | | |
| rounds | 5.17 | 4.42 | 4.08 |
| node movements | 1283.08 | 1036.5 | 962.83 |
| Split (%) | 7.22 | 5.39 | 4.62 |
| Merge (%) | 0 | 0 | 0 |

Table 3 shows a comparison for three average cluster sizes. The simulation is based on similar parameters as for Table 2 with heterogeneous capacities and node selection based on capacity consideration; in this case, the simulation is more closed to the real world. The data are aggregated from 10 experiment runnings. We see that for systems with larger cluster sizes the number of node movements is reduced, and the number of splits and merges are also reduced. The table shows that there are few merges happened during the running of the load balancing procedure. A cluster merges itself with its consecutive cluster when its size reaches a lower limit. When the load balancing procedure runs, it reduces the loss of the capacity due to departing nodes (churn) through moving nodes from other clusters. From this point of view, the load balancing procedure is counter-balances churn. Since we have run our simulation without churn, we have not explored this benefit of load balancing. This point requires further study with a dynamic scenarios including churn.

# 6. Conclusion

Load unbalance in a P2P system is caused by the heterogeneities of node capacities and the popularity of their services. Also, churn (dynamic node arrival and departure) could change the load distribution among nodes and introduce randomly unbalanced situations. In a clustered P2P system, the arrival and departure of nodes changes the capacity of the clusters and affects the performance of the services they provide. In order to remediate the unbalanced load situations, due to any reason, we propose to move nodes from clusters with low load to clusters with high load in order to equalize the load situations of the clusters in the P2P system. It is clear that some kind of overhead can not be avoided.

[16] proposed a distributed directory architecture for load balancing in a P2P system; however, this induces extra network connections from nodes to directories. [18] proposed a tree hierarchy for aggregating resource information and managing resources; but this introduces extra overhead for maintaining the tree. Diffusive load balancing [21] simplifies the procedure by achieving a global balance through local balancing procedures. It does not require extra management connections and maintenance infrastructure; this improves the efficiency of resource management in P2P systems. In the diffusive load balancing scheme described in [20], the nodes are organized as a linked list and a node balances its load with its two consecutive neighbours. In order to increase the speed of global convergence, a skip list is introduced to maintain load information about nodes in other parts of the linked list; but this introduces extra overhead for managing the list.

Our diffusive load balancing procedure for clustered P2P system is similar to [20]. However, it uses all those clusters that are included in the routing table as neighborhood of a given cluster. Because of the hierarchical structure of the routing tables, this includes clusters throughout the cluster naming space; therefore we get relatively fast global convergence. The other advantage is that we can directly use the neighbourhood structure provided by the existing P2P overlay structure, which reduces the overhead.

Our diffusive load balancing procedure equalizes load among clusters based on available capacity, which is taken as our load index. Since the available capacity is directly associated with request response time, a system load-balanced based on available capacity has a uniform response time.

Our simulation compares the performance of four different load balancing schemes: a scheme using a centralized directly and three distributed schemes with different decision policies: directory-initiated, sender-initiated, and receiver-initiated. Our simulation results show that the directory-initiated policy is the best distributed decision policy. It results in tight load distributions, similar to those obtained by the centralized scheme. It can also quickly respond to changes of the load index within a small number of rounds, which makes it be a superior scheme in dynamic P2P systems.

Our load balancing procedure moves nodes from lightly loaded clusters to heavily loaded clusters; this movement adds extra churn to the system. We also show that this churn can be reduced through adjusting system parameter; for example, configured with a larger average cluster size, the number of node movements and cluster splits is reduced. However, the dynamics of additional churn introduced by load balancing, in the presence of traditional churn through arriving and departing nodes, requires further exploration. We will also study the effect of running the load balancing procedure concurrently on several clusters within the system.

# 7. Reference:

[1]: T. Locher, S. Schmid, R. Wattenhofer, "eQuus: A Provably Robust and Locality-Aware Peer-to-Peer System," *p2p*, pp. 3-11, *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06),* 2006

[2]: J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution." In *Proceedings of the 1996 ACM*

*SIGMOD international Conference on Management of Data* (SIGMOD '96), pp. 173-182.

[3]: E. Pitoura, B. Bhargava, "Data consistency in intermittently connected distributed systems," *IEEE Transactions on Knowledge and Data Engineering,* vol.11, no.6, pp.896-915, Nov/Dec 1999

[4]: O. Kremien, J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 03, no. 6, pp. 747-760, Nov., 1992

[5]: T.L. Casavant, J.G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141-154, Feb., 1988

[6]: N. G. Shivaratri, P. Krueger, M. Singhal, "Load distributing for locally distributed systems," *Computer*, vol.25, no.12, pp.33-44, Dec 1992

[7]: S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1327-1341, Sept., 1988

[8]: M. Livny, M. Melman, "Load balancing in homogeneous broadcast distributed systems," in *Proceedings of the Computer Network Performance Symposium* ACM, New York, NY, 47-55, 1982

[9]: D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A comparison of receiver-initiated and sender-initiated adaptive load sharing," *SIGMETRICS Performance Evaluation Review* 13, 2, Aug. 1985

[10]: M.J. Zaki, Wei Li, S. Parthasarathy, "Customized dynamic load balancing for a network of workstations," *Fifh IEEE International Symposium on High Performance Distributed Computing (HPDC-5 '96),* p. 282, 1996

[11]: S. Zhou, X. Zheng, J. Wang, and P. Delisle. "Utopia: a load sharing facility for large, heterogeneous distributed computer systems," *Software-Practice and Experience.* 23, 12, Dec. 1993, pp. 1305-1336.

[12]: M.-V. Mohamed-Salem, G. v. Bochmann, and J. W. Wong, "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, 2003.

[13]: S. P. Dandamudi, and K. C. Lo, "A Hierarchical Load Sharing Policy for Distributed Systems," in *Proceedings of the 5th international Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems,* MASCOTS. IEEE Computer Society, Washington, DC, 1997

[14]: J. Byers, J. Considine, and M. Mitzenmacher. "Simple Load Balancing for Distributed Hash Tables," In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, February 2003.

[15]: A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. "Load Balancing in Structured P2P Systems," In *Proceeding of 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003

[16]: B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies* , vol.4, no., pp. 2253-2262, vol.4, 7-11 March 2004

[17]: S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica, "Load balancing in dynamic structured peer-to-peer systems," *Performance EvaluationVolume 63, Issue 3, , P2P Computing Systems,* March 2006, Pages 217-240.

[18]: Y. Zhu, Y. Hu. "Efficient, proximity-aware load balancing for DHT-based P2P systems," *IEEE Transactions on Parallel and Distributed Systems,* vol.16, no.4, pp. 349-361, April 2005

[19]: A. R. Bharambe, M. Agrawal, and S. Seshan, "Mercury: supporting scalable multi-attribute range queries," In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communication.* SIGCOMM '04. ACM, New York, NY.

[20]: P. Ganesan, B. Mayank, and H. Garcia-Molina. "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems," in *VLDB, 2004*.

[21]: A. Corradi, L. Leonardi, F. Zambonelli, Diffusive Load-Balancing Policies for Dynamic Applications, *IEEE Concurrency, vol. 7, no. 1,* pp. 22-31, Jan.-Mar. 1999,

[22]: Cybenko, G. 1989. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.* 7, 2 (Oct. 1989), 279-301

[23]: Burkhard Monien and Robert Preis, Diffusion schemes for load balancing on heterogeneous networks, *Theory of Computing Systems*, *vol 35*, 2002.

[24]: T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 725-730, Jul., 1991

[25]: D. Ferrari, and S. Zhou, "A load index for dynamic load balancing," in *Proceedings of 1986 ACM Fall Joint Computer Conference* IEEE Computer Society Press, Los Alamitos, CA, pp. 684-690, 1986

[26]: R. Jain, "The Art of Computer Systems Performance Analysis"
1991, John Wiley & Sons, New York

[27]: Zhu, H., Yang, T., Zheng, Q., Watson, D., Ibarra, O. H., and Smith, T. 1998. Adaptive Load Sharing for Clustered Digital Library Servers. In *Proceedings of the 7th IEEE international Symposium on High Performance Distributed Computing* (July 28 - 31, 1998).

[28]: Bhaskaran Raman; Katz, R.H., "Load balancing and stability issues in algorithms for service composition," *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE* , vol.2, no., pp. 1477-1487 vol.2, 30 March-3 April 2003