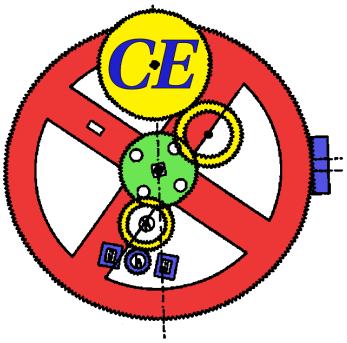


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2008

MSc THESIS

First Approach for a Decentralized Resource Allocation in Ad Hoc Grids

Shervin Haji Amini

CE-MS-2008-09

Abstract

As more people and organizations use computers, the demands on computer systems have changed. Therefore, with increasing the number of participants, the scalability has become an important issue for managing resource allocation. A centralized system relies on one computer node that processes the whole application. In such systems, a small number of nodes provide the resources that can only serve a limited number of clients. On one hand, if the centralized system becomes large enough, load balancing will create too much overhead. On the other hand, limitation on the network bandwidth increases the network bottleneck which affects the system performance. Decenralizing control, distributing processing loads and network bandwidth among all participating nodes will reduce the overhead in very large systems. In this light, Peer-To-Peer(P2P) systems offer an alternative solution to old server-based approach. Generally, a P2P system must have three essential properties. First, the system needs to be scalable. That is, it does not rely on single points of failures and bottlenecks. Second, the P2P system needs to be self-organized when peers join and leave the system arbitrarily. Third, the system needs to be fault-tolerant. In other words, the system must be robust when subjected to faults. In this thesis, we propose a dynamic decentralized P2P resource allocation method having as many resource allocators as required given the number of nodes, tasks and resources. In spite of the capability of the proposed resource allocation scheme for implementing on any structured overlay system, we implemented our application on *Pastry* as the underlying structured overlay system.
We investigated the matchmaking process in the fully populated system when all resource allocators are active as well as the non-fully populated system when only some resource allocators perform resource allocation process. As we wanted to have a self-organized system in terms of resource allocators, we defined a mechanism that dynamically segment/desegment the ad hoc grid by promoting/demoting resource allocator(s) according to the workload of the existing resource allocator(s). This mechanism enables the ad hoc grid to adapt itself to the changing environment from a centralized to a decentralized form and back to the centralized form.

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science

First Approach for a Decentralized Resource Allocation in Ad Hoc Grids

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Shervin Haji Amini
born in TEHRAN, IRAN

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

First Approach for a Decentralized Resource Allocation in Ad Hoc Grids

by Shervin Haji Amini

Abstract

As more people and organizations use computers, the demands on computer systems have changed. Therefore, with increasing the number of participants, the scalability has become an important issue for managing resource allocation. A centralized system relies on one computer node that processes the whole application. In such systems, a small number of nodes provide the resources that can only serve a limited number of clients. On one hand, if the centralized system becomes large enough, load balancing will create too much overhead. On the other hand, limitation on the network bandwidth increases the network bottleneck which affects the system performance. Decenralizing control, distributing processing loads and network bandwidth among all participating nodes will reduce the overhead in very large systems. In this light, Peer-To-Peer(P2P) systems offer an alternative solution to old server-based approach. Generally, a P2P system must have three essential properties. First, the system needs to be scalable. That is, it does not rely on single points of failures and bottlenecks. Second, the P2P system needs to be self-organized when peers arbitrarily join and leave the system. Third, the system needs to be fault-tolerant. In other words, the system must be robust when subjected to faults. In this thesis, we propose a dynamic decentralized P2P resource allocation method having as many resource allocators as required given the number of nodes, tasks and resources. In spite of the capability of the proposed resource allocation scheme for implementing on any structured overlay system, we implemented our application on *Pastry* as the underlying structured overlay system. We investigated the matchmaking process in the fully populated system when all resource allocators are active as well as the non-fully populated system when only some resource allocators perform resource allocation process. As we wanted to have a self-organized system in terms of resource allocators, we defined a mechanism that dynamically segment/desegment the ad hoc grid by promoting/demoting resource allocator(s) according to the workload of the existing resource allocator(s). This mechanism enables the ad hoc grid to adapt itself to the changing environment from a centralized to a decentralized form and back to the centralized form.

Laboratory : Computer Engineering
Codenumber : CE-MS-2008-09

Committee Members :

Advisor: Koen Bertels, CE, TU Delft

Chairperson: Stamatis Vassiliadis, CE, TU Delft

Member: Koen Bertels, CE, TU Delft

Member:	Stephan Wong, CE, TU Delft
Member:	Luc Onana Alima, DSL, Universite de Mons-Hainaut

This thesis is dedicated to my family, specially my parents for their love and support, and to my brother, Shahriar, who has given me his endless guidance, advice and support in all aspects of my life.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my deepest gratitude towards my supervisor Prof. Koen Bertels for his endless guidance, affection, attention, and support he has given me during my master thesis project. This thesis would not have been possible without his immense help, guidance and encouragement. I am forever indebted to him and I shall remember him always.

I am very grateful to my mentor, Tariq Abdullah, for his continuous help, assistance, patience, and useful advice during all the time with this thesis work. I would like to thank him for his constructive suggestions which have been always helpful.

Shervin Haji Amini
Delft, The Netherlands
September 22, 2008

1 Introduction

Distributed computing is a type of computing by which different parts of a program can be run in parallel on different computing nodes. These nodes are connected by a communication medium that are placed in different geographic locations. Solving large-scale communication problems at affordable cost is the main motivation for organizing distributed networks. Advances in distributed computing has resulted in widely applicable *grid* network environment which provides large resource sharing, secured access, balanced resource usage in which the distance must not make difference for efficiently access to the computer resources.
As the grid computing system is developed, the number of the users which are interacting with each other affects the behavior of the network. In other words, when such network is used in a local environment, there is not such a big deal to have only one resource allocator that satisfies providing resources for the different clients in the system. However, when the grid network is considered in large scale environments, some issues have to be taken into account such as the system throughput in terms of the efficiency and the response time, the bandwidth, the scalability and the fault resiliency. In this case, it becomes necessary to have sufficient number of resource allocators who can share responsibilities between themselves for providing services to the requesting clients. In order to achieve the above mentioned management strategies, there should be some predefined criterions by which a processing node could be nominated for being a resource allocator. In addition to the importance of the role of the resource allocator, another important issue which will be focused in this project is how the network is segmented among the resource allocators. This goal defines the procedure that assigns each resource allocator a certain number of clients that exchange data locally. If a request message can not be replied due to insufficient knowledge of a resource allocator in a segment, the message must be routed to an appropriate segment to have its request matched by the corresponding resource allocator. The message routing is one of the most important factors in every grid network due to its impact on grid performance.

In this chapter the motivation of this research is first identified. After that the framework of this thesis is presented.

1.1 Motivation and Problem Definitions

In large distributed and heterogeneous systems like grid, one of the important issues to be addressed is the system scalability. Scalability indicates that the performance and reliability of the system is not influenced when increasing the number of participating client nodes.
In conventional resource management, a centralized resource allocator matches jobs and

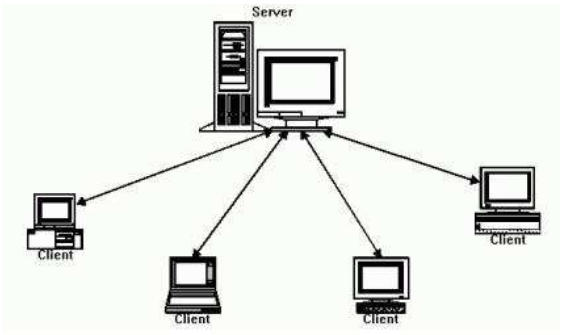


Figure 1.1: Centralized resource allocation

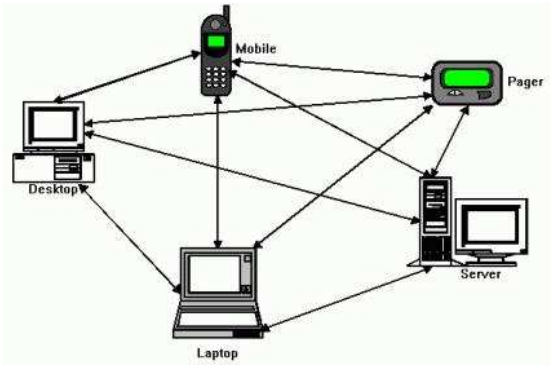


Figure 1.2: Decentralized peer to peer resource allocation

resources based on a relatively static model where the number of the clients do not change during the matchmaking process time (Figure 1.1). In addition, in such systems the resources are dedicated in the sense that the central controller is aware of the client nodes whose computational jobs are going to be executed on its resources at any given point in time.

In spite of the advantageous this scheme has such as easy administration and implementation as well as finding the best match for a job/resource in acceptable response time, this approach fails to work in dynamic systems, known as *ad hoc grids*, where the distribution of jobs and availability of resources may change at any time [19]. Moreover, maintaining a central directory for information in ad hoc grid is expensive and the whole system may fail mainly due to failure of the resource allocator. In this case, the resource allocation process has to be decentralized in such a way that the matchmaking efficiency is preserved and still be able to get the benefits of the centralized approach.

In this thesis project, we achieved decentralization of the resource allocation by introducing/removing the resource allocators to dynamically segment/desegment the ad hoc grid environment. The workload of the resource allocator is used as a criterion for introduction/removal of the resource allocator(s) who can communicate with each other in a P2P manner in order to route messages to different segments of the ad hoc grid (Figure 1.2).

1.2 Thesis Framework

This section discusses the framework of this thesis. The thesis is organized as follows: Chapter 2 reviews the background research of this thesis. First, a comparison between two types of P2P overlay systems is presented. Second, an overview of structured P2P overlay networks is discussed. Finally, some resource discovery methods utilized in P2P system are discussed and compared with each other.

Chapter 3 introduces the proposed algorithm by which the resource allocation process is decentralized. First, we present the description of the algorithm in detail. Second, we discuss about how a client node can find a responsible resource allocator in the ad hoc grid. Third, we investigate about the mechanism that enables the ad hoc grid to dynamically self-organize with respect to the changing environment.

Chapter 4 discusses about the results of the experiments conducted in Planet-Lab, the test bed for our experiments. First, we define the experimental setup. Then, we define the evaluation criteria for analyzing the decentralized resource allocation. Finally, we present the results.

Chapter 5 presents the conclusion of the thesis and describes the future research steps regarding the obtained results.

2 Related Research

Peer-to-peer (P2P) overlay networks are distributed systems which are composed of large number of distributed heterogeneous, autonomous and highly dynamic peers. In this environment, the peers share their resources with other participating peers. The shared resources can be the processing power(CPU time) or the storage capacity(memory). Each client in P2P network can act as a client and a server at the same time. In other words, the P2P system is fully decentralized. The term *overlay network* reflects the fact that the peers are connected to each other over an existing network, such as Internet, by which the overlay routes messages between the nodes. A self-organizing overlay network is formed by the peers on the Internet infrastructure which provides scalability, self-managing and fault resiliency. Scalability indicates avoiding dependency on centralized points. Self-managing deals with the adaptation of the overlay network to the changing environment in terms of arrival/departure of the peers. Fault resiliency represents the capability of the overlay network to be recovered when a failure occurs. The defects of client/server systems become obvious in large scale distributed environments. In client/server model, the resources are placed on limited number of nodes. In addition, the performance of such systems is degraded due to limitation of network bandwidth. Peer-to-peer overlay networks offer a solution to the limitations of the client/server systems at the cost of applying complex methods for improving system performance. However, the P2P overlay networks have several advantages such as decentralization of control and replication of data items that makes P2P systems very fault tolerant [10].
In this chapter, the P2P overlay networks are discussed as follows: Section 2.1 talks about two types of overlay networks in P2P systems. Section 2.2 reviews and compares some well known structured overlay networks. Section 2.3 discusses about the resource discovery mechanisms in P2P systems and finally section 2.4 concludes this chapter.

2.1 Basic Schemes for Peer-To-Peer Overlay Networks

Based on the procedure for storing data items in a peer, the basic infrastructure and the search method for locating data items, two main design approaches can be identified for building P2P overlay networks. They can either be *structured* or *unstructured*. With respect to the criteria mentioned in Table 2.1, we compare these two types of overlay networks in the following sections.

2.1.1 Unstructured P2P

In unstructured overlay networks, peers join the network by contacting any other existing peers. Thus, the peers are organized in a random graph. If a peer wants to find

Criterion	Peer to Peer Overlay Networks	
	Structured	**Unstructured**
Overlay Architecture	Structured graph	Random graph
Data Placement	Each data item is mapped to a responsible node.	Arbitrarily data placement
Routing method	Key-based	Flooding
Routing Performance	$O(log$(number of peers)	Limited by Time-To-Live
Neighbor Table (per node)	$O(log$(number of peers)	Not limited
Lookup Query Type	Simple lookup for rare items	Complex lookup for popular items
Maintenance Cost	Very high	Very low
Scalability	High for locating popular items	High for locating rare items

Table 2.1: Comparison of Structured and Unstructured Overlay Networks

a desired data item in the network, the query is flooded through the network because the peer does not know where the data item is stored. Even if Time To Live(TTL) is used to limit the lookup search domain, it is not easy to find an appropriate TTL value. When a peer receives the flood query, it sends all the content that match the query. The matched data items are selected based on the local data items stored on the peer. The main disadvantage of such search method is that the queries may not always be resolved. The search is successful when a data item is replicated on several peers, but if a peer looks for a data item which is shared by only a few other peers, that search may not be effective. Thus, unstructured systems do not scale well with growing number of queries and sudden increase in system size.
Unstructured overlay networks have some advantages. Such networks support complex queries and there is no constraint on data placement, for example each node chooses any other node as its neighbor in the overlay network and it can store the data item it owns. However, this approach tends to be inefficient as the flooding is used for search. Unstructured networks have very low maintenance traffic in terms of peers joining/leaving because there is no correlation between a peer and the data item managed by it and the flooding-based technique is used to easily update the peers about the changes in the overlay network. Most of popular file sharing P2P networks such as Napster [3], Gnutella [1] and KaZa [2] are unstructured.

2.1.2 Structured P2P

The first generation of P2P overlay networks, including Napster and Gnutella, had one basic limitation that was their inscalability due to sending the lookup queries to a large number of nodes in order to find items that are not widely replicated. Therefore, structured P2P networks has been proposed to make routing queries more efficient. The term "Structured" refers to the content location strategy in the sense that the data items (e.g., files) are inserted not at random peers but at specified locations. In this category, the

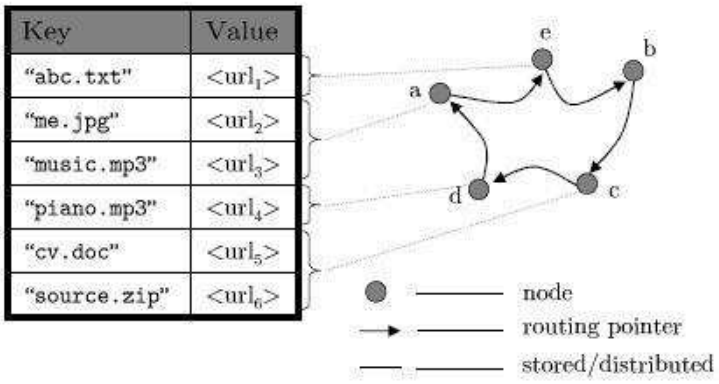


Figure 2.1: Example of a Distributed Hash Table [10].

peers are organized in a controlled manner so that any peer can be reached in a certain number of hops, typically logarithmic in the size of the network [16, 20]. In order to achieve this, structured P2P networks use the *Distributed Hash Table*(DHT) as a basic building stone.
A distributed hash table is a hash table which is distributed among the cooperating peers which we refer to as *nodes*. A hash table is a data structure that contains *key/value* pairs which are generally known as *items*. DHT supports *lookup* as a primary operation to find the value associated with any given key. This operation is performed by transforming the key using a hash function into a hash value that is used as an index to find the location having the desired value. Each node maintains a routing table consisting of pointers to other nodes(its *neighbors*). By means of the routing tables, a query is routed to a neighbor whenever it is not under responsibility of a receiving node. Figure 2.1 depicts a distributed hash table consisting of $(filename, URL)$ item pairs that are distributed among the nodes *a*, *b*, *c*, *d* and *e* having routing pointers to each other [10].
The foundation of DHT is an abstract address space consisting of N identifiers where $N = \{0, 1, 2, \ldots, N-1\}$ such that nodes and data objects to be stored are given identifiers in the same address space. The identifiers associated with nodes are called *node identifiers* and the ones associated with data objects are called *keys*. The overlay network organizes its peers into a structured graph such that each key is mapped to a unique responsible node and an item is stored at the node responsible for its key. Node identifiers are selected randomly in such a way that two nodes whose node identifiers are numerically similar are not physically close to each other on the address space.
Because of the cost for maintaining the address space such as inserting and updating items, each node takes the responsibility for a part of the items, which it stores locally. Different DHT-based systems have different schemes for assigning items to the nodes. For instance, an item may be assigned to its successor node whose node identifier is numerically closest to the item's key (Figure 2.2)[10].
As mentioned above, structured overlay networks utilize keys for efficient discovery of items. When a node wants to retrieve a certain data item, it constructs a lookup query for the key associated with that data item. The lookup query is forwarded by the peers whose node identifiers are numerically closer to the given key in the address space until

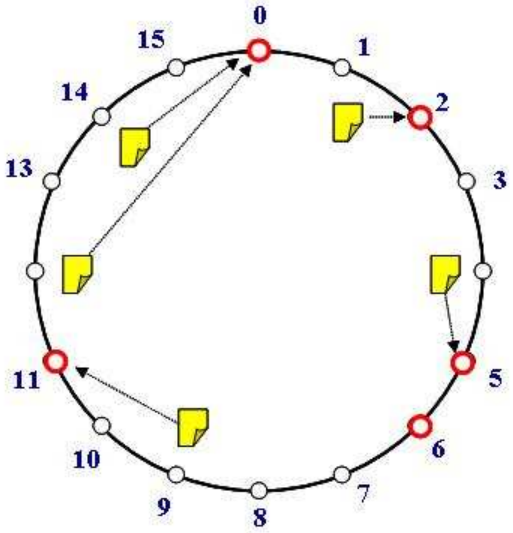


Figure 2.2: Example of an address space with 5 items and 5 nodes [10].

a responsible node is met for the data item. Assuming N is the total number of nodes in DHT-based systems, any data item can be found in $O(logN)$ overlay hops on average using $O(logN)$ neighbors per node [16].
In spite of finding exact match for lookup queries, structured overlay networks do not support complex lookup queries. Moreover, the structured networks have much more maintenance overhead than unstructured ones because there is a strong correlation between the location of data items and network topology. As structured P2P networks can efficiently find rare items using key-based routing, they do not scale well for locating highly replicated items. Although DHT-based systems offer various features such as decentralization and scalability, they have a drawback in terms of data item lookup latency. Despite of the closeness of two nodes with numerically similar node identifiers on the overlay address space, these nodes can be far from each other with respect to physical topology of the underlying network. Therefore, the latency of the lookup queries can be quite high in structured P2P overlay networks [16]. Some well known structured P2P networks are Content Addressable Network (CAN) [22], Chord [26], Pastry [23], Tapestry [29] and Distributed K-ary Search (DKS) [7].

2.1.3 What is better?

With respect to the above mentioned characteristics for both structured and unstructured P2P overlay networks, no definite decision can be made about which of the two types of the overlay networks outperforms the other. The reason for this acknowledgment is that the application, that we want to deploy an overlay network, and its performance requirements such as scalability, routing latency and the cost of network maintanance affects our desire to select the most suitable P2P overlay network [16]. However, despite the interesting properties of the structured networks, over the Internet today, the unstructured P2P overlay networks are more commonly used. The argument is that

although structured overlay networks guarantee to find an item if it exists, they are less suitable for current mass-market data sharing applications than unstructured overlays due to some reasons as follows. First, the need for finding extremely rare items is not required in mass-market data sharing applications. Second, P2P data sharing environments have not widely deployed DHT-based routing methods for their applications. Third, more tests are required to prove the ability of the structured overlay networks to implement key-based routing method [8]. Large body of research studies are ubderway to determine which overlay architecture is better for today's P2P applications.
As our application was implemented on a structured overlay, we investigate and compare the structured P2P overlay networks: CAN [22], Chord [26], Pastry [23] and DKS [7].

2.2 An Overview and Comparison of Structured P2P Networks

Structured P2P systems use the Distributed Hash Table (DHT) structure by which the location information about an item is placed at the peers with identifiers corresponding to the data item's unique key. The interesting feature of DHTs is that the uniform random node identifiers(nodeIds) are assigned to the set of peers in an identifier space. Unique identifiers, known as keys, are attributed to data items. The keys are selected from the same identifier space. The overlay network maps the keys to the active peers in the identifier space. Each peer keeps a neighbor table containing the nodeIds as well as the IP addresses of its neighboring nodes. The general approach for locating an item in DHTs is to route the message to the nodes with nodeIds closer to the given key in the identifier space [16].

2.2.1 Comparison Criteria

The comparison between the Structured Overlay Networks presented in this report is based on different criteria [**?**]. Some of these criterions have been briefly explained as follows:

- **Overlay network.** In this criterion, the architecture of the overlay system is described. The building block of an overlay network affects the routing table structure used by each peer for contacting other peers.
- **Lookup protocol.** This criterion indicates how a lookup process is performed. In addition, the lookup protocol determines the performance of an overlay network.
- **Responsibility of nodes for data items.** It is important to know how the identifiers of both nodes and data items are related with each other. In other words, we want to know which node takes the responsibility for which data item in an overlay network.
- **Nodes' joining, leaving and routing maintenance.** One of the most important features of the overlay networks is to specify the way a node follows to join

or leave the system. As these operations change the network, some maintenance policies are required to update the information stored on the nodes. However, the cost of the routing maintenance has to be taken into account.

- **Reliability and fault resiliency.** This feature shows the robustness of the overlay system when subjected to nodes' departures or failures.

- **Applications and implementation.** For each P2P overlay system, some of the developed applications using that overlay network are presented in this section. Furthermore, the available implementations for each overlay system are named as well.

2.2.2 Structured P2P Overlay Networks

As was mentioned in section 2.1.2, the keys are assigned to data items and the nodes are organized in the overlay network in such a way that each pair is mapped to a peer in the network. In spite of efficient locating of items by the given keys, performing complicated lookups is not supported in this type of networks. In general, this type of network topology supports precise-match queries. In this section, some well known structured overlay networks are described and compared using the above mentioned comparison criteria.

2.2.2.1 Content Addressable Network(CAN)

The Content Addressable Network (CAN) is a purely decentralized P2P infrastructure with the hash table structure. The CAN is designed to be scalable (nodes store information about their local neighbours), fault-tolerant (recovering after nodes failure), and self-organizing [22].

- **Overlay network.** The CAN is designed as a d-dimensional torus where the basic operations performed on its hash table are the lookup, the insertion and the deletion for (key, value) pairs. Each CAN node possesses a zone which is a part of the entire hash table indeed. The information about the adjacent zones is also included in the corresponding node table.

- **Responsibility of nodes for data items.** To store a pair (k, v), key k is mapped onto a point P in the coordinate space by a hash function. A node, owning the zone containing point P, then stores the corresponding (key, value) pair.

- **Lookup protocol.** Routing in a CAN is performed by nodes' routing tables holding the IP addresses of their adjacent peers as well as their coordinate sets. The lookup query for retrieving an entry corresponding to key k rehashes the key to obtaining the value from point P. If the requesting node or its neighbours do not have point P, the CAN routes the message through the zones until it reaches the node within a zone where P is placed. Given a CAN message, the message is routed to a destination whose coordinates are resided in the message such that the

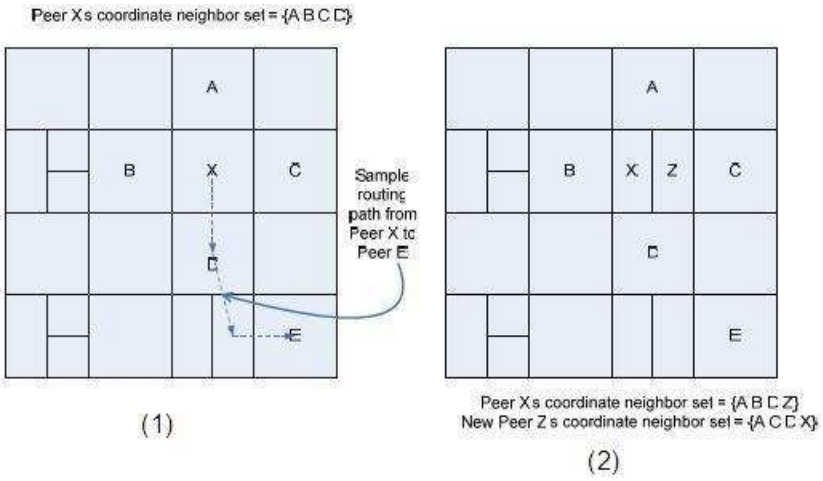


Figure 2.3: The status of CAN before(1) and after(2) node z joins [16].

destination address matches closely a neighbour's coordinate set. If d represents the number of dimensions in the coordinate space, each message is routed towards its destination in $(d/4)(n^{1/d})$ hops and each node maintains $2d$ neighbours in its routing table. As there may be several paths between two nodes in the coordinate space, a node can route the message towards a next available path when some neighbours of the requesting node fail to response [16].

- **Nodes' joining, leaving and routing maintenance.** When a new node joins the system, it must obtain a chunk (zone) of the coordinate space. The contacted peer splits the entire zone in half and transfers the ownership of a half to the new joining node. The CAN construction process takes the following three steps [22]:

  1. The new node must first find the IP address of one or more bootstrap nodes in the system. The joining node then gets the IP addresses of a number of live nodes provided by the bootstrap node.
  2. A point P is randomly chosen at identifier space and a request message is sent via any available CAN node. By exploiting the CAN routing strategy, the message is routed until it reaches the node in whose zone P lies.
  3. The zone of the contacted node is then split between the contacted node and the new node. Upon the split, the new node learns the IP addresses of its neighbors from the previous owner. The neighbors of the newly joining node include the neighbors of the previous owner plus the previous owner itself. Figures 2.1 and 2.2 show the CAN status when a peer z joins the overlay network [22].

  When a node leaves a CAN, its zone and associated (key,value) pairs has to be merged with a neighbour to create a valid zone. If no node is found for zone merging, then the zone is transferred to a neighbor owning the smallest zone.

Under normal conditions, a node sends periodic update messages to each of its neighbors. These updates contain the node's zone coordinates and the coordinate zones of its neighbors.

- **Reliability and fault resiliency.** When a node can not receive the periodic update messages from its neighbor for a long time, all the failed node's neighboring nodes try to take over the zone of the failed node. Among the failed node's neighbors, a node with the smallest zone takes over the failed node's zone.
  Replication in CAN is achieved by creating multiple instances of the coordinate space. Each instance is called a reality in which each peer owns a different zone from the owning zones in other realities. Thus, the data availability is improved by replicating of each (key,value) pair at distinct nodes on different realities.

- **Applications and implementation.** Some applications have been implemented on CAN regarding efficient insert and retrieval of file contents in a large storage system. Moreover, a DNS-like application has been constructed on CAN which is utilized for wide area name resolution services [22].

2.2.2.2 Chord

Chord uses consistent hashing to assign keys to its peers. As each node is assigned the same number of keys, the total workload is evenly distributed among the nodes by utilizing consistent hashing functionality. If there are N nodes in the system, each node maintains routing information about $O(log_2 N)$ other nodes, and all lookups are resolved in $O(log_2 N)$ hops. Simplicity, provable performance and correctness are the features that differentiate Chord from other structured P2P overlay networks [26].

- **Overlay Network.** The overlay graph of the Chord ring is assumed to be a circular identifier space of size N . An m-bit node identifier is obtained by hashing the node's IP address whereas a key identifier is generated by hashing the key. The identifiers are ordered on an identifier circle (Chord ring) modulo 2^m. If identifiers are represented as a circle of numbers from 0 to $2^m - 1$, a Chord node has a pointer to the first following node as well as the first preceding node on the identifier space. In addition, a node maintains $O(log_2 N)$ pointer to the corresponding following nodes in the Chord ring.

- **Responsibility of nodes for data items.** A key k is stored at the first node that follows clockwise on the identifier space. This node is called the successor node of k, denoted as successor(k).

- **Lookup protocol (Simple Key Location).** In order to perform a simple and slow lookup procedure, each node tries to contact its current successor node on the identifier circle. The successor nodes pass the query lookups around the chord ring based on a given identifier until the desired identifier is found to be between a pair of nodes. In this case, the query is mapped to the second node for resolving the query [26].

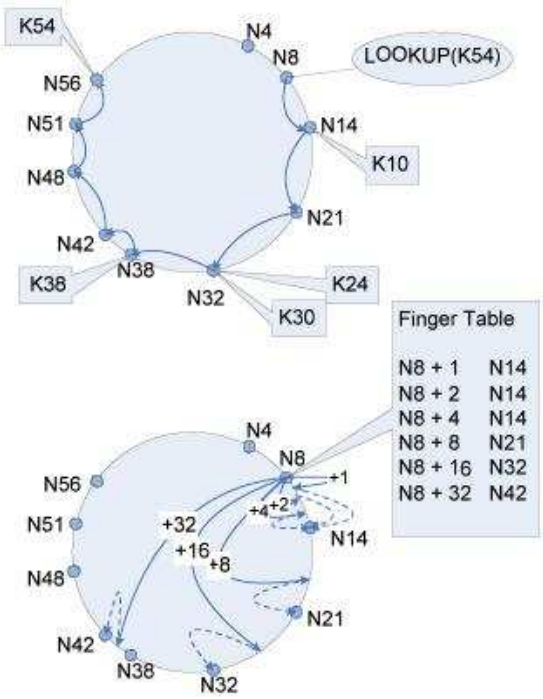


Figure 2.4: Chord ring (identifier circle) consisting of 10 nodes storing five keys. The path for locating the key 54 by a look up query at node 8 is depicted in this figure [16].

- **The lookup protocol (Scalable Key Location).** If m is the number of bits in the key/node identifiers, each chord node n maintains a routing table, known as finger table consisting of m entries. The identity of the first node s that follows n by at least 2^{i-1} is specified to be the i^{th} entry in the node's table on the identifier circle, i.e., $s = successor(n + 2^{i-1})$, where $1 \leq i \leq m$. Node s is known as the i^{th} finger of node n , and is denoted as n.$finger[i]$. Figure 2.4 depicts the finger table of node 8(N8). Node 14 is the first entry in the finger table because node 14 is the first node that succeeds $(8 + 2^0) mod 2^6 = 9$ [20, 26].
  This lookup protocol design has some important characteristics. The most important aspect of the lookup procedure is that the information about a small number of nodes is stored in the node's routing state. Moreover, the node in the context has more information about the other nodes which are closer to itself on the identifier circle than the nodes which are farther away.

- **Nodes' joining, leaving and routing maintenance.** The correctness of the Chord lookups depends on how each node's successor pointer is updated. To achieve this goal, the *stabilization* protocol is used. This maintenance protocol is run periodically by each node in which the Chord's finger tables and successor pointers are updated.
  According to [26], when node n wants to join the Chord ring, $n.join(n^{'})$ is called, where $n^{'}$ is a known Chord node. If no node is present in the system, then $n.create()$

is invoked to make n be the starting node in the Chord network. Node $n^{'}$ is asked by join() method to find the immediate successor of n. In addition, the procedure $notify$ is called inside the stabilize function to notify node n's successor of n's existence which gives the successor the opportunity to change its predecessor to n. The procedure $fix\_fingers$ is periodically called to initialize the newly joined nodes's finger tables and updating existing nodes' finger tables with new joins as well. If the node's predecessor has failed, the procedure *check_predecessor* is called by each node to clear the node's predecessor pointer.
The above mentioned stabilization scheme guarantees the reachability of existing nodes when adding nodes to the Chord ring, even if some nodes are joined or lost concurrently [26].

- **Reliability and fault resiliency.** Knowing which node is going to be a successor of a Chord node causes the correctness of the Chord protocol. When some peers fail, it is possible that a node does not know its new successor, and it has no chance to learn.
  To increase robustness, a successor list of size r , which contains the peer's first r successors, is maintained in each Chord node. When the peer's successor does not respond, the peer contacts the next node on its successor list. If p be the probability of each node failure, $p^{'}$ is the probability that all r peer's successors fail simultaneously. The system will be more robust by increasing the parameter r. It is important to note that when a node leaves the network, its effect on the system can be treated as a node failure. In this case, the leaving node may transfer its keys to its successor and notify its predecessor as well as its successor about its departure. If $log_2N + 1$ successive nodes fail simultaneously, the Chord ring is disconnected or a data item is lost [26, 16].

- **Applications and implementation.** Chord has been used as a basis for a number of subsequent research projects. The Chord File System (CFS) is one of the important research projects in which several content providers cooperate to store and serve each other's data. In this application the total workload is distributed evenly among the hosts which will lower the system cost.
  In Chord-based DNS, a look up service is provided with host names as keys and their IP addresses as values. In such service, each host name is hashed to a key. While the conventional DNS systems depend on a set of central servers, Chord based DNS operates without relying on any root servers since it is accounted as a pure decentralized P2P network.
  The original implementation of Chord is in C++. However, the Chord has also been implemented in $C\sharp$ as it was supplied with a file sharing application.

2.2.2.3 Pastry

Pastry is a decentralized, scalable distributed object location and routing substrate in a very large network of nodes connected via the internet [23]. Pastry performs the application-level routing and query lookups in a large scale overlay network based on the Internet infrastructure. Pastry takes into account the network locality by which the

distance traveled by each message is minimized according to a scalar proximity metric such as the number of IP routing hops. It is important to know that Pastry can be automatically adapted to the arrival, departure and failure of nodes.

- **The Overlay Network.** A unique identifier, chosen in a circular identifier space ranging from 0 to $2^{128} - 1$, is assigned to each Pastry node. Given a message and a key, the Pastry node routes the message to the node whose identifier is closest to the key in less than $log_B N$ hops where B ($B = 2^b$) is a configuration parameter with $b = 4$ and N represents the number of nodes in the overlay system. As the node identifiers are randomly generated by hashing the IP addresses/Public keys, the corresponding nodes may seem near each other geographically whereas they communicate with each other through a node in a totally different place because the hash of their identifiers are far apart in the identifier space.
  Figure 2.5 shows an example of a pastry routing state that uses 16-bit node identifier($b = 2$) in a Pastry node with identifier 10233102. The node identifiers in this table have been represented as "*common prefix with local node identifier + next digit + rest of node identifier*". According to this Figure, each node keeps a routing table, a neighborhoodset and a leafset.
  The leafset provides correctness. It consists of a set of nodes with $|L|/2$ numerically closest larger and smaller node identifiers respectively with regard to the present node's identifier. The value of L is generally set to 24. Thus, around 12 closest neighbors to the left and the same to the right are stored in the leafset. If only leafset was to be used for accomplishing routing, the message would always send to the furthest left or the furthest right entry of the leafset if the key is not within the extents of the leafset. However, this would cause $O(N)$ hops(N is the number of nodes in the network) besause a message has to traverse N/k nodes(k is $1/2$ number of nodes in the leafset) to reach to the other side of the ring. Thus, it would be correct, but not scalable, to only use the leafset for routing.
  The routing table provides the scalability as it gives $O(logN)$ routing hops to reach the destination node. The routing table is consisted of $\lfloor log_{2^b} N \rfloor$ rows each one has $2^b - 1$ entries. As can be seen in Figure 2.5, each entry in row n refers to a node whose identifier is common with the present node's identifier in the first n digits, but whose $(n+1)th$ digit is different form that of the present node. The top rows in the routing table are the furthest away nodes in the ring that are known to a node. As we move down the rows of the routing table, each row represents a magnitude closer than the row above. For example, all nodes in row 6 are closer than any node in row 7 (the top row of the routing table is 7, and the bottom row is 0). Most, if not all, of the entries in the lowest rows of the routing table will also be found in the leafset. Sometimes there is hole(s) in some rows the routing table. This can be because there is no node in the network with an identifier that matches the prefix, or because the node that is known with an appropriate prefix recently failed.
  The node idetifiers and IP addresses of $|M|$ nodes which are physically closest to the local node are maintained in the neighborhood set where the value of M is

Node Identifier 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

7	-0-2212102	1	-2-2301203	-3-1203203
6	0	1-1-301233	1-2-230203	1-3-021022
5	10-0-31203	10-1-32102	2	10-3-23302
4	102-0-0230	102-1-1302	102-2-2302	3
3	1023-0-322	1023-1-000	1023-2-121	3
2	10233-0-01	1	10233-2-32	
1	0		102331-2-0	
0			2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Figure 2.5: Pastry node state with node identifier 10233102, $b = 2$, and $l = 8$[23].

usually set to 24. The neighborhoodset is populated based on a proximity metric such as the number of IP routing hops or geographic distance.

- **Responsibility of nodes for data items.** In pastry, a key is under responsibility of a node whose identifier is the closest to the given key. As the identifiers in the leafset are the closest ones to the local node's identifier, the key is also replicated to a certain number of nodes with such identifiers in the leafset.

- **Lookup protocol.** The node identifiers and keys are considered as a sequence of digits with base B. In order to forward the message to its destination, the message is routed to the node whose node identifier is numerically closest to the given key. According to [23], the node first checks the message key to see if it falls within the range of node identifiers in its leaf set. If it is in the range, the node whose node identifier is closest to the key is selected to receive the message as the destination node.
  If the leaf set can not satisfy finding the right node, then the common prefix of the key with the node's identifier is computed and the routing table is searched for the node whose identifier shares a longer prefix with the key than the present node shares with the key.
  If the corresponding entry in the table is empty or its associated node can not be reached, the message is forwarded to a node whose identifier has at least a common prefix with the local node's one but is closer to the key than the local node's identifier. Pastry guarantees that such a node exists in the leafset if it exists in the network. In other words, if no node can be found with a better prefix match in the routing table, then there is a guarantee that either:

  1. the local node knows of a node who is closer than itself in the identifier space(using the routing table or the leafset) or

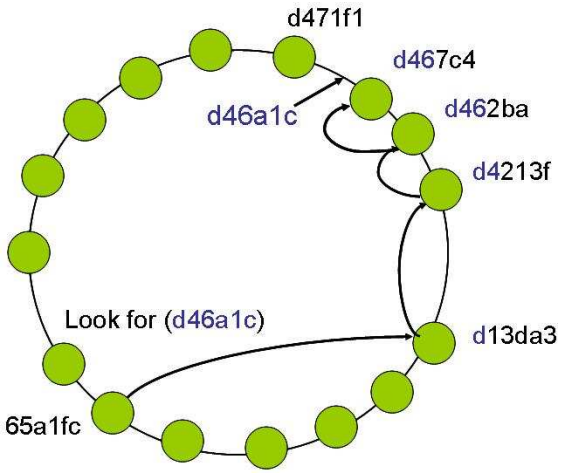


Figure 2.6: Message routing in Pastry from node with identifier *65a1fc* with key *d46alc*[9].

2. the local node is the root(closest node to the key) or
3. There was a massive simultaneous failure in leafset.

Figure 2.6 depicts the routing path of a message in Pastry. Using the routing table, the message is forwarded by the local node to a node whose node identifier shares a prefix with the key which is at least one digit(or b digits) longer than the prefix that the key shares with the identifier of the local node. If no such node is found in the routing table, the message is forwarded to a node whose identifier shares the same prefix with the key as the local node does, but whose node identifier is numerically closer to the key. As we mentioned above, two nodes having adjacent node identifiers on the overlay network are far away from each other in terms of geographic distance. The reason for such an observation can be explained by the populating of entries in the routing table. In fact, each hop that a message travers to reach the destination moves down a row in the routing table. The more we go to the lower rows of the routing table, a longer common prefix is needed for a match and the fewer candidates in the network are allowed to go into the routing table entries. Pastry uses proximity measurements to try to select the closest node in each routing table spot. As the higher rows are more populated than the lower ones, there is a high probability to choose a close neighbor(in terms of network delay) for routing a message whereas we have the "average network delay" which is much higher when dealing with the lower rows of the routing table. So, each row gets us 1 digit closer in "identifier space", but because there are exponentially fewer candidates, we get exponentially further in "physical" network space.

- **Nodes' joining, leaving and routing maintenance.** Since upon arriving a new node its state table has to be initialized, it is assumed that the new node, with identifier X, knows initially about a neighbor node A. This node is presumed to be close with respect to the underlying physical network. By means of the above described routing algorithm, a *join* message with the key equal to X is routed by

node A until it reaches the node Z whose identifier is closest to X. Node X then initializes its route state with as physically close nodes as possible using the route states sent by the nodes who already forwarded the *join* message. The route state initialization process takes place as follows.
Node X's neighborhood set is initialized with the one in node A as node A is nearby X. The leafset of node X is also affected by the one in node Z as its identifier is the closest one to node X. The row entries of X's routing table are initialized according to the nodes which are placed along the path from A to Z [23]. A copy of X's constructed state is finally sent to its corresponding nodes in its neighborhood set, the leaf set and the routing table as well. These nodes can update their route states as well. The cost of the above mentioned joining process is estimated to be $3 \times 2^b log_{2^b} N$ [23].

When a Pastry node can not communicate with its immediate peer in the network, that peer is considered as a failed node. In order to recover from a failure, the failed node's entry in the leafsets needs to be replaced by the neighbor nodes. To acheive this goal, the failed node's neighbor asks for the leafset of the furthest live node on the side of the failed node [23]. Among the nodes which are not common in both leafsets of the present node and that furthest node, the most appropriate one is inserted into the present node's leafset.
For repairing a failed node entry R_l^d in a node's routing table, where R_l^d refers to an entry at column d and row l, the node first contacts a node in the same row but with different column number (i.e., R_l^i, $i \neq d$) and asks for the failed entry R_l^d in their routing table. If no appropriate entry is found, the present node tries to find a node in next higher rows in its routing table [23].
In order to update the neighborhood set, a node tries to contact all its neighbors periodically. If a failed node is detected in the neighborhood set, the detecting node requests the neighborhood tables from all its neighbors to keep its neighborhood set up to date.

- **Reliability and fault resiliency.** In Pastry, a copy of a requested information is stored on k nodes in a node's leafset having numerically closest identifiers to a key in the identifier space. Thus, the queries for the same information can be first found on a node closer to the requesting node than the responsible node.

- **Applications and implementation.** A variety of peer-to-peer applications can be supported by Pastry such as data sharing and global data storage. In PAST, for instance, the hash of the file's name is computed as a fileId and considered as a Pastry key for a file. A file can be looked up by sending a message via Pastry, using the fileId as the key. The Pastry nodes having the closest identifiers to the fileId can store the replicas of the file, despite nodes failure and arrival.
The SCRIBE publish/subscribe system is another application in which a node with an identifier closest to the topicId of a topic stores a list of subscribers. This node acts like a meeting node where a message is sent by subscribers via Pastry using the topicId as a key. A publisher sends data to that meeting node via Pastry using

the topicId. Then, the data is forwarded along the multicast tree from the meeting node to all subscribers.
FreePastry, written in java, is the platform on which the Pastry nodes can route their messages.

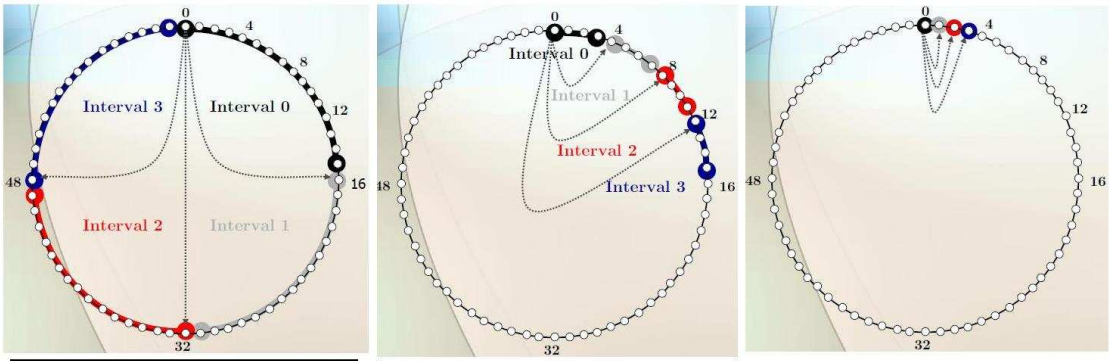
2.2.2.4 DKS

The *DKS(N,k,f)*, standing for Distributed k-ary Search, is a fully decentralized overlay network where N represents the maximum number of nodes in the network; k is the number of the search areas inside the network and f is the fault tolerance degree. It provides a distributed hash table as an infrastructure for distributed inserting and retrieving (key,value) pairs. The maximum number of the nodes in the network is $N = k^L$ where k, an integer, is equal or greater than 2 and L is a parameter which represents the number of levels in the search. [7, 28].

- **The Overlay Network.** The DKS can be counted as a generalization of the Chord overlay network. The identifier space in DKS is a circular space modulo N. Each identifier (node) is a root of a virtual k-ary tree that spans the whole identifier space. The height of such a tree is the logarithm of the system size. Each node in DKS mainatains a routing table. The routing table consists of $L = log_k(N)$ levels which consists of k intervals each. Moreover, each node maintains a reponsible(contact) node for every interval in its routing table. The responsible node is defined to be the first node encountered, moving in clockwise direction, starting at the begining of the interval [7]. For any level l, each node is itself responsible for the first interval. Assuming that the search area is first equal to the whole identifier space, it is divided into k equal parts in each search step until there are k equal parts each containing only one node. Figure 2.7 illustrates the division of the identifier space performed by node 0, for $N = 64$ and $k = 4$. The routing tables containing the levels, intervals and respective responsible nodes is also shown in this Figure [10]. At each level, the intervals are represented as I_i where $0 \leq i \leq k - 1$.

- **Responsibility of nodes for data items.** Like other Peer-To-Peer overlay networks, the nodes as well as the data items are assumed to be identified by the same logical name space. The logical name space is known as the identifier space consisting of identifiers $\{0, 1, 2, \ldots, N - 1\}$. Once N is defined, all nodes and keys are mapped onto the identifier space by means of a hash function. In this identifier space, a key/value pair, denoted as (k, v), is under responsibiity of a node with the closest matching identifier to the key.

- **Lookup protocol.** When a node n wants to search for a key with identifier id in the network, it first finds the interval within the first level($l = 1$) where id belongs to. If the node is itself responsible for that interval, the node tries to find an appropriate interval at the second level and so on until it finds an interval for which the checking node is not responsible. The *lookup* message is then forwarded to that node. When the lookup message is received by a node n' for key identifier



Node0	I_0	I_1	I_2	I_3
Level 1	[0,16[	[16,32[	[32,48	[48,0[
Responsible	0	16	32	48

Node0	I_0	I_1	I_2	I_3
Level 1	[0,16[	[16,32[	[32,48	[48,0[
Responsible	0	16	32	48
Level 2	[0,4[	[4,8[	[8,12[	[12,16[
Responsible	0	4	8	12

Node0	I_0	I_1	I_2	I_3
Level 1	[0,16[	[16,32[	[32,48	[48,0[
Responsible	0	16	32	48
Level 2	[0,4[	[4,8[	[8,12[	[12,16[
Responsible	0	4	8	12
Level 3	0	1	2	3
Responsible	0	1	2	3

Figure 2.7: A DKS with $k = 4$ and $N = 64$. The left most part of the figure shows the intervals on first level. The intervals of second and third levels are shown in center and right most parts of the figure respectively [10].

id, n' checks if id is between itself and its predecessor. If so, then n' returns the value associated with id to n. Otherwise, it attempts to find an interval at level $l + 1$ within which id exists. This process continues until the responsible node for id is found. In this case, the responsible node returns the associated value to the message originator recursively. In general, a lookup is resolved in $log_k N$ routing hops [7].
Figure 2.8 shows an example of a lookup search with key identifier 27 from node 0 in DKS when all the nodes are alive. *Message level* indicates the level size in the routing table by which a correct interval is found for a key. As can be observed from the routing tables in Figure 2.7, the size of the interval is divided by k for each successive level. Thus, the receiving node tries to find an interval whose size in not larger than the the previous interval size. So, at each routing step, the size of the level is increased by one.

- **Nodes' joining, leaving and routing maintenance.** The joining process in DKS is performed in an atomic manner in such a way that the *join* message, initiated by the joining node n, is routed to a known node n' in the network. This node performs the lookup search and sends the reply, which is the closest successor node $S(n)$ in DKS ring, back to the joining node n. After that n asks $S(n)$ for insertion into the DKS ring. $S(n)$ calculates the initial routing table for the new node n so that it can join the network.
The leaving process is first initiated when the leaving node n informs its successor node $S(n)$ of its departure and asks him to grant permission for leaving the DKS ring. Thus, every received message by leaving node n intended for performing an

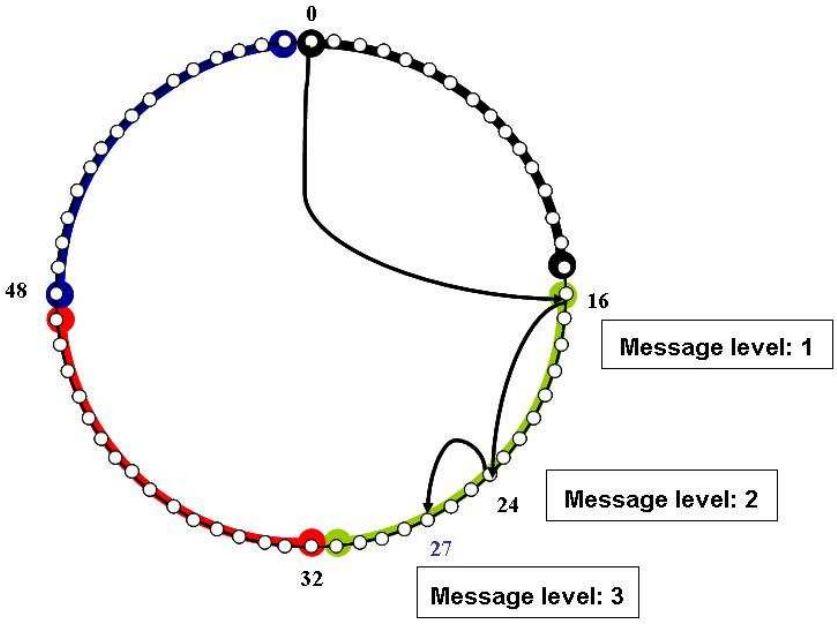


Figure 2.8: Example of a lookup search from node 0 for key identifier 27 in a DKS ring with $N = 64$ and $k = 4$.

operation such as a lookup search, joining a new node or inserting a key/value pair is queued at n. Then $S(n)$ sends a message to n granting him permission to leave the DKS ring. Finally, n sends all its queued messages to $S(n)$ and leaves the network without any additional messages [7].
The routing information maintenance in DKS is performed in a way that the least communication overhead is generated during the nodes' arrivals and departures and the correctness of the look up queries is guaranteed in case of simultaneous node failures. However, the novelty of this Peer-To-Peer overlay network is that the maintenance of the routing paths is not performed in a separate phase which is the case in most DHT-based systems. For instance, Chord and Pastry periodically run seperate *stabilize* routines to ensure the accuracy of the lookups when the set of participating nodes changes [6].
Although any incorrect routing information can be quickly detecetd by *stabalize* technique, it consumes unnecessary bandwidth when the the rate of joining, leaving or failure process is low. However, in DKS, the process of updating the routing information is done when the overlay network is in operation. In other words, DKS uses the information embedded in the lookup messages to correct the invalid information. In this way, the bandwidth consumption can be saved significantly in DKS network.

The routing table in DKS are maintained using the *correction-on-use* strategy. Each lookup message carries information about the position of the receiver node including the level and the interval that the receiver can be found there.

This information helps a receiver node to notify the sender about a better candidate for the lookup process according to the receiver node's knowledge. However, the benefit of utilizing the correction-on-use is dependant on the ratio of the total traffic (e.g., number of lookup messages) over the dynamism in the network. That is, if the corresponding ratio is low, the system converges slowly towards its expected performance ($log_k N$). In order to overcome this problem, a complementary mechanism, called *correction-on-change*, has been introduced in DKS [6]. The novelty of correction-on-change is that the overlay network is corrected only when there is a change in the network due to join, leave and failure. Thus, no extra cost is paid when there is no dynamism(change) in the system. [7, 6].

- **Reliability and fault resiliency.** A node m is considered failed if it can not respond within a given time. In this case, the detecting node n tries to replace the failed node m in its routing table. In order to achieve this goal, a list of nodes is stored at node n containing the first $(f + 1)$ nodes that succeed node n (f is the fault-tolerance of the network). There are two scenarios for handling node failure depending on whether m is in n's successor list or not. If m is in the successor list of n, it is replaced by a node in the list that follows m. Otherwise, n replaces failed node m by a node which is the first successor of m [7]. Furthermore, every node replicates its information on the nodes in its succesor list to ensure the reliable lookup even if f nodes in successor list fail simultaneously.

- **Applications and implementation.** The broadcast and the multicast applications have been deployed on DKS. The main implementation of DKS is in Java programming language.

2.2.3 Discussion on Structured P2P Overlay Networks

DHT-based systems used in overlays' query routing have strong theoretical foundations. These P2P systems guarantee that if a key exists in the network, it can be found in certain amount of time. However, a message may be routed to a requesting node's immediate neighbor which can be far away from the present node in terms of proximity metric. This causes a high latency lookup search which affects the network delay. Moreover, DHT-based systems are deployed in purely decentralized systems in which all peers take part in sharing their data items. This can result in a bottleneck for peers with low routing state. It is also important to note that the nodes in the above structured P2P overlay networks route lookup messages in a greedy manner. That is, each node forwards a message based on the information that it stores locally. This increases the routing distance at each routing step.
In Pastry the routing algorithm works towards matching longer prefixes until the responsible node or a nearby node having the desired data item is found. Coping with excessive dynamism in one of the challenges a DHT-based system may face. Pastry overcomes this problem by using its locality properties. [16]. By means of this feature, Pastry measures the Round Trip Time (RTT) delay to a number of peers when populating each routing table entry. So, the closest nodes, in terms of underlying network, with appropriate

identifiers are selected when building the routing tables. [23].
In Chord, the keys and peers are mapped to an identifier ring. The queries are guaranteed to reach the destination in a logarithmic number of hops and the keys will be distributed in the ring uniformly by means of Consistent Hashing in spite of arrivals/departures. The routing maintenance of Chord in terms of updating the finger tables is preserved because each node runs the stabilization routine periodically. The stabilization process needs to only update the predecessor and successor nodes [26]. The communication cost incurred by the Chord's stabilization protocol is dramatically reduced in DKS by exploiting the correction-on-use technique. As was mentioned in section 2.2.2.4, the routing table entries in DKS are corrected by carrying the routing information with respect to a node to which a lookup message is forwarded. It is important to note that DKS is applicable for P2P systems when the number of lookups and key/value insertions are much higher than the total number of nodes in the system.
The main motivation for designing CAN was to keeping the network performance (resolving lookup messages) at a constant level. To achieve this goal, each node maintains a routing table including its immediate neighbors. This key design helps CAN to cope with high dynamism rate which affects the routing performance.
Table 2.2 summarizes the comparison of structured P2P overlay networks which have been discussed in section 2.2.2.

P2P Overlay Network	**Structured Peer-to-Peer Overlay Network Comparison Criteria**							
	Overlay Architecture	**Lookup Protocol**	**Routing Hops**	**Routing State**	**Maintenance Policy**	**Assignment Policy**	**Reliability**	**Route Location**
CAN	Multi dimensional identifier space.	Mapping key/-value at point P using hash function.	$O(d.N^{1/d})$	2d	Stabilization	Key to zone owner.	Multiple nodes responsible for each data item.	RT,node table.
Chord	m-bit circular identifier space.	Matching key and a successor nodeId.	$O(\log_{2^b} N)$	$\log N$	Stabilization	Key to successor node.	Replicate data on consecutive successor nodes.	Finger table, successor list.
Pastry	128-bit circular identifier space.	Matching key and nodeId prefixes.	$O(\log_{2^b} N)$	$2^b \log_{2^b} N + 2^b$	Determinism + Stabilization	Key to numerically closest node.	Replicate data on nodes with closest nodeIds to a data key.	RT, leaf set, neighborhood set.
DKS	Circular identifier space like Chord.	Searching routing table at level l for interval I_i^l such that key $\in I_i^l$.	$O(\log_k N)$	$(k-1)\log_k N$	Correction on use + Correction on change	Key to successor node.	Replicate data on dispersed well chosen nodes.	RT, successor list.

Table 2.2: Comparison of Different Structured P2P Location Schemes(RT and nodeId represent Routing Table and node Identifier respectively)

2.3 Overview of Grid Recource Discovery mechanisms

Efficient resource discovery is a crucial problem in the development of computing Grids[27]. There are methods in computational grids that are used for sharing large and different types of resources such as processing power, storage capacity, software and file contents as shared resources or scientific instruments and sensors as dedicated devices. Resource discovery is a basic service in grids. The resources (or contact addresses of resources) are identified based on a given set of desired attributes. The adaptation of grid technology to ad-hoc networks has caused to the development of ad-hoc grids which enable mobile users to share computing resources and information. The remainder of this section reviews some proposed systems that adopt an unstructured P2P architecture to grid resource discovery. A comparison of such systems is presented in Table 2.4 at the end of the section.

2.3.1 Fully Decentralized Resource Discovery in Grid Environments

A resource discovery mechanism in [14] has been proposed which utilizes a peer-to-peer approach for selecting the resources in a dynamic environment. Four request-forwarding algorithms have been evaluated where three of them improves search performance by using past experience (i.e., remembering a neighbor which was most helpful in providing required resources). In contrast to the typical resource discovery methods which use name as a search key for using a resource, the proposed resource discovery algorithm determines its requests by means of the resource attribute and the associated quantity, for example the name and the capacity of the memory and the CPU amount time.
It is assumed that each participating user can communicate with one or more nodes known as peers in a local environment. These nodes store and provide access to the information with respect to the shared resources in the local environment. Users send their requests to their local peers(nodes). If the requested information can be found locally, it will be returned to the user by the contacted peer. If not, the peer contacts another node. The process of request forwarding may be performed by the intermediate nodes until a match can be found by a node, which will be sent back directly to the first forwarding node, or the message's time-to-live (TTL) expires. The framework is defined by two basic concepts. The first one is the membership protocol which provides each node with information about other nodes. A node(peer) can join the system by contacting any other node who sends its membership information to the joining node. The second concept is the request forwarding algorithm that is used to decide to which node (among the known nodes) the requests must be forwarded. Because the quantity of a resource offered by a node may change ovet time, the nodes store the addresses of the other peers and not their available resource amount. In addition to the contact addresses, nodes may store information regarding the type of the requests that were answered previously by a particular node. This information can help a node to forward next requests to more appropriate nodes.
[14] proposes four request forwarding algorithm which are employed based on the availability of past information and the reputation of the node(s) for answering previous requests. The algorithms are as follows:

1. Random forwarding: the request is forwarded randomly to a node because no information is initially stored on the grid nodes.

2. (Experience + random) based forwarding: nodes send their requests based on the information they have already collected about the other nodes who responded to the past similar requests. If no such an experience is found, the message is forwarded to a random node.

3. Best-neighbor based forwarding: Each node maintains the number of responses(regardless of their associated request types) received from each peer. A node sends its request to a peer who responded the largest number of requests.

4. (Experience + best-neighbor) based forwarding: when there is experience information about a node, the request is forwarded to that node, otherwise the best neighbor is selected.

The performance of the above mentioned request forwarding algorithms have been evaluated in terms of response time which is the number of hops a request has to traverse to find its required resource. In the set of experiments, the fairness of distribution of resources on nodes has also been taken into account: some nodes offer a large number of resources while others may offer just one [14]. Based on the degree of fairness, the experiments were excecuted in *balanced* and *unbalanced* resource distributions.
According to [14, 15, 13], (Experience + random) forwarding algorithm performs the best in terms of response time in both balanced and unbalanced distributions though it is expensive with respect to storage space. The reason for being expensive is that each node records the requests responded by other nodes. Best-neighbor forwarding algorithm works well only in unbalanced distribution due to ununiform distribution of resources on nodes. Random forwarding algorithm is the least efficient and the least expensive approach as it does not need storage space on nodes to maintain requests.

2.3.2 Distributed Grid Resource Discovery with Matchmakers

Centralized resource discovery mechanism does not scale well when the number of users increases in the grid. To overcome this problem, grid resource discovery mechanism has to be supplemented with peer-to-peer concept. Based on this idea, a distributed resource discovery method has been suggested in [25] in which the resource information is distributed among some local servers (matchmakers). Resource buyers/providers send their resource requests/offers to their local matchmakers. A matchmaker is responsible for providing available resource information which is stored locally.
According to the proposed framework in [4], the grid is considered as a set of distributed matchmakers each having the list of neighbors in a neighbor list. The neighbor lists are updated whenever a node joins/leaves the grid or a response on behalf of a matchmaker is received by a requesting matchmaker. A matchmaker either finds a match for a given resource request or forwards the request to a best neighbor matchmaker if no resource match is found locally. So, the matchmakers are cooperative for finding non local resource matches [25].
Request forwarding can have important effect on the resource discovery success rate.

As mentioned above, if a resource request can not be satisfied by a matchmaker, it is forwarded to another matchmaker who has a good reputation to the forwarding node. If a match(address of the responsive matchmaker) is found for a request, it is directly sent back to the matchmaker who initiated forwardig the request. Deciding to which neighbor a request message should be forwarded can be made upon the previous experience with the neighbor. In order to achieve this goal, each node maintains a repository called *experience-cache* which consist of neighbor address, number of matches found by a neighbor and the average response time taken by a neighbor to respond a request message. In addition, each matchmaker maintains the information about the request messages in its *request-cache* repository. This information include the request identifier, the matchmaker who forwarded the request, the number of times the request is sent from the requesting matchmaker and the request message time to live(TTL).
The goal of the requesting forwarding algorithm is to select the best neighbor, a matchmaker which has the highest probability to find match(es) and also has the minimum response time. The algorithm works as follows. After receiving a request message, if no experience-cache repository is provided with the node, a random matchmaker is selected to forward the message. Otherwise, the receiving matchmaker tries to find a neighbor with maximum number of responded requests and minimum response time. The request forwarding algorithm consideres the *maximum number of responded requests* as the highest priority to select the best matchmaker neighbor.
The proposed distributed grid resource discovery [25] is characterized by its request forwarding algorithm. Three scenarios may happen when a node receives a message. First, if the TTL of the message is expired, a discarded message is sent to the requesting matchmaker. the requesting matchmaker forwards the same message with respect to the number of neighbors in its neighbor list. If the matchmaker has only one neighbor, the message is sent to that neighbor again while the TTL is increased. Otherwise, the message is sent to a different neighbor provided that the number of times this message has been already forwarded is less than a predefined threshold. In the second scenario, the matchmaker receives the message which shows that a match has been found for the request. Thus, the responsive matchmaker is recorded in the matchmaker's neighbor list and the match result is sent back to the requesting matchmaker node. The third scenario happens when the node receives a forwarded request message, meaning that the previous matchmaker has not succeeded to find a match for that request. Thus, the receiving matchmaker tries to find a match locally. If the match can not be found, it decreases the message TTL and forwards the message to its best neighbor [25].

The proposed framework has been evaluated based on whether the matchmakers use the experience-cache repository for forwarding request messages or not. In case of using experience-cache, it has been shown that for varying number of matchmakers, the average TTL as well as the response time for each request is less than the case when the matchmakers do not use experience-cache. The reason for such an observation is that the matchmakers *learn* about their neighbors when no initial history is used for forwarding the requests. Similarly, the number of found matches, using the proposed request forwarding algorithm, is more than the case when the request messages are forwarded randomly [25].

2.3.3 Hybrid Resource Discovery in Ad Hoc Grids

The emergence of ad-hoc grids as a means of a computing resource sharing community is due to extension of grid technology to the ad hoc networks. The resource discovery is one of the challenging issues when deploying the ad hoc grids in resource sharing environments. Generally, the performance of a resource discovery mechanism is analyzed in terms of scalability, response time, bandwidth consumption and self-managing to changing environment. Considering these criteria, [18] proposes a *hybrid*(mixed) approach for resource discovery in ad hoc grids.
The principle idea for the proposed framework is based on the zone partitioning which has been introduced by the *zone routing protocol(ZRP)* for ad hoc networks [12, 11]. In ZRP, the network is divided into routing zones according to distances(in hops) between the nodes. In this hybrid routing protocol, if the source and destination nodes are in the same routing zone, routes are immediately available. However, when the source and destination nodes reside in different zones, a route discovery message is initiated to determine a route to the required destination.
The proposed resource discovery method takes the advantage of peer-to-peer mechanisms to find resources as the maintanance cost of such systems is low and they are self-organized. In this discovery scheme each node is assigned a zone, known as discovery zone, with a certain radius. The discovery zone consists of all the neighbor nodes whose distances from the node in the context must not exceed the radius. The radius of a zone, centered at a node, is defined to be a certain number of hops from that node.
The discovery mechanism in the proposed framework is composed of two discovery methods with regard to the resource location in the ad hoc grid. If the resources are to be found within a zone, each grid node multicasts its resource offer messages to the nodes inside the zone. By means of this mechanism, every node in the zone has a complete information about all its neighbors within that zone. If the required resources are not found within the discovery zone, the requesting node sends its message to the nodes whose distances from the requesting node equals the radius of the discovery zone. These nodes, known as peripheral nodes, are responsible for getting information from the nodes existing in the adjacent zones. The distance traveled by a message can be regulated using a parameter called *Forwarding Distance*. This parameter determines how many times a message must be forwarded to the next adjacent peripheral nodes. Figure 2.9 depicts the message routing process in ad hoc grid when a request message with $ForwardingDistance = 1$ and $ForwardingDistance = 2$ is forwarded to the peripheral nodes of the client's zone and the peripheral nodes of the neighboring zone respectively[18, 24].

There are three types of messages in the hybrid resource discovery mechanism. First, *Advertisement* messages that a node uses to send its resource offer messages to other nodes in the local zone. The resource provider controls sending its messages by means of a parameter called *Advertisement Period* which specifies the number of times that a message multicast through the discovery zone. In addition, each node knows how long the resource information is maintained if no *Advertisement* message is received from the respective resource provider. Second, the *query request* message that is sent by the client to the peripheral nodes to find the required resource which do not exist within the local zone. This type of message contains the *Forwarding Distance* parameter and

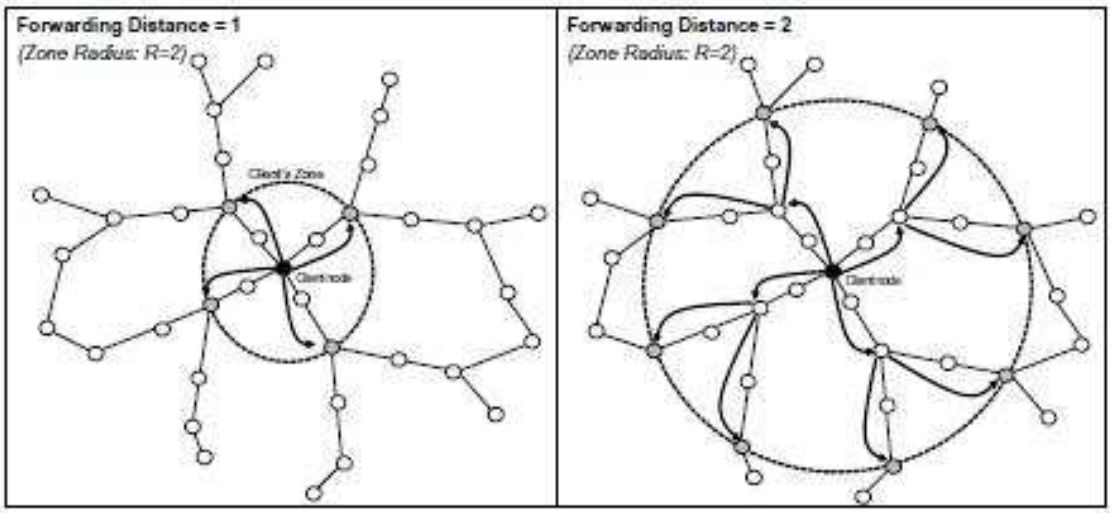


Figure 2.9: Request message forwarding with peripheral nodes in ad hoc grid[18]

the requested resource requirements such as the name and the quantity of the resource. Finally, the *query response* message by which a responsive peripheral node returns the matches with respect to the requested resource information.
The proposed hybrid approach for resource discovery is scalable in the sense that it reduces the bandwidth consumption between the nodes due to restricting the *Advertisement* message multicast to the client's zone. Furthermore, the query messages are forwarded by the peripheral nodes rather than using flooding scheme. The query response time in hybrid resource discovery method is less than the fully decentralized systems because each peripheral node has the complete information about all the nodes within its zone. As each node in the ad hoc grid advertises its resources periodically, no extra cost is paid for maintaining the resource information [18].

2.4 Conclusion

In this chapter, we discussed the related research on P2P overlay networks. The deployment of P2P systems depends on the application and the its required performance in terms of scalability, fault resiliency and the cost for maintaining the system. We also discussed about the structured P2P networks as good candidate for constructing large scale and robust Internet applications for resource sharing environments [5, 17]. One of the main challenges in P2P networks is discovering resources in a dynamic environment where resource providers and requesters join and leave the network arbitrarily. We investigated some resource discovery strategies in this chapter as well.

Resource discovery mechanism	**Architecture**	**Resource organization**	**Query resolution**
Multiple matchmakers	Within each organization, one node acts as a matchmaker.	A matchmaker maintains resource information of all nodes in the local organization.	Request forwarding is used for routing a query. A forwarding decision is made based on the previous experience with the neighbor matchmaker.
Hybrid	The network is divided into discovery zones based on distances between nodes. Each node has its own discovery zone.	Each node has complete information about all the neighbors within its discovery zone.	Queries are resolved by either: *Advertisement method* to discover nodes within the zone or *Query method* to discover nodes out of the zone.
Fully decentralized	Flat P2P overlay network where each organization has one or more peers.	Each peer provides access to one or multiple resources locally.	Queries are resolved by four request forwarding algorithms: random walk, learning, best neighbor, learning + best neighbor.

Table 2.3: Qualitative comparison of resource discovery systems

3 Adaptive Overlay Networks

Ad Hoc Grids are a type of computing networks where the availability of resources and tasks changes over the time. Distributing the tasks among the available resources in a balanced way is a challenging task because all the nodes have to receive equal utilization from the Grid resources.
In a decentralized environment, the processing loads and the network bandwidth is completely distributed among the system nodes. In addition, the decision making process, usually performed by the centralized controller in the client/server model, is now performed individually for each peer in the ad hoc grid network. Thus, the communication among the nodes is performed in a Peer To Peer(P2P) approach such that the system becomes scalable, self organized and fault tolerant. The scalability avoids single points of failure and bottlenecks. The self-managing property indicates the capability of the system to be adaptive to dynamic environmental conditions and the fault tolerance represents the system robustness in case of agents failures. In spite of system decentralization, we still want to get the benefits of the centralized resource allocation approach that is easy administration and implementation as well as good system throughput. As the resource allocator is responsible for finding the matches for the clients in the ad hoc grid, we use the term *matchmaker* instead of the resource allocator throughout the remaining of this thesis project.

In this chapter we investigate how an ad hoc grid is decentralized among the participating agents in the system. First, we present the algorithm by which the ad hoc grid is decentralized. Then, we describe how decentralization process is achieved when having two scenarios in the network. Finally, we discuss about how a system can be adaptive to the dynamic environment.

3.1 Algorithm Definition

The proposed algorithm for decentralized matchmaking is implemented on the Pastry underlying overlay system [23]. There are N agents in our experiments each one can play the role of a consumer, a producer or a matchmaker agent. A node, known as *resource producer* can offer a shared resource (disk space, CPU or network bandwidth). A node which requests for a desired resource to execute its job is called *resource consumer*. On the other hand, there are some other nodes in the overlay system that are responsible for resource allocation(finding the suitable resources for execution of the jobs). These nodes are known as the *resource allocators*($matchmakers$). It is assumed that the system has a maximum of M matchmakers each one has an identifier(id) by which a client node (resource producer or resource consumer) can contact with. The matchmaking is performed by using the Continuous Double Auction(CDA) mechanism

which has been proposed in [19]. CDA is a market-based matchmaking approach which introduces money and prices for coordination between resource consumers and producers [19]. Each node is granted a limited budget upon joining the ad hoc grid. The budget is used by a node to buy required resources or to sell its idle resources. In this project, the price indicates the matchmaker workload and is considered as the *Transaction Cost*(TCost). The TCost value represents the number of request/offer messages to be processed before processing the newly received request/offer message. As each request/offer message has its own TCost, this value for a matched (request,offer) pair is the average of their individual TCost values.
The consumer and producer nodes send their resource requests/offers to the matchmaker. The matchmaker finds the matches between producers and consumers by matching requests and offers which are stored in descending and ascending order respectively. When a matchmaker receives a request message, it searches all available resource offers and returns the best match considering request's constraints such as requested resource quantity, resource availability, task execution time and the price. In addition to these constraints, each request/offer message contains a Time To Live(TTL) which represents the validity period of the message and also the amount of available budget which increases or decreases according to buying/selling a node's resource. If no match is found, the request/offer remains in the matchmaker's request/offer buffer until its TTL expires or a match is found. Whenever the above mentioned requirements are compatible for each (request,offer) pair, a trade is executed immediately and the confirmations are sent to the trading agents [21, 19].

The system behaves dynamically in such a way that the matchmakers join or leave the system arbitrarily. The following are two main issues which are investigated regarding the self-organized dynamic ad hoc grid environment:

- **Matchmaker Attributes.** A node has to fulfill some criteria (conditions) in order to be recognized as a matchmaker node in the system.

- **Matchmaking Workload.** The workload can be considered as a good condition by which a client can be promoted as a matchmaker when the existing matchmaker(s) are overloaded.

As Algorithm 1 presents, the principle idea for having a self organized system with dynamic matchmakers is as follows:
Assuming there are at most $mmCount$ matchmakers in the overlay network, the identifier space is divided into $mmCount$ zones(segments) that are almost equal in size. Each matchmaker has an identifier($mmID$) by which a client node(producer or consumer) can communicate. The zone size,denoted as $zoneSize$, is obtained by dividing the total number of nodes in the system($nodeCount$) by the total number of the matchmakers($mmCount$). The zones are defined as $z_{zoneID} = [zoneStartNodeID, zoneEndNodeID]$ where $zoneStartNodeID$ and $zoneEndNodeID$ represent the identifiers of the first node and the last node in a zone with identifier $zoneID$ ($0 \leq zoneID \leq mmCount - 1$) respectively. Each zone is under responsibility of a matchmaker. A matchmaker node, belonging to zone with

identifier *zoneID*, is defined to be the immediate successor of the last node with identifier $zoneEndNodeID$ in a zone with identifier $z_{zoneID-1}$. In other words, a responsible matchmaker for a zone is always the first node member of the next zone in the overlay network. *zoneInfoArray* represents the zone information including *zoneStartNodeID*, *zoneEndNodeID* and *mmID* of a zone. Each node keeps this information with respect to *every zone* in the ad hoc grid environment. It is assumed that a matchmaker is not under responsibility of any other matchmaker in the system.

In order to segment the overlay system into several zones, the following assumptions are considered:

- At any given time instance, the system can have a predefined maximum of $mmCount$ matchmakers.

- When the nodes want to join the overlay network, they can be either a client node (producer or consumer) or a matchmaker node. In fact, we determine the role of each system node before its creation.

- Each joining node(producer, consumer or matchmaker) with identifier *nodeID* is aware of a zone it belongs to by knowing $zoneStartNodeID$ and $zoneEndNodeID$ corresponding to the first node and the last node of the respective appropriate zone.

- In the partitioned system, a consumer node may have its job executed on a given resource. In this case, that node sends a request message to the nearest active matchmaker(MM) which is responsible for the zone(s) containing the requesting node's identifier but in order to do so the client node needs to know $zone_{zoneID}$ and $mmID$ of the corresponding zone. Assuming the contacted matchmaker(MM) has the requested resource, it sends back a confirmation as a reply to the requesting consumer node.

Figure 3.1 illustrates a segmented overlay network with 64 nodes and 4 matchmakers ($nodeCount = 64$,$mmCount = 4$) where there are 16 nodes in each zone ($zoneSize = 16$). The matchmaker responsible for a zone is the successor of the last node in the respective zone. As can be seen in this figure, each zone and its associated matchmaker has been shown with a different color. In addition, the zones are numbered in the clock wise direction which are defined as follows according to Algorithm 1 for zone partitioning:

$z_0 = [0, 15], \quad z_1 = [16, 30], \quad z_2 = [32, 47], \quad z_3 = [48, 63]$

For example, $MM2$ with $matchmakerID = 32$, belonging to z_2, is responsible for the client nodes with $nodeIDs$ varying from 16 through 31 that are populated in z_1. It should be noted that $MM2$ is the first node appearing in z_2.

Algorithm 1 Ad Hoc Grid Segmentation

1: **int** $zoneSize = nodeCount/mmCount$;
2: **for** $i = 1$ to N **do**
3: $zoneStartNodeID = i * zoneSize$;
4: $zoneEndNodeID = ((i + 1) * zoneSize) - 1$;
5: **int** mmID = **successor**(zoneEndNodeID);
6: **if** (zoneStartNodeID $\leq nodeID \leq zoneEndNodeID$) **then**
7: $Node_{nodeID} \in zone_{zoneID}, zone_{zoneID} = [zoneStartNodeID, zoneEndNodeID] \wedge$ $matchmaker_{mmID} \in zone_{zoneID+1}$ /* $matchmaker_{mmID}$ is responsible for the consumer/producer nodes in $zone_{zoneID}$ */
8: **end if**
/* Each consumer/producer node keeps information about all the zones in the ad hoc grid */
9: $zoneInfoArray[zoneID] = \{zoneStartNodeID, zoneEndNodeID, mmID\}$;
10: $zoneStartNodeID = zoneEndNodeID + 1$;
11: **end for**

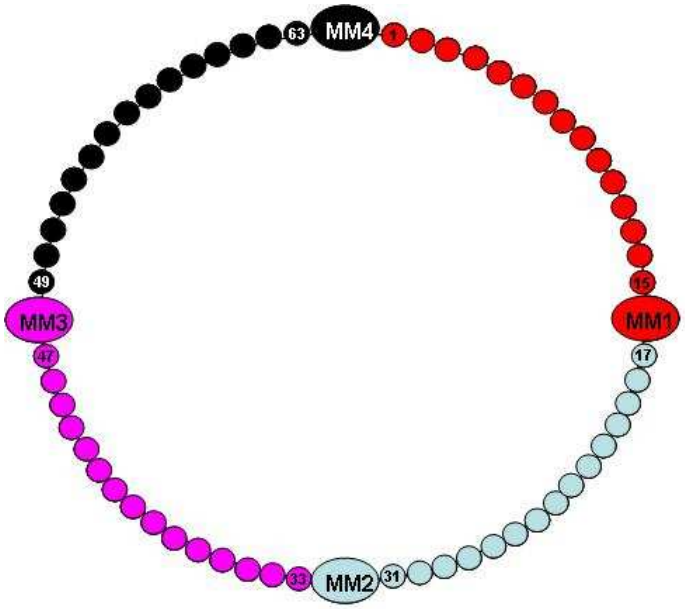


Figure 3.1: An overlay network with N=64 and M=4.

3.2 Overlay network status for connection between client and matchmaker

Having the partitioned overlay network populated with the clients and the responsible matchmaker in each zone, the producers and consumers should contact an appropriate matchmaker in order to send their resource requests and resource offers. However, a client has to be informed of the *matchmaker activity status* before sending its request/offer message to the matchmaker. In the following, we discuss about two network situations

Algorithm 2 Finding responsible matchmaker in fully/non-fully populated system

1: $\exists(Node_{nodeID} \in zone_{zoneID} \wedge matchmaker_{mmID} \in zone_{zoneID+1})$, $0 \leq zoneID \leq mmCount - 1$
boolean mmResponse = $Node_{nodeID}$.**mmActivityStatus**($matchmaker_{mmID}$) /* $Node_{nodeID}$ sends *are you active?* message to its own $matchmaker_{mmID}$ that is responsible for $zone_{zoneID}$. */
2: **if** ($mmResponse == yes$) **then**
3: $Node_{nodeID}$.**sendMessage**($matchmaker_{mmID}$) /* $Node_{nodeID}$ sends its request/offer messages to $matchmaker_{mmID}$ */
4: **else**
5: zoneID++
6: **while** ($zoneID < mmCount$) **do**
7: **int** $mmID'$=$Node_{nodeID}$.**find_zoneResponsibleMatchmaker**(zoneID)/* $Node_{nodeID}$ finds the nearest $matchmaker_{mmID'}$ */
8: **boolean** mmResponse = $Node_{nodeID}$.**mmActivityStatus**($matchmaker_{mmID'}$)
9: **if** ($mmResponse == yes$) **then**
10: $Node_{nodeID}$.**sendMessage**($matchmaker_{mmID'}$)
11: **else**
12: zoneID++
13: **end if**
14: **end while**
15: **end if**

find_zoneResponsibleMatchmaker(zoneID)
String zoneInfo=zoneInfoArray(zoneID)
int mmID = zoneInfo[matchmakerID]
return mmID

where a client meets its responsible matchmaker in a segmented overlay network.

3.2.1 Fully populated overlay network

The overlay network is defined to be fully populated when all the matchmakers are active(alive) and perform resource allocation process for their clients in the respective zones. In this case a producer/consumer node sends "*are you active*" message to its own responsible matchmaker. As the system is fully populated, the matchmaker sends "*yes*" reply message to the producer/consumer node indicating that the matchmaker can perform matchmaking process. lines 1-3 of Algorithm 2 present the procedure by which a consumer/producer node searches its responsible matchmaker in a fully populated system according to what was mentioned above. $Node_{nodeID}$ and $matchmaker_{mmID}$ denote a consumer/producer node and a matchmaker node with identifiers *nodeID* and *mmID* respectively. *mmResponse* indicates the reply of the matchmaker about its matchmaking activity status. *mmActivityStatus* works by finding an active matchmaker. The consumer/producer node invokes *sendMessage* to send its request/offer messages after finding its own responsible matchmaker node.

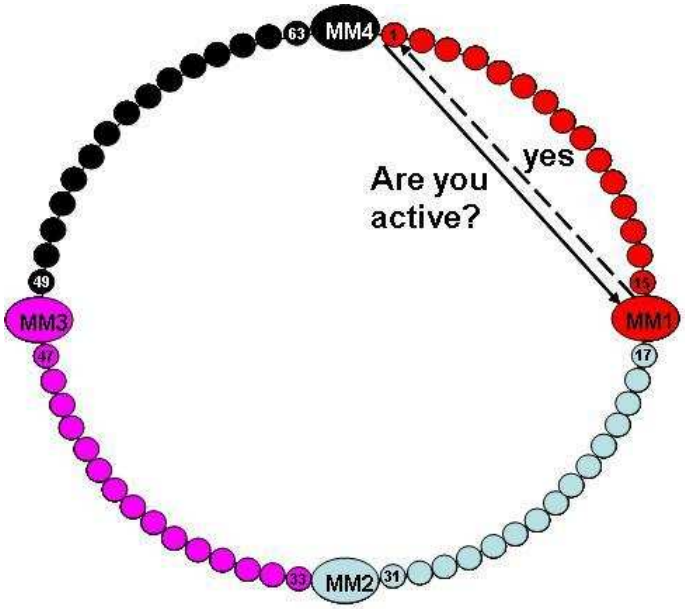


Figure 3.2: Connection between client and matchmaker in a fully populated system.

Figure 3.2 depicts a fully populated system where MM1 notifies a requesting client in z_0 of its activity mode. Regarding the network communication overhead, the ideal case for a producer/consumer node is to send its request/offer message to the matchmaker that is responsible for the zone containing the identifier of the requesting node.

3.2.2 Non fully populated overlay network

The overlay network is defined to be non fully populated (sparsely populated) when at least one matchmaker either exists but is not performing matchmaking process or it does not exist on the overlay network. As the system must be self organized in case of matchmaker(s) failure, the client nodes which were used to be served by their own matchmaker, are now managed by the successor of the failed matchmaker.
In the non fully populated system, the use of successor of the end of a zone guarantees that a client eventually meets the responsible matchmaker. lines 4-14 of Algorithm 2 describe the procedure that a consumer/producer node uses to find the nearest matchmaker when the system is sparsely populated. when $Node_{nodeID}$ executes $find\_zoneResponsibleMatchmaker$, it searches the overlay network for a nearest active matchmaker to contact with by using *zoneID* of a zone. This responsible matchmaker has to be the successor of $Node_{nodeID}$'s inactive matchmaker in the clockwise direction on the overlay network.
Figure 3.3 shows the steps in a non fully populated system in which a client node tries to meet its nearest matchmaker. According to this figure, *MM1* is not active and the clients in z_0 and z_1 are under responsibility of *MM2*(1). As a producer/consumer node in z_0 does not know whether *MM1* is active or not, it sends "*are you active*" message to the matchmaker(2). After receiving "*No*" reply message from *MM1*, the client then

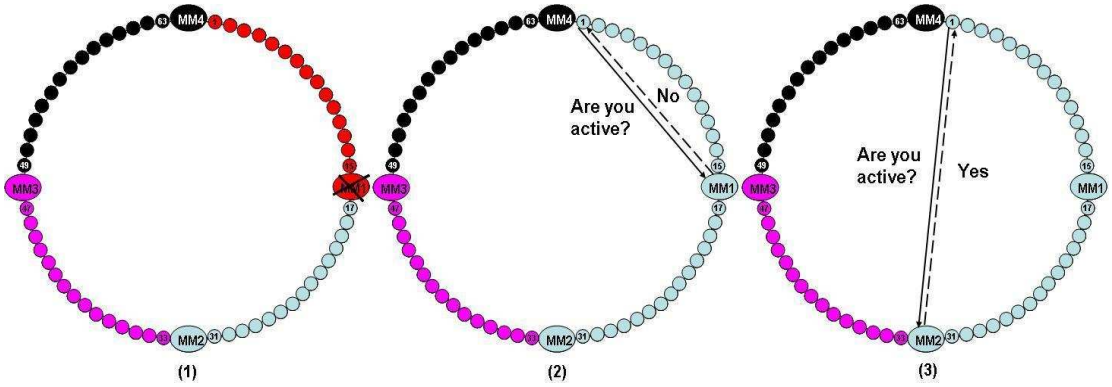


Figure 3.3: Connection between client and matchmaker in a non-fully populated system.

contacts the next matchmaker and when it finds that *MM2* is alive, it initiates sending the request/offer messages to *MM2*(3).

3.3 Overlay network decentralization with multiple matchmakers

In this section, we define a mechanism that dynamically segment and merge back the overlay network segments by introducing/removing matchmaker(s) based on the workload of the existing matchmaker(s). This mechanism enables the overlay network to be dynamically adaptive from a centralized to completely decentralized form and back to the completely centralized form. The dynamic adaptation is achieved in such a way that the total workload in the system is efficiently distributed among the existing matchmakers.
Here, the TCost *upper threshold value* is used as an indicator for dynamic segmentation and desegmentation of the overlay network. A matchmaker promotes a node as a matchmaker or demotes itself back to normal node when its average TCost is above/below the matchmaker's upper threshold value. In the following, we describe how the resource allocation is decentralized so that the matchmakers can route messages to different segments of the overlay network.

3.3.1 Matchmaker promotion: segmentation

For segmenting the overlay network, we assume that a few matchmakers are active (performing matchmaking) in the overlay network. Each active matchmaker is supposed to maintain the information about the matchmaking status of its immediate predecessor and successor matchmaker nodes. When the workload of a matchmaker is beyond the *TCost upper threshold value*, it promotes its *predecessor* matchmaker node as a new matchmaker. The request/offer messages corresponding to the client nodes, belonging to predecessor matchmaker node, that were used to be processed by the overloaded matchmaker, are then processed by the promoted matchmaker node. Thus, the new matchmaker shares the workload of its overloaded counterpart. Algorithm 3 describes the match-

Algorithm 3 Matchmaker Promotion in Ad Hoc Grid

1: /* $matchmaker_{mmID}$ is overloaded */
2: **if** ($matchmaker_{mmID}.TCost \geq TCostUpperThreshold$) **then**
3: $matchmaker_{mmID}$.**MM_promotion**(mmID-1) /* $matchmaker_{mmID}$ promotes $matchmaker_{mmID-1}$ as a new matchmaker; ad hoc grid is segmented */
4: **end if**
5: $\forall\ Node_{nodeID} \in zone_{zoneID'}$, $\exists\ matchmaker_{mmID} \in zone_{zoneID+1}$
int $nodeID = matchmaker_{mmID}$.**getMessageSenderID**() /*$matchmaker_{mmID}$ determines the sender of a received request/offer message */
6: **while** ($0 \leq zoneID^{'} \leq zoneID - 1$) **do**
7: $matchmaker_{mmID}$.**update_matchmaker**(mmID-1,nodeID) /*$matchmaker_{mmID}$ informs a client node with identifier $nodeID$ of its new matchmaker with identifier $mmID - 1$ */
8: **end while**

making promotion process in the ad hoc grid where the system is non-fully populated. When the workload of $matchmaker_{mmID}$(TCost) is larger than the maximum tolerable workload ($TCostUpperThreshold$), $matchmaker_{mmID}$ executes $MM\_promotion$ to promote its predecessor matchmaker as a new matchmaker. $MM\_promotion$ indicates that the client nodes, belonging to $matchmaker_{mmID-1}$, that were under responsibility of $matchmaker_{mmID}$ are now managed by $matchmaker_{mmID-1}$. Having a segmented ad hoc grid, $matchmaker_{mmID}$ checks the sender node of a received message. If the associated $nodeID$ belongs to a zone other than the zone for which $matchmaker_{mmID}$ is responsible($zone_{zoneID}$), $update\_matchmaker$ is executed by $matchmaker_{mmID}$ to inform $Node_{nodeID}$ of its new matchmaker. Thus, $Node_{nodeID}$ will send its next request/offer messages to the predecessor of $matchmaker_{mmID}$ ($matchmaker_{mmID-1}$).

Figure 3.4 shows the zone partitioning process in an overlay network when there are only two active matchmakers (MM3 and MM4). According to this figure, MM3 asks about the matchmaking status of its predecessor matchmaker(1). Assuming MM2 is not performing matchmaking, it is promoted by MM3. In this case, the client nodes in z_0 and z_1 which were under responsibility of MM3 are then managed by MM2(2). A consumer/producer node which is not aware of the matchmaking promotion process, sends its request/offer to the overloaded matchmaker. The message is processed by MM3. However, MM3 sends "*MM2 is your matchmaker*" notification message to the requesting client node so that it sends its next request/offer messages directly to MM2(3).

3.3.2 Matchmaker demotion: desegmentation

Similar to the ad hoc grid segmentation, the *TCost threshold value* is used to trigger merging back the segments dynamically. Desegmentation process is fulfilled by removing the underloaded matchmaker(s) whose TCost are below the *lower threshold value*. When a matchmaker becomes underloaded, it first demotes itself and then updates the next successor matchmaker about its matchmaking status. In this case, the request/offer messages corresponding to the client nodes that were used to be processed by the demoted(underloaded) matchmaker, are then processed by the successor of the

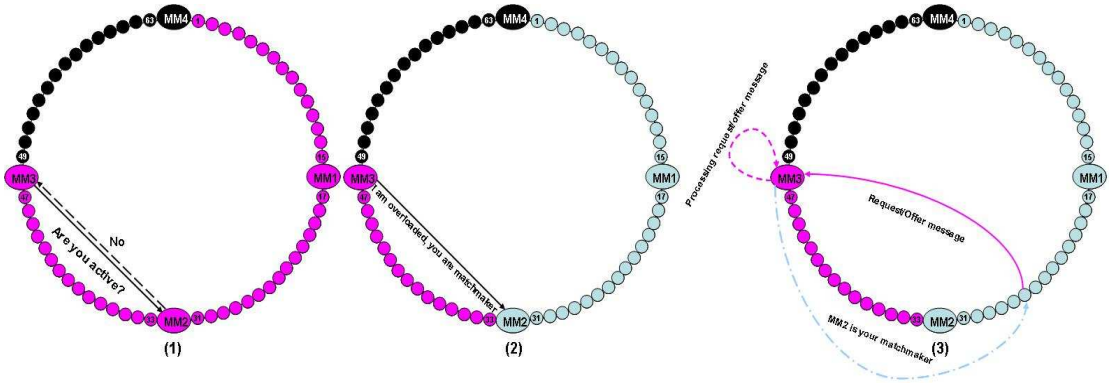


Figure 3.4: Ad hoc grid segmentation process when MM3 promotes MM2.

demoted matchmaker. Thus, the underloaded matchmaker forwards the received request/offer messages to its successor matchmaker node without processing them. The required steps for matchmaking demotion in a non-fully populated system are presented in Algorithm 4. When the workload of $matchmaker_{mmID}$(TCost) drops below the minimum workload($TCostLowerThreshold$), $matchmaker_{mmID}$ executes $MM\_demotion$ and $demotion\_update$ to demote itself as normal node and updates the successor matchmaker about its activity status respectively. $inactiveMode$ indicates that $matchmaker_{mmID}$ is not performing matchmaking process any more. $demotion\_update$ denotes that $matchmaker_{mmID}$ does not perform matchmaking and $matchmaker_{mmID+1}$ is now responsible for the consumer/producer nodes that were used to be under responsibility of $matchmaker_{mmID}$. Whenever the underloaded matchmaker receives a request/offer message from $Node_{nodeID}$, $forward\_message$ is invoked by $matchmaker_{mmID}$ to forward the message to its successor($matchmaker_{mmID+1}$). Moreover, $matchmaker_{mmID}$ executes $update\_matchmaker$ in order to inform $Node_{nodeID}$ of its new matchmaker so that the next request/offer messages are sent directly to $matchmaker_{mmID+1}$.

Figure 3.5 illustrates the ad hoc grid desegmentation process when all the existing matchmakers are active(performing matchmaking). Assuming MM3 is underloaded, it informs its successor (MM4) of its matchmaking status. So, the clients nodes in z_2 which were under responsibility of MM3 are then managed by MM4. In this case, MM3 reroutes each received resource request/offer to MM4. At the same time, MM3 sends "*MM4 is your matchmaker*" message to the respective consumer/producer nodes so that the next request/offer messages are directly sent to MM4.

3.4 Conclusion

This chapter talked about the adaptive matchmaking approach with multiple matchmakers. We proposed a resource allocation mechanism that can self- organize itself in the ad hoc grid environment. The decentralization is performed by dynamically segmenting/desegmenting the ad hoc grid environment according to the workload of the matchmaker. The matchmaker overload/underload threshold value is used as a criterion for promoting/demoting the matchmaker(s) in order to balance the workload of the ex-

Algorithm 4 Matchmaker Demotion in Ad Hoc Grid

1: /* $matchmaker_{mmID}$ is underloaded */
2: **if** ($matchmaker_{mmID}.TCost \leq TCostLowerThreshold$) **then**
3: **boolean** inactiveMode=$matchmaker_{mmID}$.**MM_demotion**(mmID) /* $matchmaker_{mmID}$ does not perform matchmaking by demoting itself as normal node, that is inactiveMode=false */
4: $matchmaker_{mmID}$.**demotion_update**(mmID+1,inactiveMode) /* $matchmaker_{mmID}$ updates the successor matchmaker with identifier $mmID + 1$ about its changing activity status; ad hoc grid is desegmented */
5: **end if**
6: $\forall\ Node_{nodeID} \in zone_{zoneID'}$, $\exists\ matchmaker_{mmID} \in zone_{zoneID+1}$ **int** $nodeID = matchmaker_{mmID}$.**getMessageSenderID**() /* $matchmaker_{mmID}$ determines the sender of a received request/offer message */
7: **while** ($0 \leq zoneID^{'} \leq zoneID$) **do**
8: $matchmaker_{mmID}$.**forward_message**($mmID+1, nodeID$) /* $matchmaker_{mmID}$ forwards a received request/offer message from $Node_{nodeID}$ to $matchmaker_{mmID+1}$ */
9: $matchmaker_{mmID}$.**update_matchmaker**(mmID+1,nodeID) /* $matchmaker_{mmID}$ informs a client node with identifier $nodeID$ of its new matchmaker with identifier $mmID + 1$ */
10: **end while**

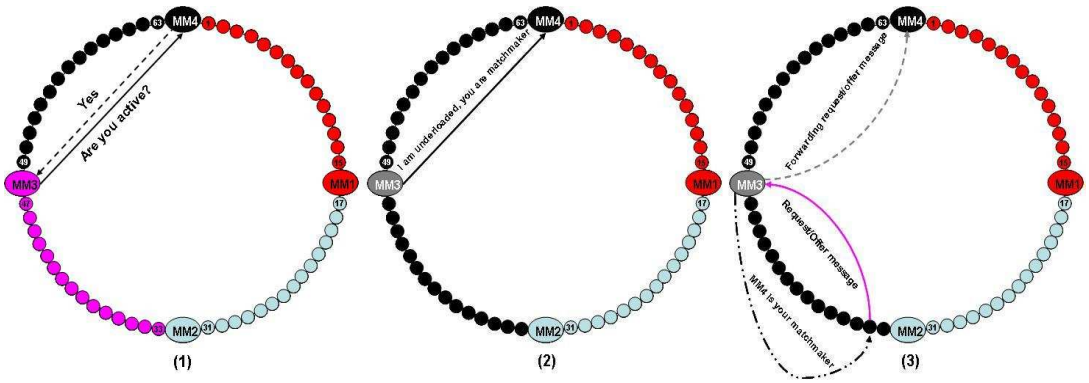


Figure 3.5: Ad hoc grid desegmentation process when MM3 is demoted.

isting matchmakers in the system. The next chapter will discuss about the results of the experiments conducted in a global testbed environment.

4 Experiments and Results

This chapter discusses the results of the experiments in which we investigated the performance of decentralized adaptive matchmaking mechanism for balanced network structure. The experiments were first executed to determine a matchmaker overload threshold. This threshold was then applied to segment the ad hoc grid into multiple segments by introducing new matchmakers for the segments. We set up evaluation criteria to investigate the adaptive matchmaking mechanism for balanced network condition.
In section 4.1, we discuss the setup of experiment with respect to the utilized testbed and the database for our implementation. Section 4.2 presents different criteria to evaluate the proposed solution for resource allocation process in 1 matchmaker system and 5 matchmakers system. Finally, section 4.3 summarizes the main finding of the experiments and presents the analysis and conclusion of this chapter.

4.1 Experimental Setup

The experimental platform was made using *PlanetLab* testbed to conduct our experiments in a distributed environment. PlanetLab is a global research network covering large geographical area. The PlanetLab has been used by academic institutions and industrial research labs since 5 years ago. We selected PlanetLab as a testing platform due to its capability to support running an application on a large number of nodes [4]. Request/offer messages were generated randomly and within a certain periodic intervals by the client nodes. The matchmakers, which were set up on machines within the university local area network, performs matchmaking process as soon as a request is received. In this case, the request message is stored at a position in the corresponding buffer with respect to its price value. Then the matchmaker compares the first request in the *consumer buffer* against all the offers in the *producer buffer*. Similarly, the matchmaker processes the remaining request messages in the descending order. A request remains in the matchmaker's consumer buffer until its TTL(Time To Live) expires or a match is found. When a (request,offer) pair is matched, the individual messages are stored in *consumer/producer matched buffers*, otherwise they are stored in unmatched ones. It should be noted that the expired messages are removed from consumer/producer buffer before the matchmaker starts matching process.

We run our project in the PlanetLab environment with varying number of nodes from 133 to 4000 in order to increase or decrease the workload of the matchmaker(s)(multiple Pastry nodes were running on each PlanetLab node). The number of matchmakers was varied from 1 to 5. As the whole project was implemented in Java,

we had to install Java Virtual Machine(JVM) on any connected remote workstation. So, we copied a tar archive, containing Java installation files, to the desired nodes in PlanetLab. After unpacking the archive, we were able to run our project on the corresponding nodes concurrently. For running our project, FreePastry-2.0_04 and JVM 1.6 were utilized as the underlying platform and the runtime environment respectively.

We setup database for storing and analyzing of all experimental results. The database was installed and configured on a certain number of matchmakers using *MySQL*. MySQL database is an open source relational database management system (RDBMS) that uses Structured Query Language (SQL) for performing operations such as inserting, accessing, removing, processing and updating data in a database. Our database stores the information including IP addresses of the clients, TTL, matchmaker workload(TCost), number of received request/offer messages and response time in different tables. These information can be easily retrieved and analyzed by using *SQL* query statements.

4.2 Evaluation Criteria

To evaluate decentralized matchmaking performance under balanced network condition, the following evaluation criteria are defined:

- **Matchmaking Efficiency**: The matchmaking efficiency is measured in terms of producer and consumer utilization. Consumer utilization is the percentage of tasks that are allocated to available resources and in the same way producer utilization is the percentage of the available resources that are used by allocated tasks. In general, the matchmaking efficiency is determined as ($\sum$matched message/ $\sum$message) $*$ 100.

- **Matchmaker Workload**: The matchmaker workload is represented as the Transaction Cost (TCost). The Tcost value indicates the number of messages to be processed by the matchmaker before processing the current received request/offer message(s). As each request/offer message has its own TCost, the TCost value for a matched (request,offer) pair is the average of the corresponding TCost values for the request and offer messages. The TCost value is calculated for each request/offer message as soon as it is received and inserted in a buffer at its proper position. In our project, we define lower and upper thresholds for the workload of the matchmaker (TCost). As the average TCost is calculated by a matchmaker periodically, the ad hoc grid can be segmented/desegmented whenever the average TCost value is above/below the matchmaker's upper/lower thresholds.

- **Matchmaker Response Time**: The matchmaking response time is calculated from the time a request/offer message is received by the matchmaker to the time the matchmaker makes a decision for the massage. The decision is made upon whether a match is found for the message or its TTL expires. The matchmaking response time is defined in terms of consumer response time and producer response time. The consumer/producer response time is the time for a consumer/producer

to find an appropriate resource/task. As mentioned above, we have also considered the response time for the unmatched messages in our experiments.

4.3 Experimental Results Analysis

Experiments were performed to investigate the matchmaker throughput in terms of matchmaking efficiency, workload(TCost) and response time. During our experiments, we considered CPU time as the resource in the system. A request/offer is sent to the matchmaker whenever a consumer/producer needs/has some CPU time. The experiments were executed in a balanced task-resource condition which means that the number of consumers and producers were almost the same. That is, at any given time instance, the resources and tasks have more or less equal increasing/decreasing rate. The time unit for all the executed experiments is in minutes.
TTL(Time To Live), message frequency and the job execution time are the parameters that significantly affect the ad hoc grid performance. These parameters are randomly generated from a predefined range. The TTL of a request/offer message was set to 10000 milliseconds. The message frequency is calculated with respect to a predefined sleep time. The sleep time determines the time interval after which a message is sent. The sleep time was set to 10000 milliseconds. so, each client node sends 6 messages per minute. All the jobs have a execution time period through which they have to acquire resources. The minimum job execution time was fixed to 500 milliseconds and the maximum job execution time was fixed to 4000 milliseconds. As the TTL specifies a message life time, it has to be larger than maximum job execution time.

The first set of experiments was executed to determine the matchmaker overload threshold for one matchmaker. The overload threshold of a matchmaker represents its TCost upper threshold value. This threshold was applied to the ad hoc grid environment with more than one matchmaker in the second set of experiments. The TCost upper and lower thresholds were set to 20 and 1 respectively. These thresholds are used to dynamically segment/desegment the ad hoc grid by introducing/removing the matchmaker(s).

4.3.1 One Matchmaker

Figure 4.1 shows the matchmaker TCost when we increase the workload of the matchmaker gradually. From this figure, we can figure out that the TCost increases at a higher rate with increasing workload of the matchmaker. The decrease of the matchmaker workload (TCost) at the end of the experiment can be explained by the departure of the client nodes from the ad hoc grid environment as each node has a certain life time during the experiment.
Figures 4.2 depicts the matchmaking efficiency with increasing workload of the matchmaker. *cUtil* and *pUtil* represent the consumer utlization and producer utilization respectively. *Mming* represents the overall matchmaking utilization in terms of all matched (request,offer) pairs. It is observed that at the start of the experiment, the utlization of both requests and offers is low because the consumer and the producer buffers are not

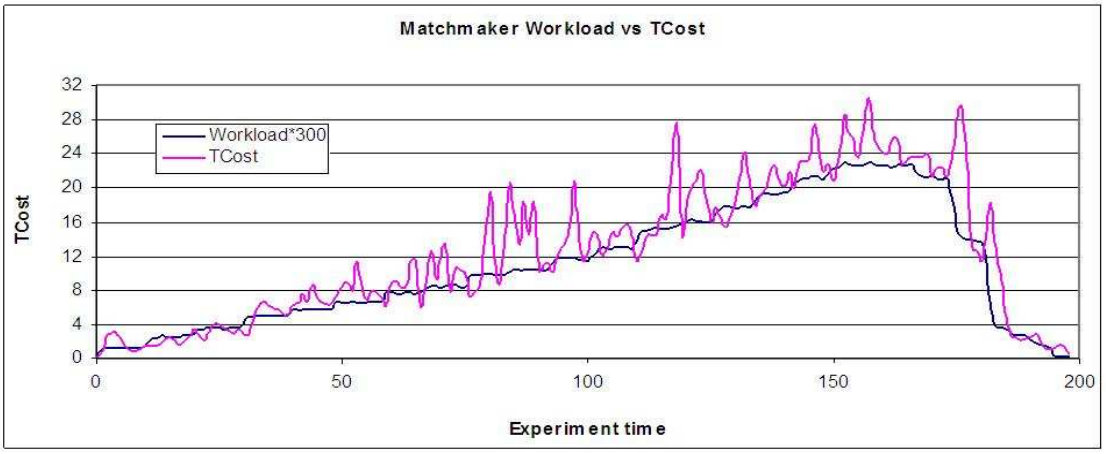


Figure 4.1: Matchmaking transaction cost for ad hoc grid with one matchmaker.

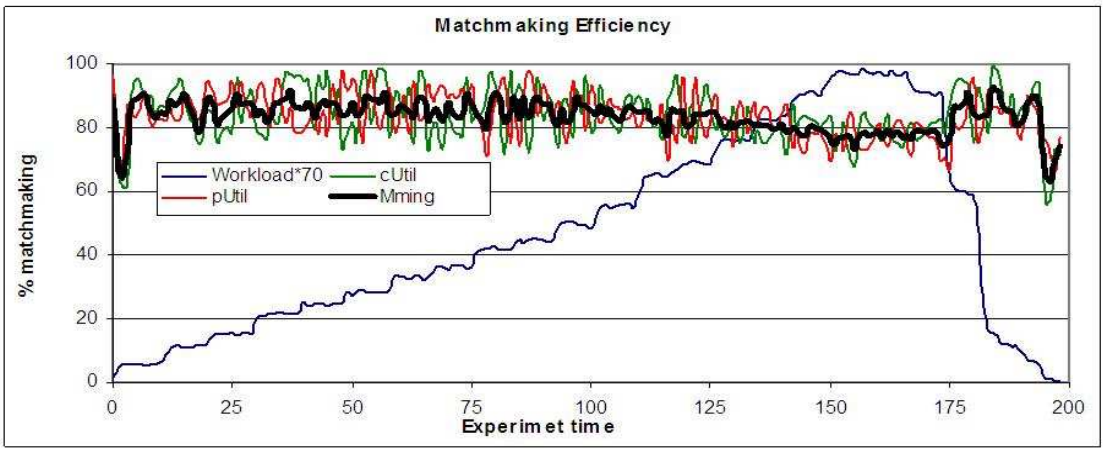


Figure 4.2: Matchmaking efficiency for ad hoc grid with one matchmaker.

sufficiently populated and that can not be matched. As the workload increases, more requests/offers are submitted to the matchmaker which results in a higher utilization rate.

Figure 4.3 depicts the matchmaker response time for request/offer messages with increasing workload of the matchmaker. *Matched* indicates the response time for matched messages whereas *All* indicates the response time for both matched and unmatched messages. The response time is high at the start of the experiment. The reason is due to initial low number of requests and offers in the corresponding matchmaker buffers which results in longer waiting time for a request/offer to find a match. As the matchmaker receives more messages, the likelihood of finding a match increases, resulting in a lower response time. From Figure 4.3, it is obvious that the response time for *All* messages is larger than the response time for *Matched* messages due to larger response time value with respect to unmatched messages. Regarding Figures 4.2 and 4.3, we can see that the matchmaking efficiency decreases and the response time increases at a higher rate

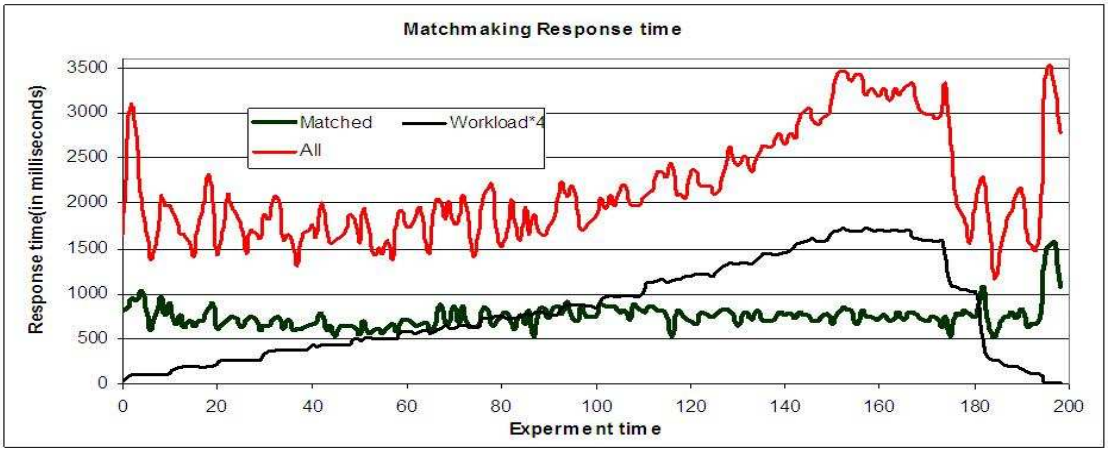


Figure 4.3: Matchmaking response time for ad hoc grid with one matchmaker.

respectively. In this case, it can be concluded that the matchmaker has reached its *TCost upper threshold* with the given message frequency. At this point, the consumer/producer nodes have to pay a higher price (TCost) in order to get the benefit from the trade. consumer/producer nodes have to pay With respect to Figure 4.1, it is observed that the upper threshold is 20 for 145 minutes. The TCost upper threshold indicates that a single matchmaker can not process all the received request/offer messages. Furthermore, this upper threshold is used to introduce a new matchmaker to share the workload of the first matchmaker. The experimental results for more than one matchmaker, using TCost upper threshold are presented in the next section.

4.3.2 Multiple Adaptive Matchmakers

In this section, the performance of the ad hoc grid environment with multiple matchmakers is investigated. As the second set of experiments are performed with more than one matchmaker, the average value of the matchmaking TCost, efficiency and response time are represented . The associated experiments were executed with varying number of nodes in the ad hoc grid. All these experiments started with one matchmaker. The workload of the matchmaker(s) was managed in such a way that the matchmakers are promoted when the workload increases and demoted when the workload decreases. The workload of the first matchmaker was increased so that the matchmaker workload went beyond the TCost *upper threshold* value and a predecessor inactive matchmaker was promoted as a new matchmaker. The promoted matchmaker serves the request/offer messages of the consumer/producer nodes which were previously served by the overloaded matchmaker. Whenever the TCost value was below a matchmaker's *lower threshold*, then the underloaded matchmaker demoted itself as a normal node and updated its succussor matchmaker about its matchmaking status. The request/offer messages of the consumer/producer nodes which were previously processed by the underloded matchmaker are then processed by the successor of the demoted matchmaker.

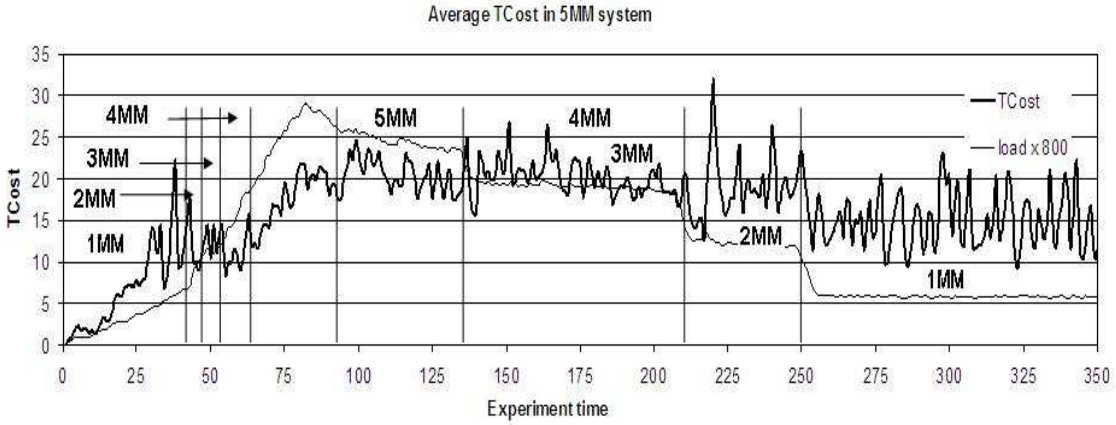


Figure 4.4: Average transaction cost for ad hoc grid with multiple matchmakers.

Figure 4.4 depicts the TCost variation of multiple adaptive matchmakers. As the maximum workload of the system with multiple matchmakers is around 3 times the maximum workload of a single matchmaker (24000 messages per minute for multiple matchmakers vs 7200 messages per minute for one matchmaker), the TCost *upper threshold value* in the former case is reached in less time than the corresponding time in the latter one. The TCost value increases with increasing matchmaker workload. When the first matchmaker reaches the TCost *upper threshold value*, then its predecessor matchmaker is promoted to take the responsibility of the consumer/producer nodes which were previously under responsibility of the overloaded matchmaker. Thus, the ad hoc grid is segmented and the average TCost value decreases below the upper threshold of one matchmaker by promoting the second matchmaker. Similarly, when second matchmaker is overloaded then its predecessor matchmaker(third matchmaker) is promoted in the ad hoc grid and so on. This phenomenon is reversed when a matchmaker is demoted. Whenever a matchmaker workload decreases below the TCost *lower threshold value*, then the underloaded matchmaker is demoted and the ad hoc grid segments are merged back. Again, the average TCost remains under the TCost *upper threshold* for all the existing matchmakers in the ad hoc grid at any given time instance. As we mentioned above, the TCost upper threshold was set to 20 and the TCost lower threshold was set to 1. Temporary fluctuations in the TCost refelcts sudden change in the workload of the matchmaker(s) or in the number of matchmakers.

Figure 4.5 depicts the average response time variation with multiple matchmakers. Similar to the matchmaking average TCost behavior(Figure 4.4), the response time remains stable when the number of matchmakers increases in the ad hoc grid. According to Figure 4.5, the average response time for consumers/producers is around 3 seconds which is less than the predefined TTL. The temporary variation of the matchmaking average response time (Figure 4.5) is due to change in workload of matchmakers or in the number of matchmakers.

Figure 4.6 depicts the request/offer utilization with multiple adaptive matchmakers. As the nework is balanced, the requests and offers have similar utilization. we can observe

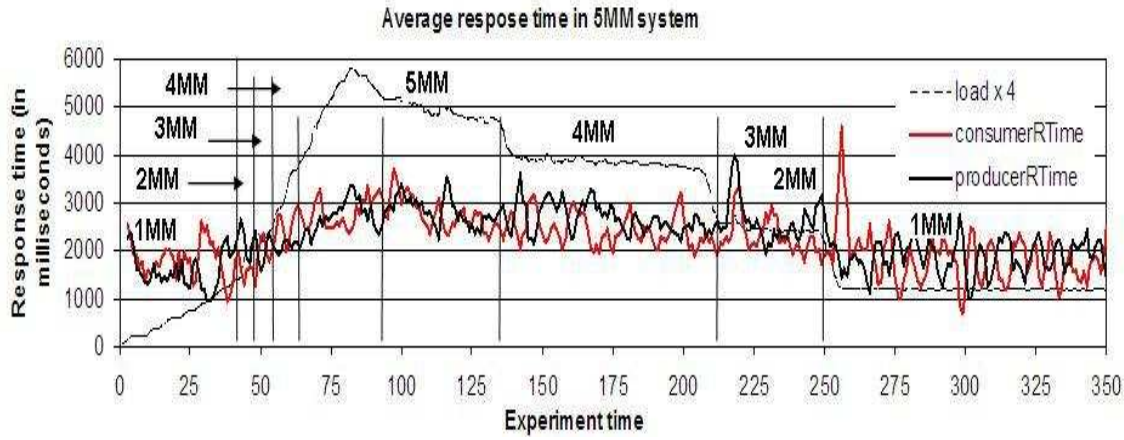


Figure 4.5: Average response time for ad hoc grid with multiple matchmakers.

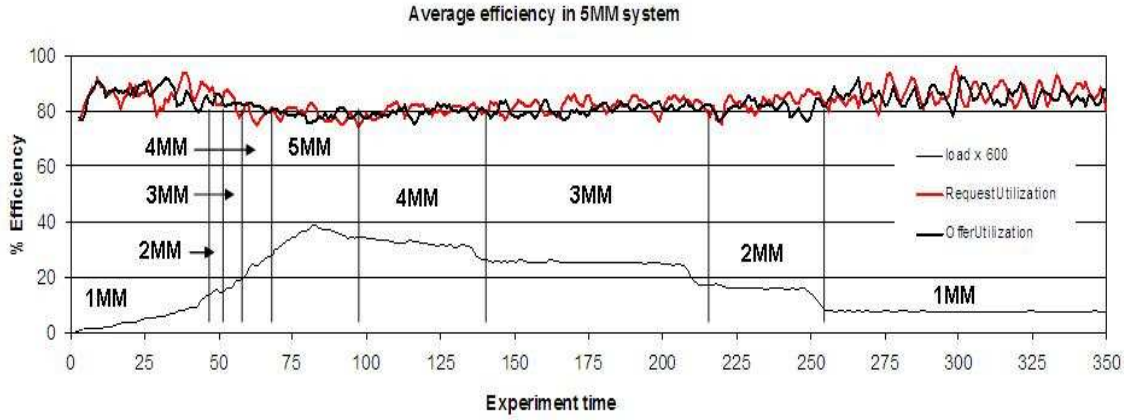


Figure 4.6: Average efficiency for ad hoc grid with multiple matchmakers.

that the matchmaking efficiency in terms of consumers and producers is stable(around 80%) as compared to approximately the same matchmaking efficiency for a single matchmaker(Figure 4.2).

From the above experiments, we can observe that the matchmaking performance of the ad hoc grid is stabalized irrespective of the number of processed request/offer messages by the existing matchmaker(s)in the system. This way, the matchmaking TCost, utilization and response time remain in the acceptable domain in spite of some temporary variations. It is also interesting to observe that the average value for TCost, efficiency and response time for a system with multiple adaptive matchmakers is close to the corresponding values for a system with one matchmaker.

4.4 Conclusions

In this chapter, we analyzed the proposed decentralized adaptive matchmaking algorithm using PlanetLab paltform as a distributed computing test-bed for our experiments. Matchmaking TCost, utilization and response time have been investigated for the balanced network condition. With respect to the dynamic nature of the ad hoc grid environment, the system achieves self-organization by promoting/demoting matchmaker(s) according to the workload(TCost) of the present matchmaker(s). The TCost *upper* and *lower threshold values* of matchmaker workload were determined in such a way that the ad hoc grid is dynamically segmented by promoting the inactive matchmaker(s) as new matchmakers and desegmented by demoting the underloaded matchmaker(s) as normal nodes. As a result, the matchmaking performance remains stable independent of varying workload of the ad hoc grid environment.

5 Conclusion

5.1 Summary

In this thesis project, we proposed a dynamic self-managing P2P overly network to dynamically segment and desegment the ad hoc grid. The workload of the matchmaker was used as a basic criterion so that the system can self-manage with respect to changing environment in terms of matchmakers. The upper and lower threshold values of the matchmaker workload (TCost) were specified for balanced network condition. These values were applied to dynamically segment the ad hoc grid by promoting the new matchmaker(s) and combining the segments together by demoting the matchmaker(s) as normal nodes in the ad hoc grid. In this adaptive overlay network, each segment has a matchmaker such that the clients(consumers and producers) can communicate with each other in a local neighborhood and matchmaking process in different segments is performed concurrently. By means of the proposed decentralized matchmaking approach, we are still able to get the benefits of centralized matchmaking such as easy administration and implementation. The reason is that the segments of the network can be centrally controlled within themselves via a central matchmaker and the matchmakers can route messages to different segments of the ad hoc grid in a P2P manner. The proposed decentralized matchmaking algorithm was implemented on top of Pastry as a structured P2P overlay network and was tested on PlanetLab as a geographically distributed testbed. Experimental results indicate that the proposed framework is scalable because the matchmaking efficiency is preserved irrespective of the number of messages that are sent and the system adapts according to the changing circumstances of the network.

5.2 Future Work

In future research we plan to improve our decentralized matchmaking approach as follows:

- In current implementation, a consumer/producer node is satisfied by any match result provided by the local matchmaker node regardless of the freshness of the resource information. We are going to expand the system in such a way that at any given time instance, the current matchmakers are up to date. To achieve this goal, the matchmakers should exchange information in an appropriate manner. For instance, whenever a matchmaker receives an offer message from a resource producer, it immediately sends that new information to all the other matchmakers.
- In the proposed matchmaking algorithm, a node is nominated to behave as a matchmaker from a set of predefined nodes as candidate matchmakers. We plan

to relax this assumption such that every node in the ad hoc grid can play the role of either a resource producer/consumer or a matchmaker. For example, if a matchmaker is underloaded, it promotes its next immediate normal node as a new matchmaker to be responsible for the segment that the demoted matchmaker was responsible for. Moreover, the hardware parameters (such as CPU, memory and disk) of the nodes will be taken into account in order to determine which node can fulfill the function of the matchmaker at any given point in time.

- In current implementation, we have tried to decentralize the matchmaking process in the sense that the current matchmaker(s) in the system share the total workload of the system. In other words, the workload of the ad hoc grid is *balanced* among the existing matchmaker(s). In future work, we also plan to investigate the *upper* and *lower values* of matchmaker workload threshold (TCost) so that all the matchmakers share the same workload. This idea results in a pure self-organized ad hoc grid that can completely adapt itself with respect to changes in the ad hoc grid environment.

Bibliography

[1] *Gnutella2*, http://www.gnutella2.com.

[2] *Kazza*, http://www.kazza.com.

[3] *napster*, http://www.napster.com.

[4] *Planetlab*, http://www.planet-lab.org/.

[5] Luc Onana Alima, Ali Ghodsi, and Seif Haridi, *A framework for structured peer-to-peer overlay networks*, Global Computing: IST/FET International Workshop, GC 2004 Rovereto, Italy, March 9-12, 2004 Revised Selected Papers, 2005, p. 27.

[6] Luc Onana Alima, Seif Haridi, Ali Ghodsi, Sameh El-Ansary, and Per Brand, *Position paper: Self-properties in distributed k-ary structured overlay networks*, Proceedings of SELF-STAR: International Workshop on Self-* Properties in Complex Information Systems, May 2004.

[7] L.O. Alimal, S. El-Ansary, P. Brand, and S. Haridi, *Dks(n, k, f): a family of low communication, scalable and fault-tolerant infrastructures for p2p applications*, Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on (12-15 May 2003), 344–350.

[8] Miguel Castro, Manuel Costa, and Antony Rowstron, *Debunking some myths about structured and unstructured overlays*, NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation (Berkeley, CA, USA), 2005, pp. 85–98.

[9] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron, *One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks*, EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop (New York, NY, USA), ACM, 2002, pp. 140–145.

[10] Ali Ghodsi, *Distributed k-ary System: Algorithms for distributed hash tables*, PhD dissertation, KTH—Royal Institute of Technology, Stockholm, Sweden", oct 2006.

[11] Zygmunt J. Haas, *A new routing protocol for the reconfigurable wireless networks*, Universal Personal Communications Record, 1997. Conference Record., 1997 IEEE 6th International Conference on **2** (1997), 562–566.

[12] Zygmunt J. Haas and Marc R. Pearlman, *The performance of query control schemes for the zone routing protocol*, IEEE/ACM Trans. Netw. **9** (2001), no. 4, 427–438.

[13] A. Iamnitchi, I. Foster, and D.C. Nurmi, *A peer-to-peer approach to resource location in grid environments*, High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on (2002), 419–.

[14] Adriana Iamnitchi and Ian Foster, *On fully decentralized resource discovery in grid environments*, International Workshop on Grid Computing (enver, Colorado), IEEE, November 2001.

[15] Adriana Iamnitchi, Ian Foster, and Daniel C. Nurmi, *A peer-to-peer approach to resource discovery in grid environments*, In High Performance Distributed Computing (Edinbourgh), 2002.

[16] L. Keong, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, *A survey and comparison of peer-to-peer overlay network schemes*, Communication Surveys & Tutorials, IEEE (2005), 72–93.

[17] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu, *Peer-to-peer computing.*

[18] Rafael Moreno-Vozmediano, *Resource discovery in ad-hoc grids*, International Conference on Computational Science (4), 2006, pp. 1031–1038.

[19] B. Pourebrahimi, K.L.M. Bertels, G. Kandru, and S. Vassiliadis, *Market-based resource allocation in grids*, proceedings of second IEEE International Conference on e-Science and Grid Computing, December 2006, p. 80.

[20] B. Pourebrahimi, K.L.M. Bertels, and S. Vassiliadis, *Survey of peer-to-peer networks*, Processings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc 2005, November 2005.

[21] B Pourebrahimi, K.L.M. Bertels, S. Vassiliadis, and L.O. Alima, *A dynamic pricing and bidding strategy for autonomous agents in grids*, Sixth International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2007), May 2007.

[22] S. Ratnasamy, P. Francis, M.Handley, R. Karp, and R. Karp, *A scalable content-addressable network*, SIGCOMM 01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communication, October 2001, pp. 161–172.

[23] Antony Rowstron and Peter Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), nov 2001, pp. 329–350.

[24] R.Vozmediano, *A hybrid mechanism for resource/service discovery in ad-hoc grids*, (2008).

[25] Mohammad Imran Shaik, S. Mary Saira Bhanu, and N. P. Gopalan, *Distributed grid resource discovery with matchmakers*, SKG '06: Proceedings of the Second International Conference on Semantics, Knowledge, and Grid (Washington, DC, USA), IEEE Computer Society, 2006, p. 28.

[26] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, *Chord: A scalable Peer-To-Peer lookup service for internet applications*, Proceedings of the 2001 ACM SIGCOMM Conference, 2001, pp. 149–160.

[27] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi, *Peer-to-peer resource discovery in grids: Models and systems*, Future Gener. Comput. Syst. **23** (2007), no. 7, 864–878.

[28] Konstantin Welke, *An overview of distributed k-ary system*, Seminar paper for Peer-to-Peer Networks, Department of Computer Science, University of Freiburg, March 2007.

[29] B.Y. Zhao, Ling Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, *Tapestry: a resilient global-scale overlay for service deployment*, Selected Areas in Communications, IEEE Journal on **22** (2004), no. 1, 41–53.