Peer-to-peer platforms for high-quality web services: the case for load-balanced clustered peer-to-peer systems

Ying Qiao, Shah Asaduzzaman, Gregor v. Bochmann
School of Information Technology and Engineering (SITE)
University of Ottawa, Ottawa, Canada

Abstract

This chapter presents a clustered peer-to-peer system as a resource organization structure for web-service hosting platforms where service quality such as response time and service availability are provided with assurance. The peer-to-peer organization allows integration of autonomous resources into a single platform in a scalable manner. In clustered peer-to-peer systems, nodes are organized into clusters based on some proximity metric, and a distributed hash table overlay is created among the clusters. This organization enables lightweight techniques for load balancing among different clusters, which is found to be essential for providing response time guarantees. Service availability is provided by replicating a service instance in multiple nodes in a cluster. A decentralized load balancing technique called diffusive load balancing is presented in the context of clustered peer-to-peer systems and evaluated for effectiveness and performance.

C.1 Introduction

Web services are autonomous software systems that can be advertised, discovered and accessed through exchanging messages over the Internet using a set of standard protocols (e.g. SOAP, WSDL, UDDI). The standard protocols bring the interoperability among these autonomous software systems and allow creation of more complex and powerful applications through web service composition. This naturally allows distributed computation through execution of different components of an application hosted at autonomous Internet hosts. Computing resources, such as data, storage and CPU processing power of the hosts are thus shared by different users across the Internet.

When web services technology makes it easy to have distributed computing among computers and applications with different platforms, architectures, and programming languages, the role of distributed computing has expanded from assisting daily routines inside an enterprise to participating in the interactions among enterprises. However, the challenge to realize web service applications in a large scale still remains.

As web services are provided by software executed in computers on the Internet, situations like overloaded or failed servers or congested networks can largely affect the quality of web services, where

clients could experience unexpected delay and jitter on accessing web services. Consequently, the quality of a web service application can hardly be guaranteed with the unequal performance of its services. This directly challenges the further development of web service applications.

This situation becomes more and more severe with the large deployment of web service applications; it requires a scalable and efficient execution platform to provide high quality services. This chapter discusses the usability of a clustered peer-to-peer system with explicit load-balancing schemes as a platform for hosting web services. Web services hosted on server-pool based hosting platforms or data centers often suffer from overloaded servers and congested networks due to the static scale of the platform, which seriously degrades the perceived performance of the services. Highly scalable and adaptable peer-to-peer computing platforms may be desirable in these scenarios. The capability of peer-to-peer systems to efficiently organize large number of Internet-connected computers without any centralized controller makes it a good candidate as a platform for web services. Although there have been several works proposing the use of peer-to-peer techniques for web services discovery (Verma, 2005; Schmidt, 2004), attempts to exploit the scalability of peer-to-peer computing platforms for quality assured hosting web services are limited.

Peer-to-peer computing platforms are characterized by a huge collection of autonomous and inconsistent resources. A characteristic behavior of the resources in a peer-to-peer system is their intermittent arrival and departure, which is called *churn*. Explicit techniques are necessary to provide reliable and homogeneous services abstracting this behavior. Studies show that organizing these autonomous resources in clusters improves the reliability and the robustness of the platform (Locher, 2006). Clustered peer-to-peer systems such as eQuus improves the robustness of the system against churn by organizing the resources in clusters and replicating the resource-states within a cluster; applications built over these systems could have a more deterministic performance, e.g., CliqueStream (Asaduzzaman, 2008) for delivering life video streams over eQuus.

For the heterogeneity of the autonomous resources and the uneven distribution of service requests, some explicit load balancing mechanism is required to smooth out the performance inequalities towards some assured level of quality. A load balancing system can arrange resources according to the state of computing nodes and requests, so that the overall performance of the system can be improved. After using load balancing techniques, different web services are expected to have consistent performance characteristics across the platform.

Here we propose a hosting platform for web services using a load-balanced clustered peer-to-peer system. The platform performs load balancing at 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. Intra-cluster load balancing is achieved through commonly used techniques such as request routing, and the inter-cluster load balancing is achieved through movement of resources between different clusters. Organization of the physical resources in a clustered peer-to-peer overlay network facilitates such resource movement. A decentralized protocol, namely, *diffusive load balancing* is proposed for low overhead and effective balancing of load through resource movements between clusters (Qiao, 2009)

The chapter discusses the peer-to-peer hosting platform for web services from two perspectives: the feasibility and advantage of adopting a clustered P2P system as a platform for web service applications, and the effectiveness of the load balancing scheme used for inter-cluster load balancing. The feasibility is discussed in view of supporting quality of service with a clustered P2P system, focusing on service availability and response time as the two primary quality metrics. The effectiveness of the proposed diffusive load balancing scheme in achieving performance objectives is discussed based on simulation results.

C.2 Web Services and QoS

The deployment of Web Services and their use in a Service-Oriented Architecture (SOA) has many challenges. One of these challenges is the provision of certain levels of quality of service (QoS). Before going into discussion of how a clustered peer-to-peer platform achieves QoS for web services, it is useful to characterize QoS in the context of web services. It is also useful to discuss the commonly used techniques for QoS provisioning in existing server-pool based hosting platforms.

C.2.1 QoS Parameters and SLA

Different kinds of qualities can be considered in this context. The most important qualities are probably (a) response time (or latency), (b) availability and (c) cost. For each of these qualities different kinds of guarantees may be given. For the response time, for instance, one may refer to an average response time, possibly with additional information about percentiles. For the availability, one would typically indicate how small the probability is that, at any given time, the service would not be available. And for the cost, one has to distinguish different schemes, such as freely available (with or without subscription), pay by use, or pay by subscription.

The QoS of a given Web service is often documented in a so-called service-level agreement (SLA). The service provider organization may for instance publish this information as a statement about the service level that is intended to be provided to the public. In other situations, a SLA may be part of a contractual

agreement between the service provider and a user organization, where the SLA describes the level of response time and availability that the provider promises, probably in return for some specified costs. The SLA information is also useful for Web Service directories which provide information about available Web Services through interactive search or automatic queries. When a Web Service registers in such a directory, it may provide its SLA information; then the results of a search for a particular service function would provide a list of service instances with their QoS parameters. It would then be easy to find the service instance with the fastest response time, or the lowest cost. We note, however, that other factors may also be important for the selection of a service provider, such as quality of the information provided by the service, the reputation of the service provider, or the established business relationship that already exists.

As engineering tools for dealing with QoS, one needs means for determining the actual QoS provided and for managing the service system to assure that the intended QoS parameters are attained. Monitoring tools can be used to measure the actual QoS provided. Such tools could be used by the user and the service provider to check whether the SLA is satisfied. System management tools must be used by the service provider to manage the server hardware and software in order to optimize its performance and to assure the intended QoS parameters. This is particularly challenging in the case of services for which the demand is difficult to forecast. For certain applications, the load of service requests for various users may fluctuate over the period of a day or over weeks, and in other situations suddenly change due to some external situations. In these circumstances, the service provider should be able to adjust the hardware/software configuration providing the service in order to adapt quickly to the changing requirements.

C.2.2 QoS Provisioning Techniques

Basic approaches for obtaining high-performance, high-availability server configurations are well known. The configuration of a “server pool” consists of many identical or heterogeneous servers that provide the same service together with an entry point that distributes the incoming service requests to the different servers. Assuming that the hardware has been purchased previously or is available fast enough, it is relatively easy to introduce additional servers into the pool when the load gradually increases with time. By changing the number of servers in the pool, the average response time can be adjusted. The service availability largely increases because the failure of a single server has no impact on the availability of the service, as long as the other servers can take over the load.

In this server pool configuration, the entry point has the task of distributing the incoming service requests such that the load is balanced among all servers in the pool. We call this approach “load balancing

through request routing". In the case of identical servers, a simple round-robin algorithm may be adequate; however, for heterogeneous servers more sophisticated approaches may be preferable. For the traditional Web servers providing HTML pages, the problem of load balancing is described in (Bochmann, 2003).

In the server pool configuration considered above, there is essentially a single service that must be provided to a very large user community. The situation is different when a large number of different services are to be provided to a large user community; we call this situation "multi-service provisioning". In this case it is not feasible that each server holds the software and data for all these services. Instead, it is usually assumed that the different services are distributed over the set of available servers in such a way that the load of the different servers would be approximately balanced. This situation is considered, for instance, in (Reich, 2008; Mondejar, 2006). In this situation, one also needs some directory function which locates the server that provides the service requested by a user, which in the simplest case may be the Directory Name Service (DNS). For load balancing between the different servers, it is usually proposed that in the case of an overloaded server, one of the services provided by this server should be moved to another server that is less loaded. We call this approach "load balancing through service movement". In large server systems, the question how to find a server with little load is not straightforward, as discussed in Section C.5.

For providing high availability in the case of "multi-service provisioning", one may also duplicate each service over two or more servers. In the case that some of the services have a very large load of requests, it is also conceivable to use service duplication for balancing the load, like in the case of server pools. This may lead to a configuration of a set of "clustered servers", where the services are distributed over the clusters and each cluster of servers is responsible for a certain set of services. The minimum size of a cluster is then determined by the availability requirements of the supported services and the actual size of the cluster may be much bigger if the load of the supported services requires a large number of servers.

For managing the response time in the case of multi-service provisioning with clustered servers, one needs load balancing at two levels. Within each cluster, the "load balancing through request routing" approach may be used, as in server pools. For the balancing of the load among the different clusters, two approaches are feasible. One possibility is the "load balancing through service movement", as described above. Another possibility, called "load balancing through resource movement" consists of moving a server from an under-utilized cluster to an over-loaded cluster. This approach is further discussed in Section C.5.4.

C.3 Peer-to-peer systems and their variants

C.3.1 Peer-to-peer systems

A Peer-to-Peer (P2P) system is a form of distributed computing system with autonomous computer nodes located at distributed locations and connected to the Internet. The computers, called *peers* in P2P jargon, are usually end-user personal computers, but sometimes computing servers from service providers. The characteristic feature of a P2P system is its decentralized nature of management responsibilities. Computers or peers in a P2P system provide services to each other. There is no distinguished difference concerning the responsibilities of these peers. A peer can take the role of both the client and the server of a distributed system in the sense of a client-server architecture. Originally, creation of peer-to-peer systems was motivated by the application of decentralized file sharing. Gradually, as the versatility of the peer-to-peer organization of computers was more deeply perceived, applications of peer-to-peer system have included group communication, multimedia streaming, large scale data storage, and sharing of computational resources.

A P2P system can be decomposed into a layered architecture with a P2P application layer on the top of an overlay network layer. These two layers work on top of the IP network layer of Internet that provides the end-to-end physical connectivity. In this sense, this is a three layer system, where the overlay routing functionality at the middle is often term as *middleware*. At the P2P application layer, each peer will perform functions specified by the application, i.e. displaying file directory for a file storage application, or playing movie for a multimedia application. The overlay network layer provides a network connecting all peers and a searching mechanism for the application layer to locate objects among peers. It maintains a virtual network topology using physical connectivity of the Internet. Being able to routing lookup messages in this overlay network, P2P systems dynamically search and locate objects without centralized directory services. The ability of peer-to-peer systems to self-organize a large number of computers and to locate objects among these computers across the Internet without any central authority are useful properties for service hosting platform. The popularity of peer-to-peer applications such as Skype, PPLive, and eDunkey, indicates that its scalability, economy of cost corresponding to its large scale, and its capability of providing services on a highly dynamic network are favored by end-users. However, one big challenge that a peer-to-peer system faces is to effectively guarantee the reliability and performance of the services it provides.

C.3.2 Structured and unstructured P2P systems

Peer-to-peer systems are broadly classified into structured and unstructured peer-to-peer systems. In the structured systems, the peers choose the interconnection neighborhood following a certain pattern which can later be used to facilitate efficient routing of messages such as those for object storage and lookup. In

unstructured peer-to-peer systems, peer interconnection does not follow any pattern and thus lacks the efficiency of search in a predefined pattern. Unstructured systems, however, avoid the overhead of maintaining a predefined structure in the interconnection. A structured P2P system in effect, implements a distributed hash table (DHT) in its substrate, in which each peer has a unique identifier. Data objects are placed deterministically at the peers with identifiers corresponding to the data object's unique key. The interconnection topology of a particular pattern is maintained, such that a request for a particular object can be routed to its location solely based on local knowledge at each peer. There are well studied variants of such structured peer-to-peer systems, such as Pastry (Rowstron, 2001), Chord (Stoica, 2001), Kademlia (Maymounkov, 2002), CAN (Ratnasamy, 2001) and Viceroy (Malkhi, 2002), which mainly vary in their interconnection patterns. A popular and well studied example of unstructured peer-to-peer system is Gnutella (Oram, 2000). A survey of different peer-to-peer systems can be found in (Bochmann, 2007).

C.3.3 Clustered peer-to-peer systems

In some recently proposed structured peer-to-peer systems, while creating the interconnection topology with a particular pattern, peers organize themselves into groups or clusters. Such clustering actually provides an in-built membership management service in addition to the routing service provided by the overlay structure. This in-built membership management can be exploited for on-demand provisioning of resources in a service hosting platform.

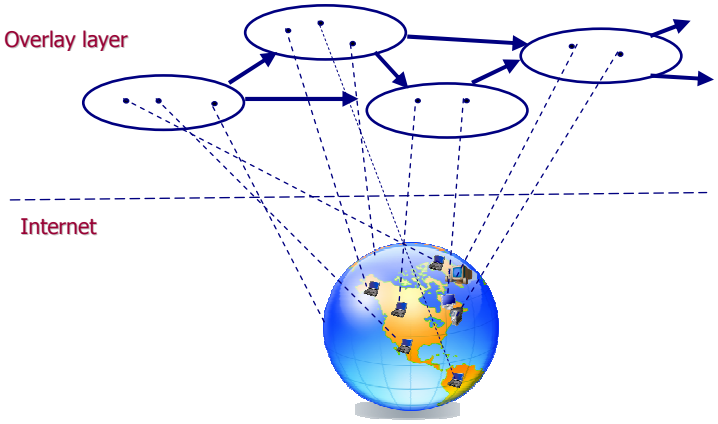

Figure 1: A clustered peer-to-peer system organizes the nodes in a number of clusters. Nodes in a single cluster may be chosen based on any proximity criterion and may come from different geographic origins.

In the clustered peer-to-peer system eQuus (Locher, 2006), peers that are close-by, based on some proximity metric, are grouped in a single cluster. The proximity metric needs to be such that peers can be placed in a low-dimensional Cartesian space based on their distances. For example network latency, geographic distance, capacity in terms of different resources like CPU and memory, or the type of the service the peer provides, can be used as the proximity metric. Peers in a single cluster are assigned a single identifier and the clusters are organized in a distributed hash table like interconnection. Thus, all peers in a single cluster share the same cluster-level neighborhood table. Also all the peers in the same cluster know each other. This allows any peer in a cluster to fail or depart without much consequence to the rest of the system. Such built-in membership management by organizing the peers in clusters allows easy service replication and request routing in a service hosting platform where a single cluster is designated for a single type of service. Later, we show in Section C.5 that balancing of load can be performed in a decentralized manner by moving resources among the clusters to adapt to the varying load on different types of services. This is the principle called “load balancing through resource movement” in Section C.2.2.

C.4 Clustered peer-to-peer system for high quality service hosting

Clusters of computers or server pools are being used for hosting web services for several years. A large number of commodity computers with storage and processing power are aggregated in a data center and interconnected with a high-speed local area network. A wide variety of web services provided by many different providers may be hosted in a single data center platform. Usually, allocation of these server resources to different services are controlled by centralized task schedulers or master controllers. Here we discuss how the resources of a hosting platform can be managed in a decentralized way using the features of a peer-to-peer system organization.

Clustering or aggregating resources in a single unit are done in various forms. In tightly coupled clustered systems, such as super-computers, multiple CPUs, memories, and storages are combined in one unit with high-speed interconnections in a fault-tolerant architecture to provide high reliability and performance for mission-critical applications such as scientific computing, air traffic control or financial forecasting. However, the high cost of building and managing these special purpose systems limit their usage in a small scope. For this reason, loosely coupled clustered systems, like pools of commodity computers interconnected in local-area networks, have gained popularity as platforms for general-purpose web service hosting.

For the correctness and liveness of the computations in the loosely-coupled clustered systems, studies have shown several uses of membership management protocols. A membership management protocol

organizes all the nodes in the system in a number of groups, and for each group provides a correct list of group members to each member of the group. First of all, having a membership service allows disseminating the failure status of individual nodes easily among the members of the group. This helps all the nodes to easily find a live node to instantiate a service. Secondly, when replication of data objects is necessary for reliability purpose, it is useful to replicate the objects among the members of a group. Having a managed group helps to easily update the replicas with consistency. Third, having a managed group membership helps allocation of necessary resources among different services. It is usually natural to assign different services to resource-clusters. Having the aggregate status of the clusters helps adaptive re-allocation of the resources as necessary. This also helps prioritizing resources among different classes of services simply by allocating necessary resources to corresponding clusters. Fourthly, the built-in membership management helps to construct an efficient inter-group load balancing mechanism.

Peer-to-peer systems, as discussed in Section C.3.2, provide a key-based message routing service in a large collection of widely distributed computers. A key advantage of peer-to-peer systems is that this routing function and the maintenance of the routing tables in peers are performed in a completely decentralized manner. Such decentralized key-based routing service is used in many computing platforms for discovery of resources. Clustered peer-to-peer systems provide an additional service of membership management besides the usual key-based routing service. That is why we argue that clustered peer-to-peer systems are more suitable for a multi-service hosting platform, being better equipped for fault-detection, status dissemination, replica consistency management, resource prioritization and load balancing. In comparison, the other non-clustered peer-to-peer systems leave these additional mechanisms to be dealt with by individual applications. For example, in Pastry or Chord, when used as a platform for a distributed file system, consistency among replicas of an object is maintained by the file system application itself, by maintaining and probing the locations of the replicas.

Having a modular membership management also makes a clustered peer-to-peer system more scalable because it can avoid redundant implementation of membership management techniques needed for the different purposes. Also, by allowing better and easier implementation of prioritization in resource allocation among different types or groups of services, and system-wide balancing of load by re-assigning resources among different clusters, a hosting platform based on clustered peer-to-peer systems yields better availability and response time characteristics for the supported services.

C.5 Load balancing techniques

Load balancing techniques can be applied to any system with multiple computing nodes, for instance, multi-processor, parallel, or distributed computing systems. These multiple computing nodes are

organized as clusters. 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 for the purpose of load balancing. In Section C.2.3, we introduced three different approaches of load balancing – request routing, service movement and resource movement in the context of improving service availability and response time in multi-service hosting platforms. Here we discuss the design and evaluation of a diffusive load balancing protocol for the clustered peer-to-peer systems using resource movement, after a brief review of load balancing techniques commonly applied in other distributed computing platforms.

C.5.1 Load balancing in distributed systems

A distributed system moves workload from heavily loaded nodes to lightly loaded nodes according to a predefined load balancing scheme to improve its overall performance (Casavant, 1988). A load balancing scheme is a combination of policies that define when and where to initiate a load movement, how to monitor and collect the system-wide load information and how to select which workload to move and where (Leinberger, 2000; Cardellini, 2003). The schemes can be broadly classified into two categories, static and adaptive. Static schemes works with a predefined set of policy parameters decided based on the average load of the system (Wang, 1984), while adaptive schemes need to monitor the system status and defines the policy parameters based on the observed status (Kunz, 1991). Architecturally, these load balancing schemes may be implemented in a centralized or in a decentralized manner. In a decentralized scheme, all nodes can locally decide to start transferring a load either into it or out from it. Different decentralized schemes vary primarily in terms of how status of the system is aggregated and disseminated. Each node may periodically broadcast its state, or the advertisement can be limited to the times when the node moves from one discrete state to anther (e.g. becoming available from busy) (Livny, 1982) . Instead of broadcasting the state, the node that is willing to trigger some load balancing action may probe a selected subset of other nodes for their status. This probing can be sender initiated (Zhou, 1988) as well as receiver initiated (Livny, 1982).

C.5.2 Load balancing in peer-to-peer systems

Load balancing techniques in peer-to-peer systems, in general, face challenges due to the characteristics of these systems. First of all, the sheer size of a peer-to-peer system indicates that a load balancing technique applied to it must be scalable. Second, all nodes of a peer-to-peer system are not replicas of each other and requests cannot be routed to and executed in any of the nodes. Alternatively, P2P systems place or re-place shared objects optimally among nodes, and overlay routing tables would redirect

requests for these shared objects to the right nodes; as a result, the load of the P2P system can be balanced.

In all the peer-to-peer systems, an implicit load distribution is achieved through random placement of the nodes in the overlay structure through random assignment of node identifiers. However, they lack the capability to adjust the placement of the objects, or reroute the requests, based on changing load in different parts of the system. Some explicit and adaptive load balancing techniques are applied in many systems. 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 (Byers, 2003) or through dynamic load balancing. In the latter case, objects can be stored in virtual servers and moved from nodes to nodes. (Rao, 2003; Godfrey, 2004; Surana, 2006) 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 some duration, it will leave the directory and register in another one in turn. (Zhu, 2005) proposed a *k-ary* tree architecture for load balancing, where the inner nodes and the 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.

Using the principle of load balancing through resource movement, 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 (Bharambe, 2004). 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. Using an estimation based on sampling, a node is able to detect a lightly loaded range, and request a node from there to move to its neighborhood if it is overloaded.

(Ganesan, 2004) proposed a load balancing mechanism that combines both object placement and node placement in a P2P system. Nodes are connected through a linear chain, and each node balances its load with its two consecutive neighbours. If a node has already balanced its load with its neighbours and it is still overloaded, it will select a lightly loaded node in the system to hand over some of its load. Before this movement, the lightly loaded node will shed all of its current load to its own neighbors. 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 linear chain; this requires frequent updates of the skip list when the load situation changes.

One common aspect in the dynamic load balancing techniques for P2P systems is that they can achieve global load balancing through local balancing procedures. Local balancing means that balancing occurs among nodes in a certain scope, e.g., the two immediate neighborhoods of a node in a linked list, or some subset of nodes in the system, e.g., some nodes randomly selected. Each decision component that runs a load balancing procedure has a scope within which it searches targets: overloaded nodes or under-loaded nodes (senders and receivers) for possible load transfers. The scopes of different decision components may overlap; if this overlapping leads to global connectivity among all local scopes, the system has the property that it will be balanced when all local scopes are balanced.

Some load balancing techniques for P2P systems build extra connections between the nodes on top of the overlay network structure. For instance, a *k-ary* tree requires *(n-1)* connections for aggregating and disseminating load statuses, and a skip list uses a total of $(3n-2-\log_2 n)$ connections for ordering nodes according to their load statuses. These connections are maintained during the life time of the load balancing procedure. When the overlay network experiences churn, these connections are highly dynamic as well.

We propose a diffusive load balancing scheme for structured clustered P2P system using a DHT, where each cluster works as a decision component running a procedure to locally balance the loads among its neighboring clusters. The load balancing among the nodes of a given cluster is assumed to be performed by some other intra-cluster balancing mechanism. With both inter-cluster and intra-cluster load balancing, the system achieves a global balance. All the messages, including load reports from neighbors and the dissemination of load transfer decisions, are transmitted through existing inter-node connections. Load transfers between clusters are realized through moving a node from an under-loaded cluster to an over-loaded cluster.

C.5.3 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”. A diffusive load balancing policy is a policy having three aspects (Corradi, 1999): 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, all components of the 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 domain; also, they are efficient on achieving global balancing with a small amount of message overheads.

Diffusive load balancing policies can be classified according to two aspects: decision making and load migration. While making a decision, the component 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 on senders and receivers in its domain and initiate load migrations among them (termed as a directory-initiated policy). Each component is only allowed to participate in one load migration action at a time, either sending or receiving, which prevents it from receiving or shedding loads multiple times at the same time.

C.5.4 Diffusive load balancing for clustered peer-to-peer system

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 already been implemented inside each cluster.

We adopt resource movements instead of service movement for load balancing. Resource movement is performed by re-allocating resources from one service to the other, which means re-assignment of computing nodes in a loosely coupled networked computing platform. For peer-to peer systems, node movement involves reconfiguring the neighborhood table of the concerned nodes, which is very simple compared to moving the service across long-distance network. This also helps avoiding the overhead of maintaining data consistency among a large number of nodes.

C.5.4.1 Choice of the load index: available capacity

A dynamic load balancing scheme identifies the system status according to a load index for each node. A load index should correctly reflect the amount of load at a node, and from this index, the performance of a node could be evaluated. CPU queue length is generally preferred as a load index (Ferrari, 1986; Zhou 1988; Kunz, 1991) because it has a strong correlation with the mean response time of tasks at the node. Other load indexes include utilization, request-response time and available capacity.

We adopt the average of the available capacities of the nodes in a cluster as the load index for the cluster, as proposed in (Zhu, 1998) and (Raman, 2003). Using a M/M/1 queuing model, it can be shown that the

average response time at a node is the inverse of the available capacity of the node; this means that, when two nodes have the same available capacity, even if they have different maximum capacities, their mean response times, for a given request, will be the same (Qiao, 2009). Under the assumption that the load among the nodes in each cluster have been balanced by using some intra-cluster load balancing procedure, and the load among different clusters has been balanced by the here described procedure, all nodes in the system will have an available capacity close to the overall mean. Hence, the mean response times of all nodes are close to an average value.

C.5.4.2 Inter-cluster diffusive load balancing algorithm

Using the average 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 overlay neighbors, and makes decisions concerning possible load movements with these neighbors. We use the traditional meaning of sender and receiver here: a sender is a cluster that transfers its load out, and a receiver cluster transfers load in. Because node movement is used here instead of load movement, nodes are in fact transferred from the load-receiver cluster to the load-sender cluster.

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

- *LB triggering*: the execution of LB is triggered by a timeout event after a predefined amount of time from the last LB execution, or a state change event when the cluster becomes either receiver or sender.
- *Load determination*: First, the cluster determines its own load status as well as the load status of its neighborhood through sending probing messages to its neighbors, and waits for responses from them; a probed cluster responds with its load index.
- *Decision:* Dynamic thresholds are used to determine whether a cluster is considered overloaded or under-loaded. First, the load average is calculated for all the clusters in the neighborhood. Then the upper and lower load thresholds are calculated by the formula: threshold = average_neighborhood_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 (upper) 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 neighborhood.
- *Receiver-initiated:* If the cluster is a receiver, then it tries to identify a corresponding sender in its neighborhood.

- *Load transfer*: Note that nodes are moved from a load-receiver cluster to a load-sender cluster to bring the balance. 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 load average, 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.

  The required capacity for an overloaded cluster to reach the average load status of the neighborhood is the difference between the current load index of the cluster to the average load index of the neighborhood multiplied by the number of nodes in the cluster. As our algorithm only moves a single node at a time, the required capacity of that node could be calculated as:

$$\text{required\_capacity} = \text{average\_neighbourhood\_index} * (\text{current\_size} + 1) - \text{current\_index} * \text{current\_size}$$

C.5.4.3 The inter-cluster diffusive load balancing procedure

A procedure is designed to realize the above algorithm. The state diagram of the procedure is shown in Figure 2, where each state corresponds to a phase of the algorithm.

Each cluster selects a leader among its nodes; this nodes acts as coordinator for the load balancing procedure. As in diffusive load balancing, a cluster would participate in the neighborhoods of several other clusters. When the coordinator of a cluster fails before finishing a round, the round will be discarded. Another node will be selected as a coordinator and will start a new round. Those coordinators also take responsibility for monitoring the load status of their own cluster and respond to the probing messages and load transferring instructions from other clusters.

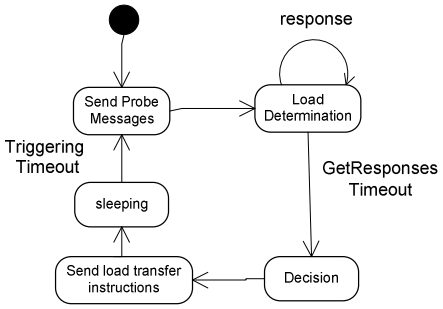

Figure 2. The state diagram of the load balancing procedure

As the load balancing procedures coordinated by different clusters are not synchronized, the load balancing protocol uses asynchronous messaging. When a coordinator is in the “Load Determination” state, it waits for responses from the probed clusters. Since the existing inter-cluster connections of the clustered P2P system are used to transfer the messages for load balancing, the health of the connections can be monitored by the functions in the P2P overlay, and the time for transferring messages can be estimated.

However, a cluster can participate only in one node movement at a time. If the cluster is participating in a node movement, the coordinator will refuse any additional request for node movements.

C.5.5 Evaluation of diffusive load balancing

In order to evaluate the effectiveness and performance of the diffusive load balancing in the context of clustered peer-to-peer systems, we performed several simulation studies. We compared the effectiveness and performance of three different location policies – directory initiated (DI), receiver initiated (RI) and sender initiated (SI), as introduced before. We also compared them with an idealized scheme, called central directory (CD), 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 as the other schemes.

C.5.5.1 Simulation setup

We have implemented the load balancing procedure with the three different location policies in a simulation of a clustered DHT overlay network (a modified version of eQuus). eQuus [1] is a structured P2P overlay which was proposed to improve the reliability and robustness of a DHT overlay networks. In eQuus, the nodes are organized into clusters according to a proximity metrics, and the clusters are connected using a DHT mechanism: each cluster is identified by a unique ID, and the DHT routing tables are constructed based on these IDs. The system has inter-cluster connections similar to Pastry using prefix matching in its routing algorithm; therefore the number of steps in a lookup procedure is bound by

O(logN), where *N* is the number of clusters. There are no super nodes; each entry in a routing table contains up to *k* nodes belonging to the same cluster; a node can select a node from these *k* neighbors to forward a lookup message.

In addition to the operations in regular DHT systems, an eQuus system manages the size of its clusters by keeping their sizes in a fixed range by performing splitting and merging operations of clusters; correspondingly, the system is able to adjust the routing tables. Within an eQuus, the size of a cluster may change through churn (nodes leaving or joining the system). Only when the size of a cluster violates the given size limits, are the cluster splitting or merging operations invoked. Therefore, the number of messages for updating the routing tables is reduced, as compared to P2P systems without clustering.

For our simulation, we constructed a clustered peer-to-peer system consisting of 10,000 nodes of equal capacity. In our simulation, the proximity criterion for cluster formation of eQuus is relaxed: a node may join any cluster, so that, the system could allow the node movements between any pair of clusters. The average size of a cluster was 8 (varying between 4 and 16). We evaluated inter-cluster load balancing only and it is assumed that the load among nodes inside a cluster are balanced by some other mechanism. The simulation starts from an initial state where each cluster is assigned a workload that widely varies among clusters. The *bound* parameter that defines whether a cluster is overloaded or under-loaded is set to +/-20%. The simulated load balancing algorithm keeps moving nodes between different clusters in rounds and the simulation stops when there is no node movement in the last round.

When a cluster is probed during a node movement, the coordinator returns a load index which corresponds to the expected load situation after the node movement; the coordinator is able to estimate this value based on the last recorded load index and the capacity lost or gained from the node movement. However, the coordinator would discard another load transfer request before it finishes the current one. For this reason, the delay of node movements is not considered in our simulation.

C.5.5.2 Effectiveness of the different load balancing schemes

To evaluate the effectiveness of different schemes we measured the variation of loads among different clusters at the end of the simulation (standard deviation and *delta* = maximum - average). The capacity of the nodes is assumed to be homogeneous across the system with each node being capable of executing a maximum of 10 units of load. At the initial state, each cluster is assigned a random workload uniformly distributed between 0 and the maximum capacity of the cluster. The histogram of the initial workload across different clusters is shown in Figure-3a.

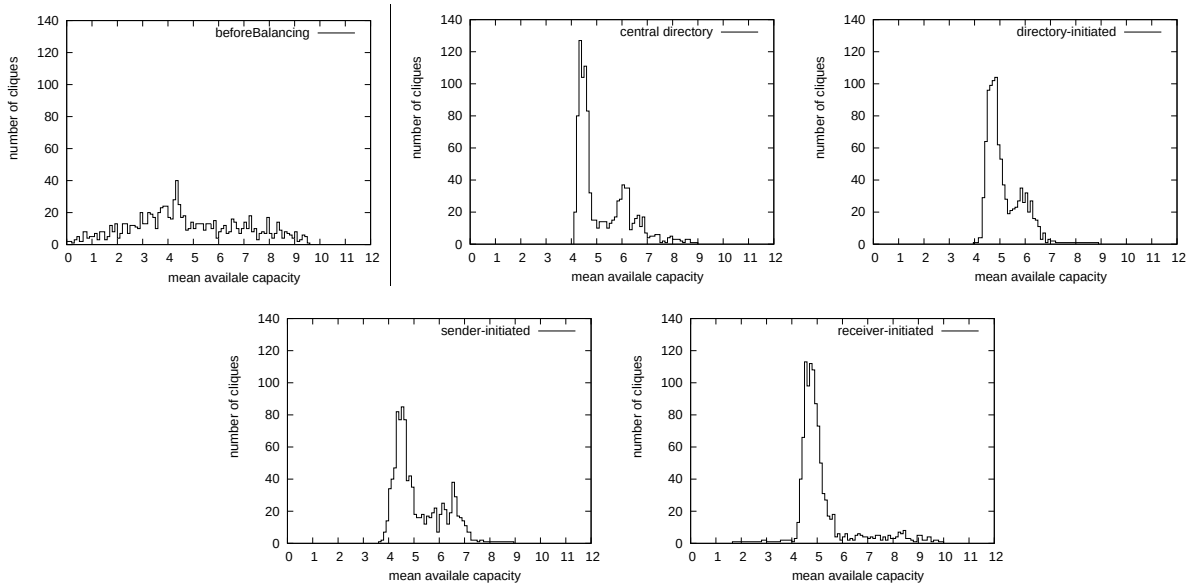

Figure 3: Distribution of load in different clusters: (a) before balancing, (b) through (e) after balancing

Except for the receiver-initiated scheme, the three other schemes balance the load tightly around the average load of the clusters. The directory-initiated scheme reaches a load distribution, where the load index has a high maximum 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, there are still some overloaded clusters remaining when the simulation stops. This is because a cluster makes a 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. The load distributions after load balancing for all the schemes are shown in Figure 3(b-e).

C.5.5.3 Performance of the load balancing algorithm

To evaluate the performance, we measured how many rounds each variant of the load balancing scheme would take to converge. We also measured the number of node movements that occurred in each case (Table 1). The simulation setup is the same as the one used for evaluating a homogenous system. We observe that the central directory scheme reaches the balanced state with the smallest number of node movements. 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 receiver-initiated scheme has slowest convergence with the most number of rounds.

To visualize how the system gradually progresses towards a balanced load through execution of the algorithm with different policies, we display in Figure 4 how different measures change with the

progression of balancing rounds. During each round, each cluster has the opportunity of executing a load balancing procedure once. Figure-4(a) shows that most of the node movement occurs during the first round in all schemes. 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%. This faster node movement corresponds to the faster convergence of the directory–initiated scheme as shown in Figure 4(b).

Table 1. Comparison of load balancing results

	CD	DI	SI	RI
Std. dev.	1.033	0.709	0.915	1.098
Delta. (%)	19.63	19.28	25.14	62.3
rounds	4.3	1.1	3.1	4.9
Node mv.	1680	1942	1695	2013
splits	180	219	200	210
merges	81	126	123	81

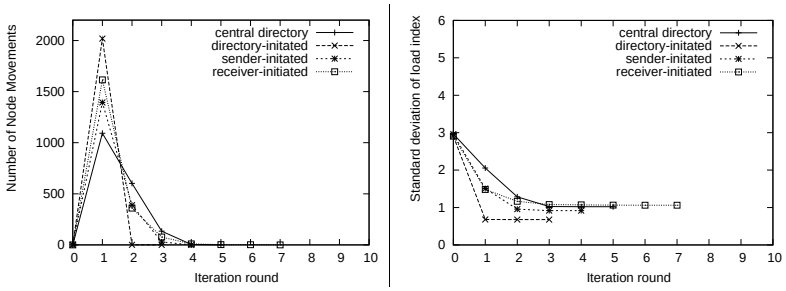

Figure 4: Progress of load balancing

C.5.5.4 Load balancing with heterogeneous node capacities

To exploit the fact that node capacities are heterogeneous in a peer-to-peer system, we modified the *Selection* policies of the load balancing schemes as follows. When a scheme selects a node for movement, it considers the capacity required for the load-sender cluster to reach the mean load index, and picks from the load-receiver cluster a node with an available capacity closest to the requirement. We call this *capacity-based node selection*.

In our simulation, we assigned node capacities from the range [100, 5000] with a Pareto distribution with shape parameter set to 2, and scale parameter set to 100. The other parameter remains the same. Table 2 summarizes the measured performance results for the four different location policies, with the capacity-based node selection policy and random node selection policy. We observe that the relative differences

among the location policies observed in the homogeneous system remain present in the heterogeneous system. The number of node movements is reduced when capacity consideration is applied, compared to random node selection. For instance, in the directory-initiated scheme, the movements are reduced by almost 20%. This indicates that selecting a node with its maximum capacity matching the required capacity is better than random selection.

Table 2. Comparison of load balancing results with random and capacity-based policy

	random		Capacity	
	CD	DI	CD	DI
Std. dev.	5.98	4.5	5.5	4.37
Delta. (%)	19.45	20.56	19.92	19.5
rounds	4.8	1.6	4.4	1.6
Node mv.	1722	2011	1204	1645
split	182	213	128	181
merge	90	119	28.4	83

CD: central directory, DI: directory-initiated

C.6 Conclusion

Server clusters have been already adopted as a reliable and high-performance hosting platform for websites and web-service based applications. Their centralized resource-management structure, however, does not allow resource sharing among autonomous resource providers. In this chapter, we give arguments for a web service hosting platform with clustered peer-to-peer organization of resource nodes. This platform is able to host multiple web services from several providers targeted for a large user-base through sharing of autonomous resources. It provides quality assurance for services in terms of service availability and response time. In other words, it inherits the characters of reliability and performance from server cluster systems and the scalability from peer-to-peer systems.

For providing assurance of service quality in terms of response time and service availability, it is necessary to distribute and balance the workload over different resources. In the clustered peer-to-peer organization, different services may be assigned the resources of different clusters. The clustered organization allows load balancing to be performed relatively easily by exchanging the node-positions in the cluster structure. Thus, necessary resources can be allocated to an overloaded cluster to balance the load.

Assuming that load among different nodes inside a cluster are balanced using some known technique, we proposed a diffusive load balancing algorithm to conduct the inter-cluster load balancing by moving resources between clusters. As the available capacities of clusters are equalized in this way, the response times provided by the different clusters for the different services approach an overall mean value. The load balancing algorithm works in decentralized manner based on the local knowledge of the load of the clusters in the neighbourhood within the peer-to-peer overlay.

Through simulation study, we measured the effectiveness and performance of the load balancing algorithm for three different decision schemes, namely, directory-initiated, sender-initiated and receiver-initiated. The directory-initiated scheme converges faster and results in a smaller variance of load among the clusters, compared to the other two schemes. However, it results in a larger number of node-movements. In general, the fast convergence of the loads of clusters around the mean value, demonstrates the effectiveness of the diffusive load balancing algorithm for the clustered peer-to-peer service hosting platform.

However, several problems regarding this platform remain to be solved. In our discussion we explained the technique to balance the load equally among clusters. This would result in a single level of quality for all services. However, different services may require different response times, and different users may ask for the same service with different response times. The question of how different classes of services with different levels of quality could be provided in the proposed platform, needs further investigation.

The QoS of web services is often described by multi-dimensional attributes. In addition to availability and performance, which we discussed here, reputation, price, locality, and other aspects need to be considered as well. It is possible to create different quality profiles for different classes of services based on different value-ranges of these attributes. Cluster of resources may be created to support each of these classes. However, how to maintain the quality profile of each of these clusters, remains a problem to be solved.

There are also some implementation hurdles that need to be resolved. For example, when resources are moved between clusters, services may need to be migrated between nodes inside a cluster. Techniques are needed for encapsulating a stateful service in service containers to facilitate live migrations.

In summary, with a proper implementation, the clustered P2P organization is an efficient way to manage resources in a large-scale web service hosting platform, with assurance for various attributes of service quality.

References

Asaduzzaman, S., Qiao, Y. & Bochmann, G. (2008). CliqueStream: An Efficient and Fault-resilient Live Streaming Network on a Clustered Peer-to-peer Overlay. In *Proceeding of 8th International Conference on Peer-to-Peer Computing*, Aachen, Germany, 2008.

Bharambe, A. R., Agrawal, M. & Seshan, S. (2004). 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.

Bochmann, G., Wong, J. W., Lau, T. C., Bourne, D., Evans, D., Kerhervé, B., Salem M. V. & Ye, H. (2003). Scalability of Web-based electronic commerce systems, IEEE Communications Magazine, July 2003, Vol. 41, No. 7, pp. 110-115.

Bochmann, G. & Jourdan, G. V. (2007). An overview of content distribution and content access in peer-to-peer systems (invited paper). in *Proceeding of NOTERE Conference*, Marakech (Maroco), June 2007.

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

Cardellini, V., Colajanni, M. & Yu, P. S. (2003). Request redirection algorithms for distributed web systems. *IEEE Transaction on Parallel Distributed System*, vol. 14, no. 4, pp. 355–368, 2003.

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

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

Ferrari, D. & Zhou, S. (1986). 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

Ganesan, P., Mayank, B. & Garcia-Molina, H. (2004). Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. in *VLDB, 2004*.

Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R. & Stoica, I. (2004). 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

Kunz, T. (1991). 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

Leinberger, W., Karypis, G., Kumar, V. & Biswas, R. (2000). Load balancing across near-homogeneous multi-resource servers. *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th* , vol., no., pp.60-71, 2000

Livny, M., & Melman, M. (1982). Load balancing in homogeneous broadcast distributed systems. in *Proceedings of the Computer Network Performance Symposium* ACM, New York, NY, 47-55, 1982

Locher T., Schmid, S. & Wattenhofer, R. (2006). eQuus: A Provably Robust and Locality-Aware Peer-to-Peer System. *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06),* 2006

Malkhi, D., Naor, M. & Ratajczak, D. (2002). Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing* (Monterey, California, July 21 - 24, 2002). PODC '02. ACM, New York, NY, 183-192.

Maymounkov, P. & Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers From the First international Workshop on Peer-To-Peer Systems* (March 07 - 08, 2002). P. Druschel, M. F. Kaashoek, and A. I. Rowstron, Eds. Lecture Notes In Computer Science, vol. 2429. Springer-Verlag, London, 53-65.

Mondejar, R., Garcia, P., Pairot, C. & Gomez Skarmeta, A. F. (2006). Enabling Wide-Area Service Oriented Architecture through the p2pWeb Model. In *Proceedings of the 15th IEEE international Workshops on Enabling Technologies: infrastructure For Collaborative Enterprises* (June 26 - 28, 2006). WETICE. IEEE Computer Society, Washington, DC.

Oram, A. (2000). Gnutella and Freenet Represent True Technological Innovation. Whitpaper, 2000.

Qiao, Y. & v. Bochmann, G. (2009). Applying a diffusive load balancing in a clustered P2P system. In 9th international conference on New Technologies of Distributed Systems (NOTERE), Montreal, Canada, 2009, (submitted for review).

Raman, B. & Katz, R.H. (2003). 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

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

Ratnasamy, S., Francis, P., Handley, M., Karp, R. & Shenker, S. (2001). A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications*. SIGCOMM '01. ACM, New York, pp. 161-172.

Reich, C., Bubendorfer, K., and Buyya, R. (2008). An Autonomic Peer-to-Peer Architecture for Hosting Stateful Web Services. In *Proceedings of the 2008 Eighth IEEE international Symposium on Cluster Computing and the Grid* (May 19 - 22, 2008). CCGRID. IEEE Computer Society, Washington, DC, 250-257.

Rowstron, A. & Druschel, P. (2001). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. in *Middleware*, 2001, LNCS 2218, pp.329-350, 2001.

Schmidt, C. & Parashar, M. (2004). A Peer-to-Peer Approach to Web Service Discovery. *World Wide Web* 7, 2 (Jun. 2004), 211-229.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications* (San Diego, California, United States). SIGCOMM '01. ACM, New York, NY, 149-160.

Surana, S., Godfrey, B., Lakshminarayanan, K., Karp, R. & Stoica, I. (2006). Load balancing in dynamic structured peer-to-peer systems. *Performance EvaluationVolume 63, Issue 3, , P2P Computing Systems,* March 2006, Pages 217-240.

Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S. & Miller, J. (2005). METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Inf. Technol. and Management* 6, 1 (Jan. 2005), 17-39

Wang, Y. T. & Morris, R. J. T. (1985). Load Sharing in Distributed Systems. *IEEE Transactions on Computers*, vol.C-34, no.3, pp.204-217, March 1985

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

Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J., & Chang, H. (2004). QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.* 30, 5 (May. 2004), 311-327.

Zhu, H., Yang, T., Zheng, Q., Watson, D., Ibarra, O. H. & 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).

Zhu, Y. & Hu, Y. (2005). 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