

# 國立交通大學

## 資訊科學與工程研究所

### 碩士論文

同儕網路上資源排程方法之公平性研究



On Fair Resource Scheduling over Peer-to-Peer Networks

研究生：張育誠

指導教授：陳俊穎 教授、王豐堅 教授

中華民國 九十七 年 九 月

同儕網路上資源排程方法之公平性研究  
On Fair Resource Scheduling over Peer-to-Peer Networks


研究生：張育誠

Student：Yu-Cheng Chang

指導教授：陳俊穎、王豐堅

Advisor：Jing-Ying Chen, Feng-Jian Wang

國立交通大學  
資訊科學與工程研究所  
碩士論文



A Thesis  
Submitted to Institute of Computer Science and Engineering  
College of Computer Science  
National Chiao Tung University  
in partial Fulfillment of the Requirements  
for the Degree of  
Master  
in  
Computer Science  
September 2008  
Hsinchu, Taiwan, Republic of China

中華民國九十七年九月

# 同儕網路上資源排程方法之公平性研究

研究生：張育誠

指導教授：陳俊穎 博士、王豐堅 博士

國立交通大學  
資訊科學與工程研究所  
碩士論文

## 摘要

同儕系統由於可以提供使用者相互合作及資源共享的環境，在近幾年來得到廣泛的使用。而在同儕架構上一個重要的議題便是如何達到公平性，使得所有網路的參與者均可公平的貢獻或獲得資源。關於這問題，在分散式檔案共享的應用上有廣泛的討論。但公平性的問題並不只侷限在這種形式的資源共享，也應包括動態資源像是空間機器等。在這篇文章中我們提出一個比一般平衡負載方法更嚴謹的公平計算方式，更符合動態資源共享的公平性。具體來說，此方法將整個同儕網路視為一個仲裁資源需求及供給的虛擬佇列，而以個別要求被插隊的次數來做為公平性的基準。我們利用模擬來比較不同資源排程方法的公平性，並提出一個樹狀方式的演算法作為改進。實驗結果顯示我們的方法有不錯的反應時間，並同時能達到較佳的公平性。

**關鍵字：**同儕網路、自主運算、公平性、資源排程

# On Fair Resource Scheduling over Peer-to-Peer Networks

Student: Yu-Cheng Chang


Advisor: Dr. Jing-Ying Chen, Dr. Feng-Jian Wang

Institute of Computer Science and Engineering

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

## Abstract



The last few years saw the growing popularity of peer-to-peer (P2P) systems that enable collaborative, decentralized sharing of files or other types of resources such as machine cycles, communication bandwidth, storage space, and so on. One important issue of P2P architecture is to ensure fairness – whether all participants can contribute and/or receive their fair shares of resources. While there have been extensive studies on the problem of distributing data in a fair and fully decentralized manner, it remains an open question whether fairness can also be achieved for other types of resources, especially when resource requestors and providers behave highly dynamically and irregularly. In this thesis we propose a more stringent fairness measure than the usual load balancing indicators found in literature. Specifically, the entire P2P network is modeled as a virtual queue where requests for resource consumption and contribution arrive indefinitely. Fairness is judged by the degree of preemption, i.e. the number of times a given request is cut in line by other late-arriving requests. In such a model, existing approaches to load balancing fail to achieve fairness satisfactorily if consumption requests are scheduled based on local information. To address this problem, we first investigate the impact of the proposed fairness model on common P2P networks via

simulation, and then propose an alternate scheduling algorithm that routes consumption requests along spanning trees to awaiting providers. The results show reasonable performance in terms of average response time and locality when compared to other decentralized load-balancing algorithms, while keeping the fairness measure low when compared to scheduling via centralized queue.

**Keywords:** peer-to-peer network, autonomic computing, fairness, resource scheduling



## 誌謝

本篇論文的完成，首先要感謝我的指導教授陳俊穎博士與王豐堅博士兩年來不斷的指導與鼓勵，讓我在軟體工程、同儕網路的技術上得到很多豐富的知識，使我可以在資源排程的公平性問題上找到靈感。另外，也非常感謝我的畢業口試評審委員陳健博士以及黃慶育博士，提供許多寶貴的意見，補足我論文裡不足的部分。

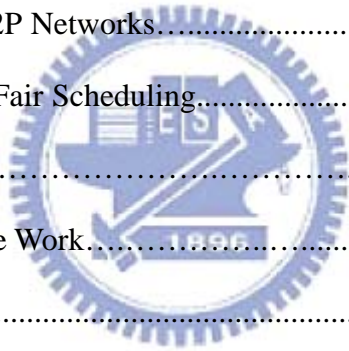
其次，我要感謝實驗室的學長姊們在研究生涯上的指導與照顧，讓我學得許多做研究的方法和技巧，得以順利的撰寫論文。

最後，我要感謝我的家人，由於有你們的支持，讓我能讀書、作研究到畢業。此外，謝謝同期碩士班好友們的打氣，在我遇到挫折時能互相勉勵並度過難關。由衷地感謝你們大家一路下來陪著我走過這段研究生歲月。



# Table of Contents

摘要 .....	i
Abstract .....	ii
誌謝 .....	iii
Table of Contents .....	vi
List of Tables .....	vii
List of Figures .....	viii
List of Algorithm .....	ix
Chapter 1. Introduction .....	1
Chapter 2. Background and Related Work .....	4
Chapter 3. FCFS Fairness for P2P Networks.....	9
Chapter 4. Decentralized FCFS Fair Scheduling.....	17
Chapter 5. Experiments.....	25
Chapter 6. Discussion and Future Work.....	39
Chapter 7. Conclusion .....	42
References .....	43



## List of Tables

Table 5.1: The environment parameters of simulation: ..... 27





## List of Figures

Figure 3.1: FCFS fairness modeled via a virtual producer-consumer queue.....	13
Figure 3.2: A cut-in line example.....	14
Figure 3.3: FCFS fairness modeled via multiple distributed queues.....	15
Figure 4.1: Virtual FCFS queue implemented as a tree of managers.....	18
Figure 5.1: Decentralized Search.....	26
Figure 5.2: Centralized managers.....	27
Figure 5.3: FFI Statistics.....	28
Figure 5.4: Average response time.....	28
Figure 5.5: Average response time and turnaround time.....	29
Figure 5.6: Average hop number.....	30
Figure 5.7: FFI Statistics between 1%, 5%, 10%, 20% requestors.....	31
Figure 5.8: Average response time between 1%, 5%, 10%, 20% requestors.....	32
Figure 5.9: Average hop number between 1%, 5%, 10%, 20% requestors.....	33
Figure 5.10: Change of FFI over time.....	34
Figure 5.11: Average job number among resource providers.....	34
Figure 5.12: FFI for the cases of 5%, 15%, 50%, 75%, 100% managers.....	35
Figure 5.13: Average response time between 5%, 15%, 50%, 75%, 100% managers.....	36
Figure 5.14: FFI and Average response time.....	37
Figure 5.15: Change of overhead over time.....	38

## List of Algorithm

Algorithm 4.1: Spanning Tree.....	18
Algorithm 4.2: Spanning Tree – Random.....	21
Algorithm 4.3: Spanning Tree - Excessive-Resource First.....	21
Algorithm 4.4: Spanning Tree - Member adjustment.....	23



## Chapter 1 Introduction

The last few years saw the growing use of peer-to-peer (P2P) software among end users. One widely recognized application area of P2P-based technology is to enable sharing of files or other types of resources such as machine cycles, communication bandwidth, storage space, and so on. One important issue of P2P architecture is the notion of fairness – whether participants joining a P2P network can contribute and/or receive their fair shares of services, respectively, especially when resource allocation decisions are made in a distributed, decentralized manner. Fairness manifests itself in many forms in P2P networks depending heavily on the types of resource being shared, as well as the expected behavior of the resource requestors and providers. Free riding, for example, concerns those selfish or malicious participants who appear to conform to the specified protocol, and approaches to dealing with the problem via various reputation earning or other incentive mechanisms have been proposed. However, even when all the participants are well-behaved, other kinds of fairness issues still remain. The power law phenomena widely recognized for P2P networks are an immediate example, which states that P2P networks evolve into structure in which relative few participants will host most of the files or other resources than the rest.

In this thesis we investigate the fairness issue when sharing generic, homogeneous resources such as machine cycles in a P2P network. This topic is closely related to the resource scheduling problem studied extensively in cluster or grid computing communities. However, most resource scheduling approaches in this context assume steady supply of resource providers, and their emphases are mainly on improving request fulfillment from the perspective of resource requestors. Such a biased view can easily lead to unfairness for resource providers, especially when scheduling decisions are based only on information local to some schedulers – a common design shared by many P2P networks. Moreover, there may

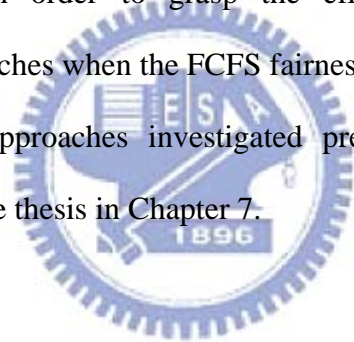
be application areas where the demand for fairness is so paramount that it trumps the other performance indexes, such that the provider who provides the resource “first” should receive the next available request, even though doing so may incur additional overhead when the request is issued by a remote requestor across the P2P network.

The scenario described above leads to a first-come-first-serve (FCFS) requirement for the scheduling of resources across P2P networks. Accordingly, we propose a fairness measure based on such FCFS requirement, which is considerably more stringent than the usual load balancing measure commonly addressed in literature. Specifically, the entire P2P network is modeled as a virtual queue where requests for resource consumption and contribution arrive indefinitely, possibly in highly skewed patterns. To simplify terminology, in what follows we refer *requests* to the consumption requests issued by resource requestors, while *resources* to contribution requests from resource providers. In other words, the latter correspond to “vaporous” resources that are created by providers and are used at most once. Fairness is judged by the degree of preemption, i.e. the number of times a given request (resource) is queue-jumped by other late-arriving requests (resources) that are fulfilled sooner. In such a model, existing approaches to load balancing fail to achieve fairness satisfactorily, for example, if resources are scheduled based largely on information local to the scheduler.

To address this problem, we investigate the impact of the proposed fairness model on common P2P networks via simulation, and propose a class of scheduling algorithms that routes requests along a spanning tree to awaiting providers. The results show reasonable performance in terms of average response time and locality when compared to other decentralized load-balancing schedulers, while keeping the fairness measure low when compared to scheduling via centralized servers. This thesis is meant as a first step towards the development of P2P networks with improved FCFS fairness measure. Although for the time being the application areas may be limited, we believe there could be important applications

in near future, especially because the continuously growing popularity of resource sharing and collaboration P2P networks can help drive the demand for *true* fairness that holds even for highly irregular access patterns.

The rest of this paper is organized as below. In Chapter 2 we survey research areas that are related to P2P systems from general autonomous systems studies to specific P2P scheduling and fairness mechanisms. In Chapter 3 we define the resource scheduling problem to be addressed in this thesis and the corresponding fairness model based on virtual queues. In Chapter 4, further assumption about scheduling resources based on FCFS fairness is discussed and several classes of scheduling algorithms are described, including some decentralized scheduling algorithms we propose based on spanning trees. In Chapter 5 we perform various experiments via simulation in order to grasp the effect of various centralized and decentralized scheduling approaches when the FCFS fairness index is concerned. In Chapter 6 we evaluate the scheduling approaches investigated previous and discuss some future directions, and then conclude the thesis in Chapter 7.



## Chapter 2 Background and Related Work

P2P networks [25] are now an important part of the emergent research discipline, called autonomic computing [3] [5] [13] [17] [20], that focus on the design and development of large-scale systems that are self-managing, self-organization, self-healing, and self-protecting, with less to none human intervention. Although P2P technology is commonly seen in file-sharing applications, today P2P networks have evolved into many application areas such as telephony, media streaming, etc., with drastically different internal structures.

Some early P2P systems such as Napster adopt a client-server model, where the central server often serves as a registry through which clients can locate other clients, or the resources behind them. In contrast, *pure* P2P networks strive for scalability, fault tolerance, and decentralized control, and try to remove any form of central servers to avoid single point of failure. Early pure P2P systems are unstructured in that the network overlay evolves gradually when peers enter or leave. Gnutella, one of the earliest P2P file sharing networks, updates its overlay topology locally without global coordination, and requests for files are *flooded* along the overlay for a certain diameter; peers with matching files will response to the requests. In Freenet [4], in contrast, receiving a new peer may involve collaboration among multiple peers to decide the responsibility of the new-coming peer. File requests traverse, rather than flood, the overlay topology according to some traversal order based on the file key as well as the historical record kept in the visiting peer (summarized in each peer's "routing table"). Accordingly, the overlay topology evolves as requests pass by, and the performance of search improves overtime, exhibiting an impressive self-optimizing and self-learning system.

Gnutella and Freenet are often classified as a type of *unstructured* P2P network, where the network overlay does not evolve in a predetermined way but driven by peer dynamism. In

contrast, a *structured* P2P network employs a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has the desired resource, even if the resource is extremely rare.

Chord [30] is a well-known system that located peers and storage data in distributed manner. It uses a consistent hashing function to compute the peer responsible for storing the key's value. Each peer maintains  $O(\log N)$  routing information of other peers and solves lookups by  $O(\log N)$  messages to other peers,  $N$  being the number of peers. Other structured P2P networks such as Pastry [29] and CAN [27] have similar idea but with different approaches to building the underlying network overlays.

Beyond file sharing, P2P technology has also been used for resource scheduling in distributed computing environments. In this direction, P2P networks are closely related to grid computing. A grid computing environment attempts to bring together disparate machine resources scattered around the network behind a small set of standardized protocols and programming interfaces (e.g. [23]), so as to provide infrastructure on top of which multiple *virtual organizations* can be formed. To help grid applications make better use of the underlying resources, it is important that there is sufficient support for resource scheduling. The open-source Globus toolkit [10] is one widely used toolkit to help building large-scale grid environments, in which there have been established modules for resource allocation and scheduling. There are other grid-related projects or systems that support more elaborated resource scheduling systems.

As mentioned previously, fairness is a crucial issue especially for P2P networks to operate and evolve soundly, and fairness manifests itself in many forms that are not easy to handle simultaneously. Different application areas have different set of requirements and considerations when fairness is concerned. Also, the same fairness mechanism that works in one domain may fail miserably in another domain.

Load balancing is probably the most basic and common criterion when fairness is concerned. Numerous researches concern this about the distribution of items and peers. Although there are still different interpretations about what load balance means in P2P network, they can all be characterized as measuring "how equally" the objects are allocated to the peers, and can best be illustrated by Jain's fairness index [15]. The formula can be expressed as

$$\text{fairness} = \frac{(\sum x_i)^2}{(n \cdot \sum x_i^2)}$$

where  $x_i$  represents the percentage of the resources assigned to peer  $i$ , and the measure ranges from  $1/n$  (worst case) to 1 (best case). This fairness index and its variations have been used by many researchers to measure fair resource allocation for communication networks (e.g. congestion control).

For P2P networks, as an example, many structured approaches based on DHTs attempt to achieve even distribution of items (or other load measures) to nodes in the DHT. In general, load balance is done by first randomizing the DHT address associated with each item with a "good enough" hash function, and then making each DHT node responsible for a balanced portion of the DHT address space. The assumption that item keys can be randomized uniformly allows DHT-based P2P networks to evenly distribute items among peers.

The randomization step, although simple and effective, restricts the kinds of queries. Obviously, due to the randomization effect, content-based (e.g. keywords) or range-based queries become difficult to implement. In [16], in addition to the usual DHT model where item keys can be randomized, another solution is also proposed that can tackle the case where the item distribution can be arbitrary.

Another issue that is faced by key randomization is proximity awareness. [33] proposes a proximity-aware load balancing algorithm for DHT-Based P2P systems. In the approach,



landmark clustering is used to generate proximity information. It is based on an intuition that nodes physically close to each other are likely to have similar distances to a few selected nodes. The proximity information is computed in a decentralized manner, and is used to guide virtual server reassignments such that virtual servers are reassigned and transferred between physically close heavily loaded nodes and lightly loaded nodes.

The abovementioned load balancing approaches are for structured (DHT-based) P2P networks, [8] concerns *unstructured* (Gnutella-like) P2P networks and proposes a load-balancing resource allocation scheme to distribute data in a decentralized manner. The idea is simple: a peer estimates its own fairness index as well as its neighbors'. When the peer finds itself has higher load than (the average of) its neighbors, it will try to replicate some of its (popular) items to its immediate neighbors hoping to shed some burden. To reduce overhead, their algorithm piggy-backs load measurement information in normal messages.

Free-riding is another problem commonly faced by P2P networks where peers consume resources solely without contributing anything to the network. For the purpose of fair resource sharing, [12] presents a reputation based trust management system called TruthRep. It concerns the honest feedback of peers and uses a feedback generation formula to calculate the reputation value. By adjusting the reputation value from time to time, each peer can have a fair reputation value with fewest errors. [21] discusses how to avoid cheating and free riders under storage sharing P2P networks. It uses an auditing mechanism to require each peer monitors others and check that whether they are cheating or not. Using a proposed accounting mechanism, each peer gets credits by providing storages and consumes it by using bandwidth. Fair resource scheduling in P2P networks is a relatively new topic compared with the fairness mechanisms mentioned above, but it has been studied extensively in networking research community. [9] addresses the problem of scheduling *divisible* requests that can be distributed partially among providers. The resource model here is more general as there are many types

of services and resources, and each service corresponds to an allocation of a set of resources. Accordingly, the problem is translated into the rate control problem in IP networks, and the authors adapt an existing congestion pricing approach to ensure some form of fairness. Specifically, the P2P network is an unstructured network of peers with heterogeneous capacities and neighboring peers are associated with a rate. A request comes with a price offer and a distributed algorithm is proposed that computes the rates such that resources are allocated based on these prices. It is shown that with this approach a peer gets a fair share of the resources available in the P2P network weighted by its contribution to the network.



## Chapter 3 FCFS Fairness for P2P Networks

The fairness models described in the previous chapter do not cover the landscape of P2P networks for resource sharing and scheduling entirely. Many fairness measures are asymptotic in nature. For example, consider a DHT-based file sharing system that distributes files among peers evenly through file key randomization. The load balancing index (e.g. Jain's fairness index) is measured over a long period of time. However, the measure assumes that new files enter the P2P network continuously. Before the number of files stored in the network grows to a certain point, load balance cannot be achieved with certainty. In addition, assume that files are large in quantity and have been evenly distributed among peers, there are still uneven distribution about the file request patterns – some files may receive significantly more interests than the others, and the peers hosting these popular files will see much more network traffic than the others. This scenario points out at least one important aspect of fairness: not only does it depend heavily on the types of resources being shared, but also depend subjectively from different point of views. In the example above, the network is fair in terms of file storage (more precisely, file counts), but unfair in terms of network traffic.

For the sake of comparison, consider another important class of systems that share files over *unstructured* P2P networks. For example, Gnutella in its original form allows a user, through a representing peer, to issue (keyword-based) queries. Gnutella floods the queries over the P2P overlay with bounding distance, and peers what store files matching queries will reply the query with results. Because Gnutella peers (servents) mainly share their own files to the rest of the network and do not have to “cache” files they are not interested, applying the same load-balancing measure as the case for DHT-based systems is less relevant. Instead, it is more relevant if the fairness index is used to measure the overhead of replying requests (when the peer has matching files) plus forwarding requests (flooding). On the other hand, Freenet –

another well-known file sharing system over unstructured P2P networks – randomizes the keys of files in a way similar to DHT-based methods, and the user is supposed to know the hashed keys when forming queries (with the help of other auxiliary tools external to the core of Freenet overlay). Freenet then attempts to re-organize the overlay structure adaptively to increase query response time and hit rates. In this case, the load-balancing measures for both file storage and messaging overhead seem relevant.

In this thesis, we focus on sharing *generic, vaporous* resources that can be consumed once only, which is quite different from sharing of files discussed above. For simplicity, we assume there is only one type of resources; hence the requests are also of the same type. All resource providers, each serving as a peer in the P2P network, provide the same type of resources, and requests for resources can be issued by any peer. The goal of such a resource sharing P2P system is, for each resource request, to locate a resource, preferably from among peers close to the requesting peer, to *fulfill* the request.

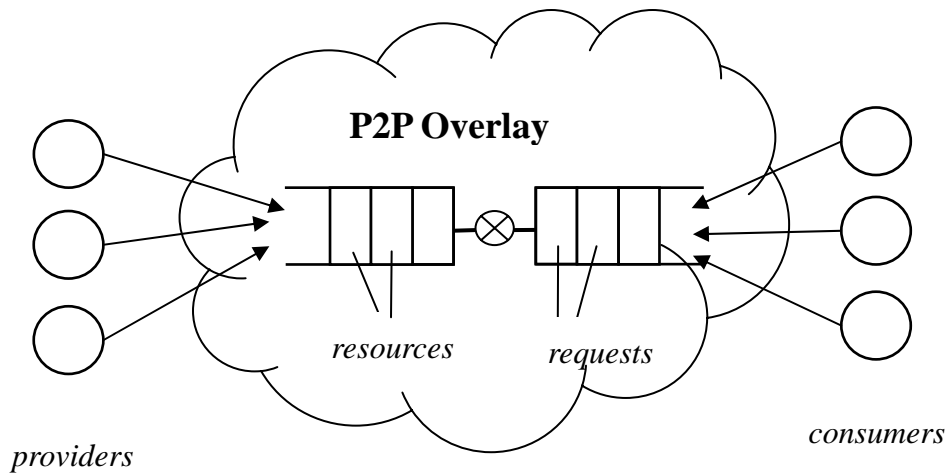
The kind of resources described above is quite common, such as machine cycles or file storage. However, some adjustment, at least conceptually, is needed. Although both machine cycles and storage space are generic, they are not considered vaporous in literature in general. In our model, when machine cycles are the resources to be shared, a peer providing a resource means the peer is available for processing a request, and the availability may be advertized by the peer to some registry. In other words, each peer provides at most one resource at a time, and which the resource is matched by a request, the peer enters into a “computing” mode and cannot accept next request until the request is processed completely (or equivalently, the peer no longer provides the resource during request processing). Furthermore, upon completion, the peer needs to advertize again about its availability. Therefore, our resource model still differs slightly from the resource model assumed by the general resource scheduling problem studied in grid or cluster computing. Similarly, storage space as a generic, vaporous resource

in our model can be interpreted. The main point is that the allocated storage space is treated like a *temporary* working space since after the allocated space serves its job fulfilling a request, it cannot serve as a persistent entry in the P2P network for subsequent access (or the cost of accessing it subsequently should be assumed ignorable), otherwise the model falls back to file sharing. Note that the assumption of single-type requests does not prevent requests from representing different kinds of *jobs*, as long as all resources can perform all the jobs. Hence, for example, our model can describe “cycle-stealing” P2P systems such as SETI@Home directly. With additional modification, our model can also be used to deal with more general resource scheduling problems that are addressed, for example, in cluster or grid computing research.

With the resource sharing model described above, what are the potential issues when fairness is concerned? As mentioned, this topic is closely related to the resource scheduling problem studied extensively in cluster or grid computing communities. However, most resource scheduling approaches in this context assume steady supply of resources from providers, and different approaches emphasize mainly on improving request fulfillment performance from the perspective of resource (or job) requestors. Such a biased view can easily lead to unfairness for resource providers if resource assignment is based on expected turn-around time or locality. For example, consider a situation in which a series of requests come steadily and moderately from a particular requestor in a P2P network for a period of time, while other requestors in the network remain (relatively) silent. Assuming that all resource providers are of similar processing power, it is likely that decentralized schedulers that make decisions based on gradually accumulated status about other peers will select the providers in the vicinity of that requestor to process the requests. If such a pattern of requests persist, providers at a distance from the focal requestor will be driven into starvation. This situation poses no problems for clusters and grids not only because most existing systems are

based on centralized control, but also because the goal is to minimize job processing time from the requester's perspective. In P2P settings, whether this result is considered unfair or not depends on the application. In case, for example, processing each request implies earning some credits for the providers; then naturally resource providers would contend for requests, and it is unprecedented that the scheduler should be fair all the time irrespective to any pattern of requests that may occur. Moreover, there may be application areas where the demand for fairness may override all other performance metrics, such that the provider who provides the resource "first" should receive the next available request, even though doing so may incur additional overhead when the request is issued by a remote requestor across the P2P network, leading to a FCFS requirement effectively.

We are interested in such FCFS-oriented fairness measures for P2P networks. Accordingly, our resource sharing model can be viewed as a *virtual producer-consumer* problem as depicted in Figure 3.1 below, which in some sense is a reverse of the ordinary resource scheduling problem where what arrive are jobs that demands resources, and the scheduler (the middle man) dispatches jobs to available resources *actively* based on some scheduling policies. In contrast, in our model the scheduler, that is, a virtual producer-consumer queue played by the P2P overlay, serves largely as a passive registry, and queued resources are only consumed when requested.



**Figure 3.1. FCFS fairness modeled via a virtual producer-consumer queue**

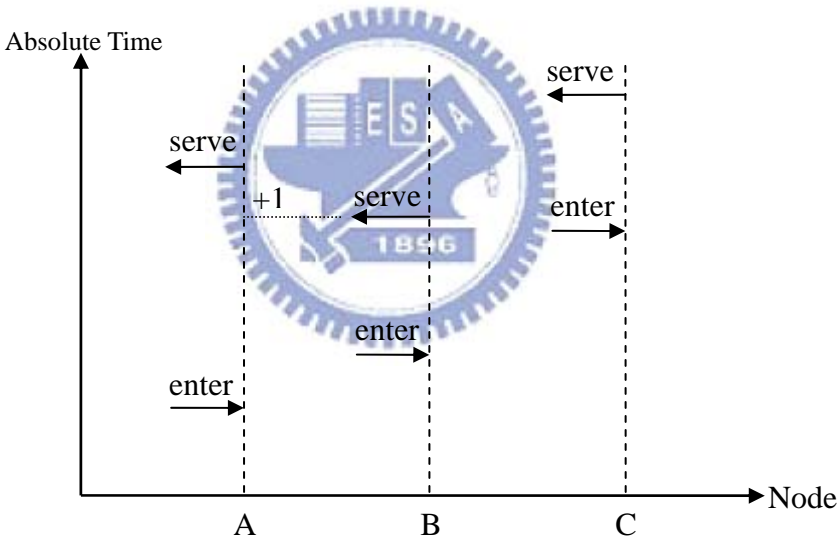
Naturally, the FCFS fairness is achieved by ensuring that the P2P overlay collectively implements a *central* FCFS queue with the obvious property, that is, earlier arriving requests or resources expect to be fulfilled or consumed than those arrive later. However, unlike scheduling jobs in operating systems or clusters, the *events* of job arrivals as well as status changes of resource availability cannot take effect immediately and it takes time to propagate the knowledge throughout the P2P network. Accordingly, there is no straightforward means to evaluate which comes first when two events from two distant peers are generated roughly “at the same time.”

To address this problem without going into the trouble of deriving a logical *causality* framework (e.g. [28]), in this thesis we simply assume the existence of a global timer solely for the measurement of FCFS fairness, although the global timer is not visible to the actual P2P system. Using this “non-existent” global timer, all events have their corresponding timestamps, and the time when a resource (request) enters the virtual queue is determined *the moment the provider (requestor) expresses its intent*, often by initiating some protocol-specific messages. With this model, it becomes straightforward to measure FCFS

fairness: for a given resource (request) entering the queue, *how many resources (requests) queue-jump it* (i.e. cut in the line) subsequently.

**FCFS Fairness Index (FFI):** for each request (resource) entering the virtual queue, the average numbers of requests (resources) that queue-jump it.

To illustrate the idea of FFI further, consider Figure 3.2 in which there are three resource providers A, B, and C who enter the virtual queue successively (in the order A-B-C). Because B enters the queue after A, yet gets fulfilled earlier than A, it therefore queue-jumps A. On the other hand, C is served last and does not queue-jump others. Accordingly, the FFI is 1 for A, and the average FFI for the whole system is  $1/3$ .



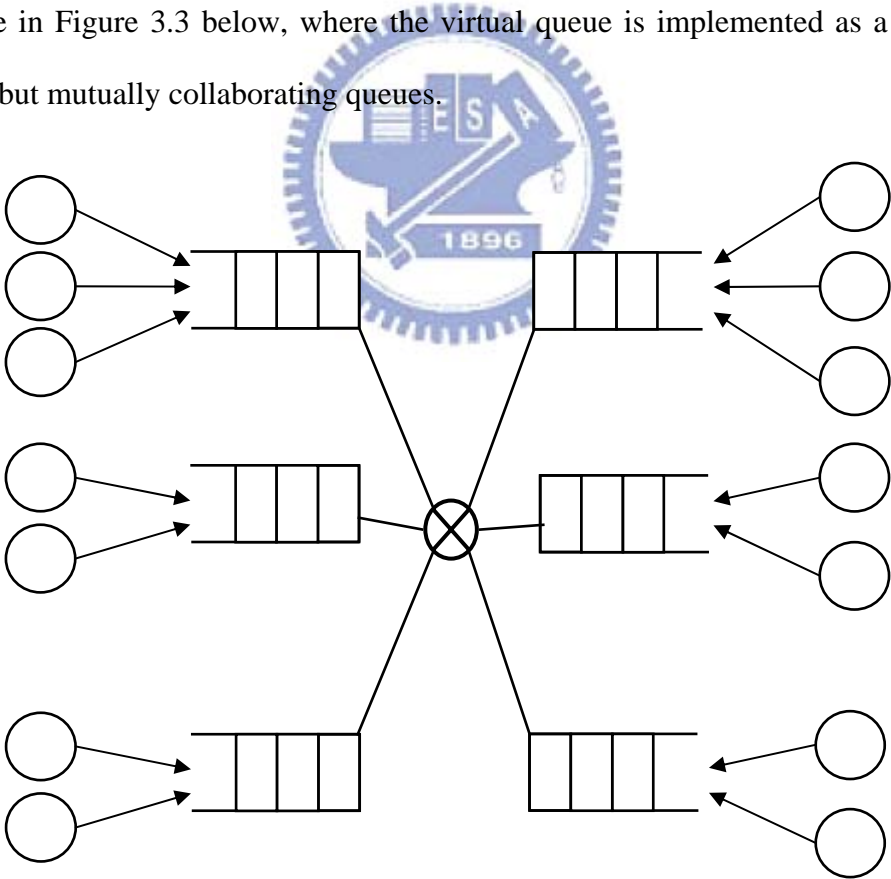
**Figure 3.2: A cut in line example**

Note that FFI measures requests and resources separately. Indeed, in this thesis we focus on minimizing FFI for resources only. Minimizing FFI for both resources and requests simultaneously and efficiently is considered future work. Also note that different variations of FFI measures are possible, especially if we measure the *maximum* number of queue-jumping experienced by the resources or requests rather than the average. The benefit of this variation is that it measures the “worst-case” scenario rather than averages. Nevertheless, our



simulation experiments are based on the average indexes only. Also note that the definition of FFI above makes it difficult, if not impossible to realize a *true* FCFS queue in P2P settings, as two requests occurring in a given order (based on the global timer) may reach a given peer in the reverse order due to unpredictable communication delays. However, the index is quite intuitive and straightforward to implement in simulator.

With the definition of FFI in mind, our goal is to investigate suitable implementation of an efficient, *decentralized virtual queue* over P2P networks. The implementation should be decentralized such that each peer should communicate with its neighbors and made scheduling decisions based on local information, although some local information may in fact contain partial global knowledge accumulated gradually as the network evolves. The idea can be illustrate in Figure 3.3 below, where the virtual queue is implemented as a collection of distributed but mutually collaborating queues.



**Figure 3.3. FCFS fairness modeled via multiple distributed queues**

The implementation should result in both low FFI and low request processing time, which are somewhat mutually conflicting since achieving low FFI suggests that some global knowledge should be accumulated or searched, e.g. to find the longest-waiting resources across the P2P network, which may result in longer response time in total when compared to simply searching for available resources nearby.

In addition, the desirable implementation should be as *proximity-aware* as possible. A proximity-aware algorithm tries to locate resources whose peers are *close* to the requesters in terms of communication speed, which is important, for example, for grid applications where executing a job may imply large-volume data transfers. As discussed in Chapter 2, although proximity-aware algorithms are desirable, the property is not built-in in many well-known P2P systems and requires additional infrastructure support.

Finally, the implementation should be adaptive. This is crucial because both the patterns of request generation and resource supply can vary over time, and the trend may last for a long period of time. As an illustration, simply consider different time zones around the globe. One probable, recurring trend may be that machines generate more requests during day time, while providing more resources during midnight. Obviously, it is possible to improve overall response time if the knowledge about request and resource supply patterns can be exploited when routine requests to resources.

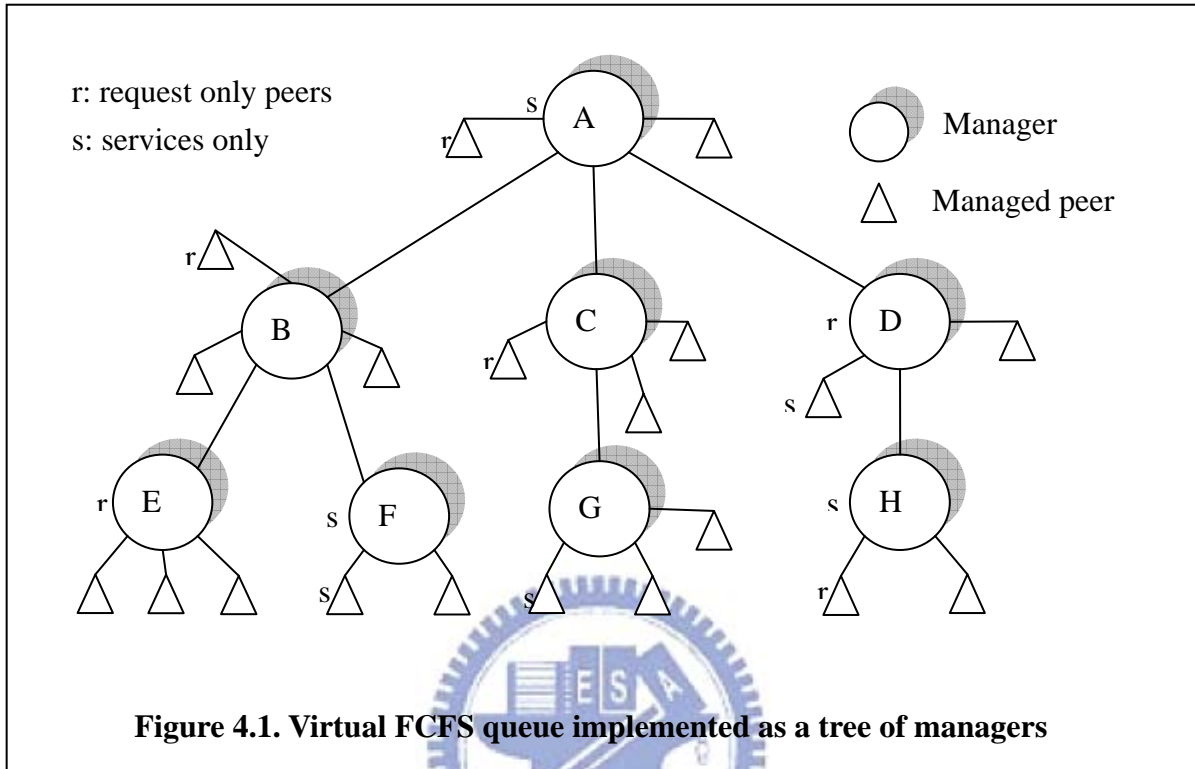
## Chapter 4 Decentralized FCFS Fair Scheduling

As mentioned in the previous chapter, our goal is to design efficient scheduling methods over P2P network that also have low FFI. In summary, a desirable virtual queue implementation should be:

1. **Decentralized**, where resource scheduling involving multiple collaborative peers,
2. **Low FFI**, so that queue jumpers are minimized,
3. **Low response time**, so that improving FFI does not incur too much overhead,
4. **Non-starving**, such that requests or resources, once enter the virtual queue, eventually get processed against skewed request patterns
5. **Proximity-aware**, so that resource scheduling should respect the underlying network topology when possible.
6. **Adaptive**, so that when mid-to-long term patterns of request or resource supply change, the virtual queue can adjust itself to minimize FFI and response time.

To meet the requirements, we propose a general approach based on spanning trees. As shown in Figure 4.1, the virtual queue is implemented as collaborating managers that are connected as a spanning tree. In particular, each manager serves both a resource queue and a request queue. Furthermore, we assume a many-one mapping from resource providers and requestors to these managers, so that every provider or requestor belongs to exactly one manager. Although this assumption seems unnecessary, as will become clear shortly, the reason behind it is to give the virtual queue the ability to adjust the mapping gradually when the network evolves. This is important when, for example, the patterns of resources and requests supply changes over time, or knowledge about proximity is learned. Note that in our

implementation model, a peer can be a resource provider, a requestor, or both. The virtual queue implementation is outlined in Algorithm 4.1.



#### Algorithm 4.1: Spanning Tree

##### Manager m

```

round : int; // current round number
roundCompleted : Boolean;
parent : Manager
children : Manager list
requests: Request queue
resources: Resource queue for current round
resourcesN: Resource queue for next round

```

##### init():

```

forming spanning tree
collect resources;
round = 0
roundCompleted = false;

```

*registerResource*(Resource res):

enqueue res to resourcesN

*requestFromLocal*(Request r):

match(r, resources.dequeue()) and updateStatus()

or, for each child in order given by *order*(children):

child.requestFromParent(r, m),

or, requestFromChildren(parent),

or, fail

*requestFromChildren*(Request r, Manager ch):

similar to requestFromLocal, but skip children ch

*requestFromParent*(Request r, Manager p):

similar to requestFromLocal, but only

downward (not calling requestFromChildren())

*match*(Request r, Resource res):

inform provider of res to process r;

when the provider finishes, it will inform the requestor of r

and then call registerResource() to its own manager

*updateStatus*():

if resources is empty and all children have reported roundCompleted

roundCompleted = true,

parent.updateStatus(),

of for the root, nextRound()

*nextRound*():

increment round

swap resources and resourceN

for all children ch: ch.nextRound();

In short, the implementation assumes that initially, the managers form a spanning tree in a distributed environment, which can be done in a decentralized and fault-tolerant manner (e.g.

self-stabilizing trees) and is not exploited further in this thesis. When a requester issues a request, it calls *requestFromLocal()* to the manager it belongs to. The manager searches for an available resource locally first, then its children recursively. If no resource is found, the parent of the manager is searched lastly.

The implementation is round-based in that resources in a given round are consumed before next round begin. A provider can issue at most one resource by calling *registerResource()*. New coming resources (created when a provider becomes available, possibly after processing a request several rounds ago) will be placed in the queue for the next round.

Generally speaking, the search order is essentially hierarchical (i.e. bottom-up, children-first). However, for each manager, the order that its children are searched can still vary (via the *order()* function). Different strategies can be conceived to adjust the search order in order to help exploiting the network dynamism, that is, the change of patterns of both requests and resource supply over time. In addition, the network topology can evolve by reassigning peer membership as well as restructuring the spanning tree in a decentralized manner, for example, by taking into account proximity information accumulated when the network evolves. The spanning tree has great advantage in this regard because different kinds of information can be gathered and computed to aid heuristic search. We have investigated several self-adaptation heuristics to help improving the scheduling performance. They are described below:

**Random Search among Children** When a manager runs out of resource and attempts to forward a request to its children, it picks a child randomly from the set of children that have not reported to the manager that they have finished for the round. Note that such status updating is realized by *updateStatus()* in the base algorithm 4.1. Because it is still possible that the manager may not have the most up-to-date information at the time it is forwarding

requests, a child may still receive requests after it has reported round completion status to its parent. In this case the child simply replies as if it has run out of resources. This randomized strategy is outlined in Algorithm 4.2 with common part similar to the base algorithm 4.1 skipped.

**Algorithm 4.2: Spanning Tree - Random**

**Manager m**

*order()*:

```
collect children with !roundCompleted into childList
shuffle childList randomly
return the childList
```

**Excessive-Resource Child First** The idea is to start from the child who “seems” to have most excessive resources, including the available resources in all of its sub-trees, hoping to guess the right path earlier. This approach also has to potential to prevent contention of resources among “busy” managers. For example, suppose a given child (including its sub-trees) generates requests more than the resources it provides for a given round. If the child is searched first (e.g. using the randomized strategy described previously) and an available resource is indeed found, this may make the requests generated from that sub-tree later unable to find a resource close-by, hence incur additional search steps. The expected “excessiveness” for a child can be accumulated and averaged round by round easily through the spanning tree (when *nextRound()* is called).

**Algorithm 4.3: Spanning Tree – Excessive Resources First**

**Manager m**

```
requestCount : int;
resourceCount: int;
```

```

requestCountAvg : float;
resourceCountAvg: float;
total RequestCountAvg : float;
total ResourceCountAvg: float;

registerResource(Resource res):
    resourceCount++
    same as registerResource(res) in 4.1

requestFromLocal (Request r):
    requestCount++
    same as requestFromLocal (r) in 4.1

averageCounts():
    compute requestCountAvg based on requestCount and old
    requestCountAvg
    compute total RequestCountAvg based on requestCount,
    requestCountAvg, and childrens' total RequestCountAvg
    compute resourceCountAvg and total ResourceCountAvg similarly

nextRound():
    (root only) perform averageCounts() bottom-up
    same as nextRound () in 4.1

order():
    order children according to their (total ResourceCountAvg –
    total RequestCountAvg)

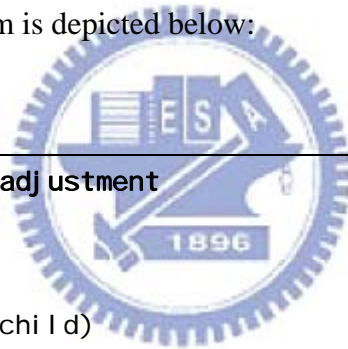
```

Note that how averages are obtained is not explicitly given in Algorithm 4.3. With a spanning tree, the averages can be computed easily in a bottom-up manner. Also, we do not give specific formula in *averageCounts()* for computing the averages for each manager because there may be alternatives. An extreme case is to compute the averages based solely on resourceCount and requestCount recorded in the current round disregarding the past



records. A more typical approach is to account for the averages from previous rounds using some weighting factors.

**Member Reassignment.** The idea behind this strategy is to distribute requests and resources evenly among managers by gradually reassigning producers and requesters to new managers in a decentralized manner. Intuitively, we would like to reassign the membership between a parent and a child if they have large difference in terms of net resource supply (i.e. average resources count minus average request count). Like the accumulation of request/resource counts in Algorithm 4.3, the net resource supply can be derived by considering both current and past resource/request statistics. With this reassignment step, it is hoped that the whole P2P network can evolve according to the patterns of resource supply and requests over time. The algorithm is depicted below:



**Algorithm 4.4: Member adjustment  
Manager m**

```

exchangePeers(parent, child)
    parentRequests = 0;
    parentResources = 0;
    halfRequests = (parent.requestAvg + child.requestAvg) / 2
    halfResources = (parent.resourceAvg + child.resourceAvg) / 2
    collect all peers belong to parent and child in I
    shuffle I randomly
    for each peer p in I
        if p is a provider
            if (parentResources < halfResources)
                assign p to parent
                parentResources += p.resourceAvg
            else
                assign p to child
        else similar for requestor p

```

Note algorithm 4.4 does not stated when and how a given parent-child pair is chosen to perform the member reassignment. Although it can be determined using a threshold, in our experiments in the next chapter we pick the pair that exhibit maximum difference in net resource supply, with the goal to examine the effectiveness of such member reassignment approach.

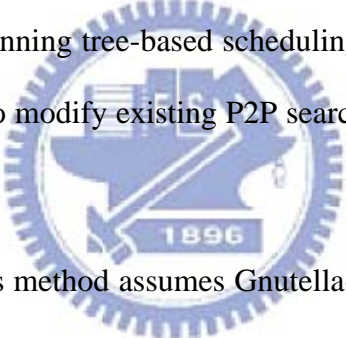


## Chapter 5 Experiments

In this chapter we evaluate several resource scheduling approaches for P2P networks via simulation. In addition to FFI and response time, there are also other important performance indexes we would like to investigate for different approaches:

- Turn-around time: the time interval between the request issue time and complete time.
- Overhead: the number of (control) messages for request routing, book-keeping messages, and so on.

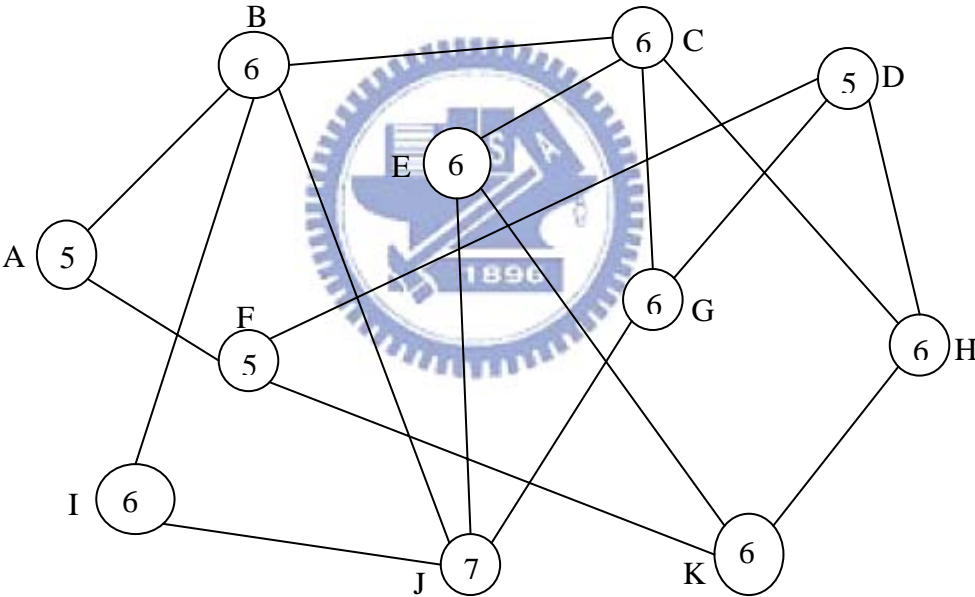
In order to compare our spanning tree-based scheduling approach to other P2P networks with different topologies, we also modify existing P2P search algorithms to suite our resource model. They are outlined below:



**Decentralized Search.** This method assumes Gnutella-like networks. The search is over a similar unstructured overlay where peers (including requestors and providers) maintain a limited number of neighbors. However, when a request is issued by a requestor, the request is *searched* (unlike flooding in Gnutella) among its neighbors. Like Freenet, some information is accumulated in each peer that is used to decide the order of search among its neighbors. Specifically, each peer maintains a *round* counter similar to our spanning tree-based approach, although the round number is not synchronized throughout the network. Like the spanning tree-based method, newly registered resources to a manager are always placed in the queue for the next round. In addition, new requests arrived at a manager will not be served locally if the manager knows that some of its neighbors have lower round number. In this case it will forward the request to the neighbor with lowest round number (but only up to a constant number of hops, called *TTL*, or time to live). The requests are served locally when all the

neighbors have equal or higher round numbers. Once the local resources are run out, the manager will advance to the next round and notify its neighbor about its new round number.

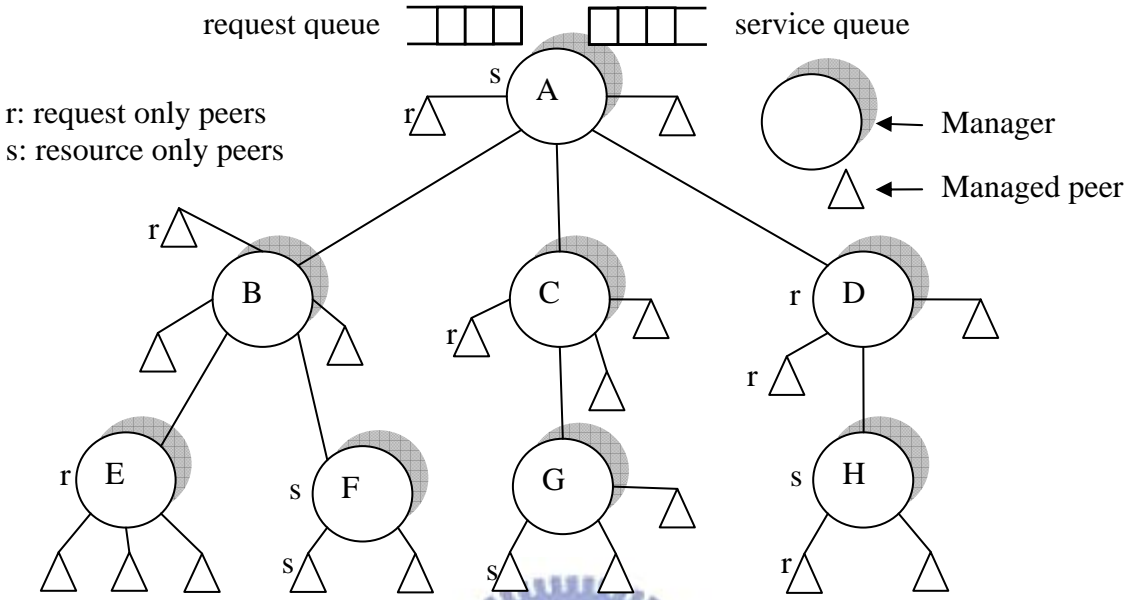
Figure 5.1 shows an example of a decentralized search over such a network in which each manager maintains a round number. Suppose manager B has round number 6 and it receives a request locally, it will forward the request to manager A, who will process the request locally because it still have resources available (otherwise it would have advanced to round 6 earlier). On the other hand, when manager C receives a request, it will process the request locally, and if the resource is the last one, manager C will increment its round number to 7 and notify its neighbors.



**Figure 5.1: Decentralized Search**

**Centralized Manager.** This method, somewhat similar to Napster, uses the same setup as our tree-based method, except that the root is assigned the responsibility of request registry, resource registry, and match making. In other words, the root implements the centralized producer-consumer queue, and all requests and resources that arrive at different managers are

routed through the spanning tree to the central manager. Figure 5.2 depicts the centralized architecture.

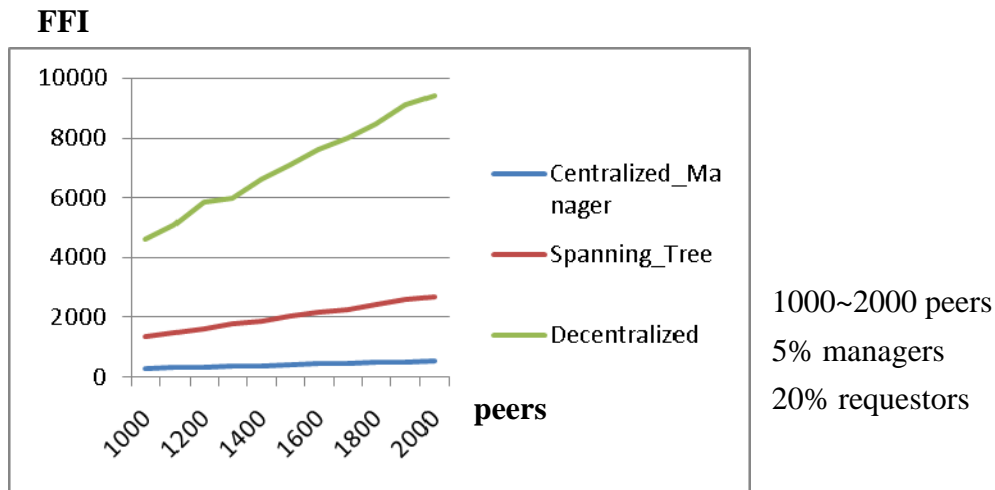


**Figure 5.2. Centralized managers**

Table 5.1 shows the list of parameters that are varied in our experiment. Each data point is obtained by performing 20 simulations each with a distinct, randomly generated network topology under the same network parameters. In the table, the communication speeds among participating peers and managers are relative.

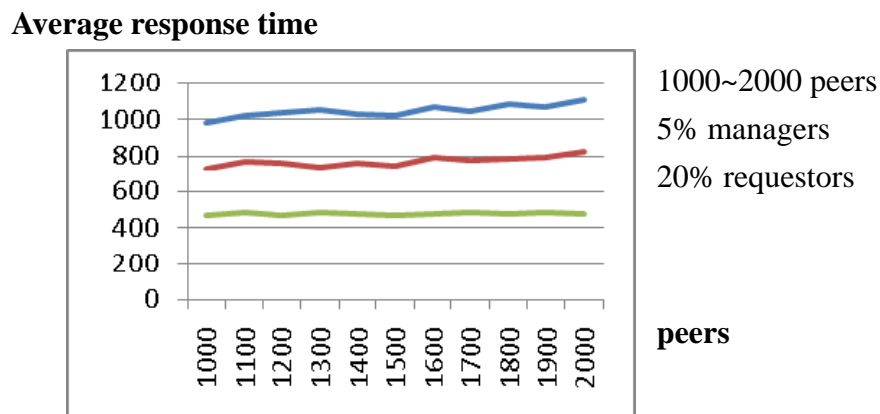
<b>Table 5.1: The environment parameters of simulation:</b>	
Network Size	1000~2000
Percentage of managers	5%, 15%, 50%, 75%, 100%
Percentage of Requesters	1%, 5%, 10%, 20%
Communication speed between managers	0.05, 0.25, 0.5
Communication speed between managers and managed peers	0.5
Communication speed between peers	0.5~1.5

Figure 5.3 shows the simulation result of FFI for a typical network setting, namely when there are 5% of managers among the overall network of peers (hence there are 20 peers in average assigned to each manager), where the number of nodes ranges from 1000 to 2000. Furthermore, there are also 20% of requestors.



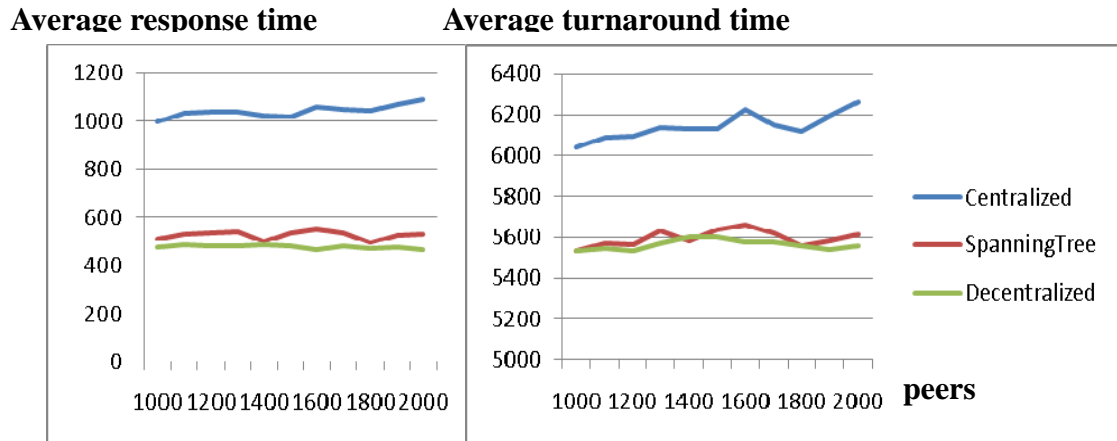
**Figure 5.3: FFI Statistics**

The result in Figure 5.3 shows that the tree-based method outperforms the decentralized method when FFI is concerned. When the response time is concerned, the tree-based method lies between the centralized and decentralized method, as shown in Figure 5.4 below:



**Figure 5.4: Average response time**

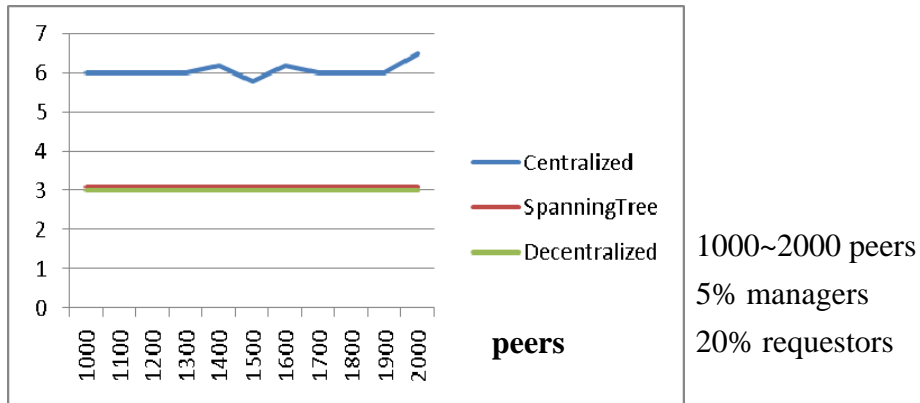
Similar results are also obtained when the request-only peers are set to 1%, as shown in Figure 5.5, where both the response time and turn-around time for the tree-based method is comparable to decentralized method; both of which are better than the centralized method.



**Figure 5.5: Average response time and turnaround time**

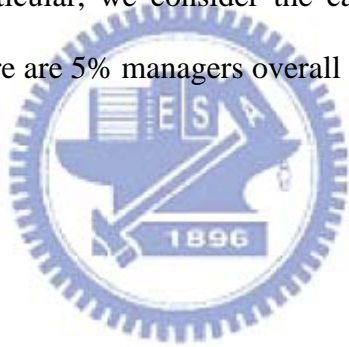
Figure 5.6 shows the the average hops for different methods. Note that similar to P2P networks such as Gnutella or Freenet, the decentralized method also imposes a fixed time-to-live (TTL) constant that limits the search range. Here it is set to 3. Accordingly, it is indicated in the figure that our tree-based method also has average hops of 3, while the centralized method has the average hop number doubled, due to the required bottom-up request routing. In general, based on our performance study, the tree-based method has good response time when compared to the decentralized method (which typically have higher FFI) while maintaining low FFI when compared to the centralized method (which typically have higher response time).

### Average hop number

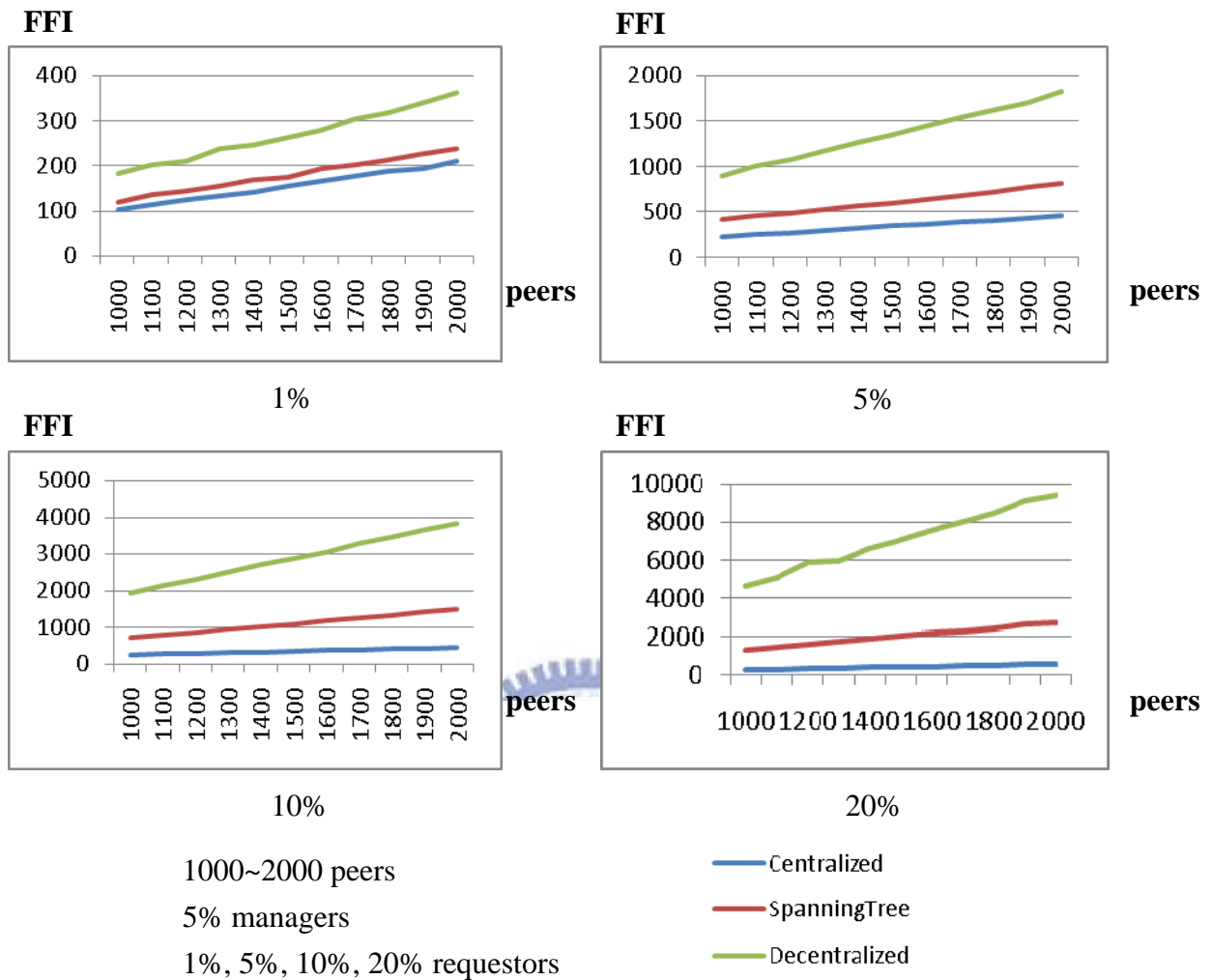


**Figure 5.6: Average hop number**

In the next set of experiments we would like to vary the percentage of request-only peers and observe the effects. In particular, we consider the cases for 1%, 5%, 10%, and 20% request-only peers assuming there are 5% managers overall (Figure 5.7).

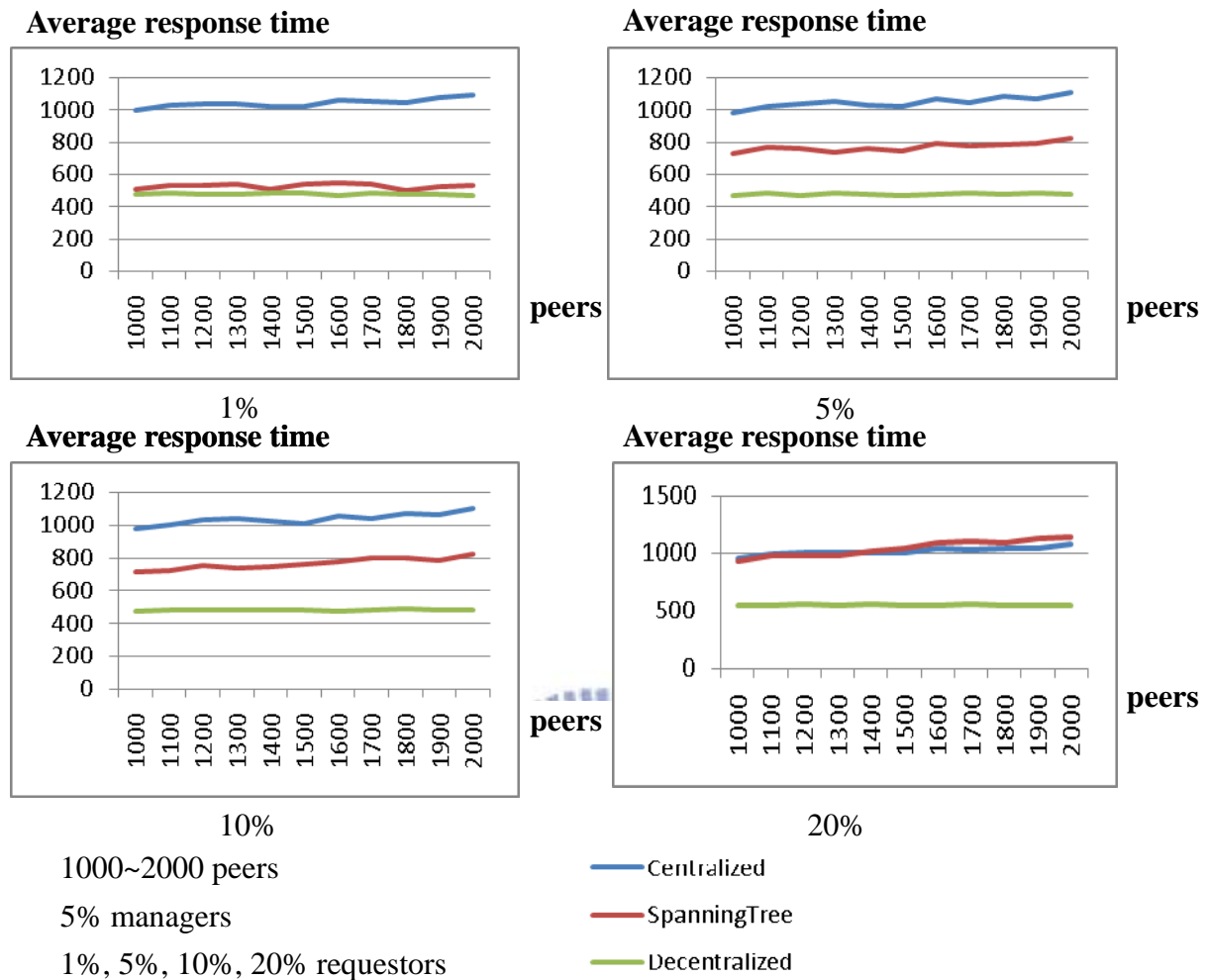






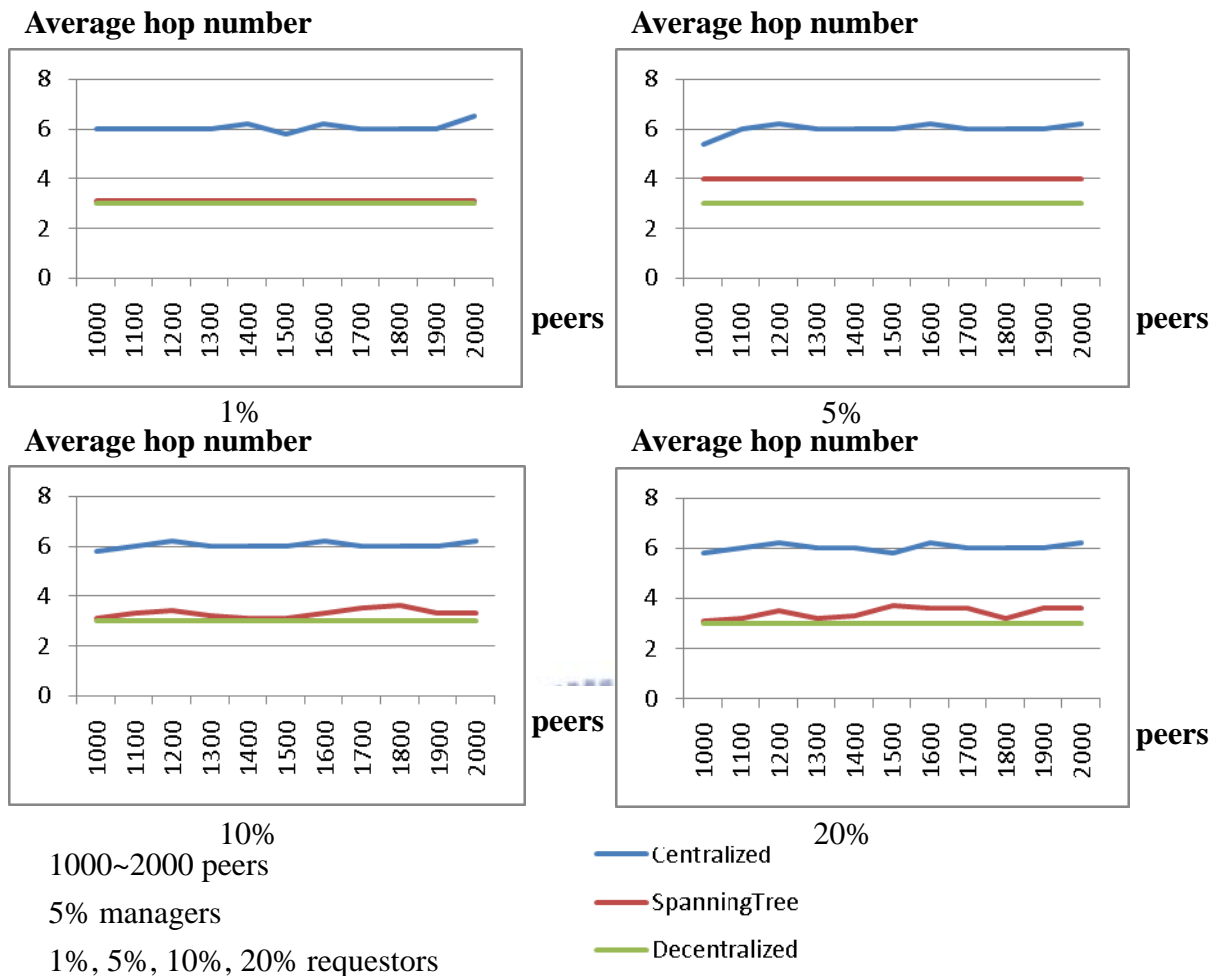
**Figure 5.7: FFI results with 1%, 5%, 10%, 20% requestors**

As shown in Figure 5.7, when the percentage of requestors increases, the FFI results for the tree-based method stay with centralized method in general, and their growth rates are less significant than the decentralized method. In the case of 20% requestors, the centralized method has quite small FFI as it should be, but the tree-based method is about four times better than the decentralized methods.



**Figure 5.8: Average response time between 1%, 5%, 10%, 20% requestors**

Figure 5.8 shows the response time results. As indicated there, the decentralized method has best response time overall, which is natural since it only acts based on local information and avoids many control overhead. When the requests are relative low in quantity, the tree-based method has comparable response time to the decentralized method. On the other hand, when the requests are abundant, the response time for both tree-based and centralized methods grows larger than the decentralized method, although it is within the 200%-250% range. Similar results are also indicated in Figure 5.9 when measuring the average hop numbers.



**Figure 5.9: Average hop number between 1%, 5%, 10%, 20% requestors**

Figure 5.10 shows the change of FFI over time for the case of 2000 nodes with 5% managers. We measure the FFI for each of the 10 intervals. As indicated there, both the tree-based and centralized methods have their FFI stabilized quickly. This implies that requests and resources get fulfilled steadily and timely. In contrast, the growth of FFI over time for the decentralized method indicates that some peers in the network may suffer from unfair scheduling and wait longer than the others.

### FFI

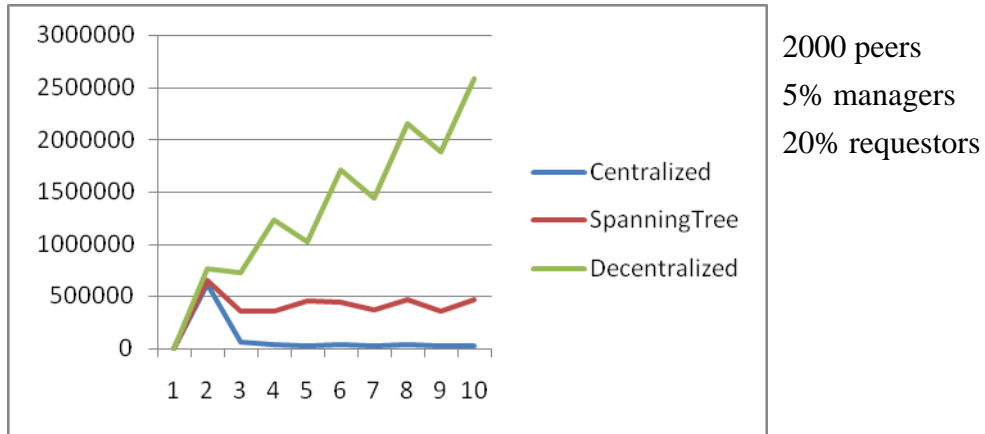


Figure 5.10: Change of FFI over time

Figure 5.11 shows an interesting observation about the distribution of requests among resource providers. Specifically, the centralized method exhibits the desirable behavior because the requests are distributed evenly among resource providers. This is not surprising because the all resource providers that become ready need to (re)enter the central queue and get fulfilled in an FIFO manner. The decentralized method, on the other hand, has the worst request distribution. This is due to the uneven request pattern generated among the requestors and the fact that requests are served by the resource providers closer to them.

### Average job number

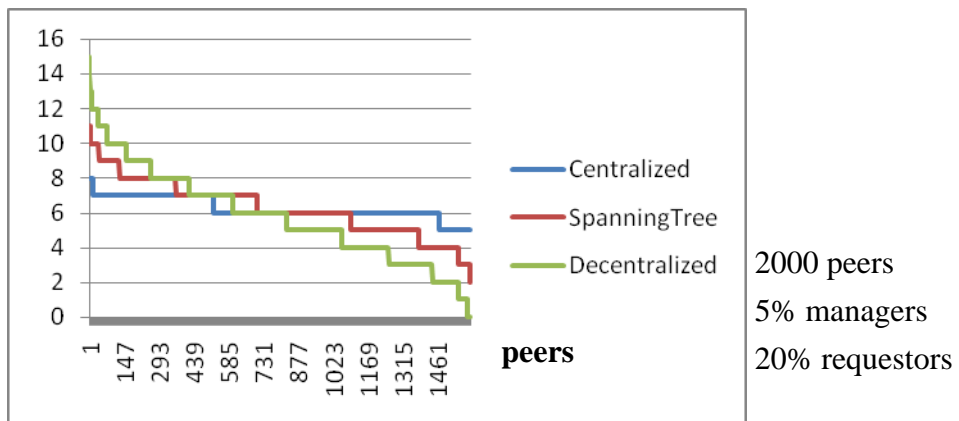


Figure 5.11: Average job number among resource providers

The next set of experiments are concerned with the impact of the manager percentage, which represents the degree of decentralization – the larger the percentage of the managers, the more decentralized the resulting network is. We investigate the cases of 5%, 15%, 50%, 75%, and 100% managers when the requesters are 20%.

Figure 5.12 shows the FFI for different methods and different manager percentages. As before, the centralized method has lowest FFI, and is relatively insensitive to the change in manager percentage. On the contrary, the decentralized method is quite sensitive to the manager percentage change, and its FFI is larger in general. In all three methods, the FFI improves when the number of managers increases.

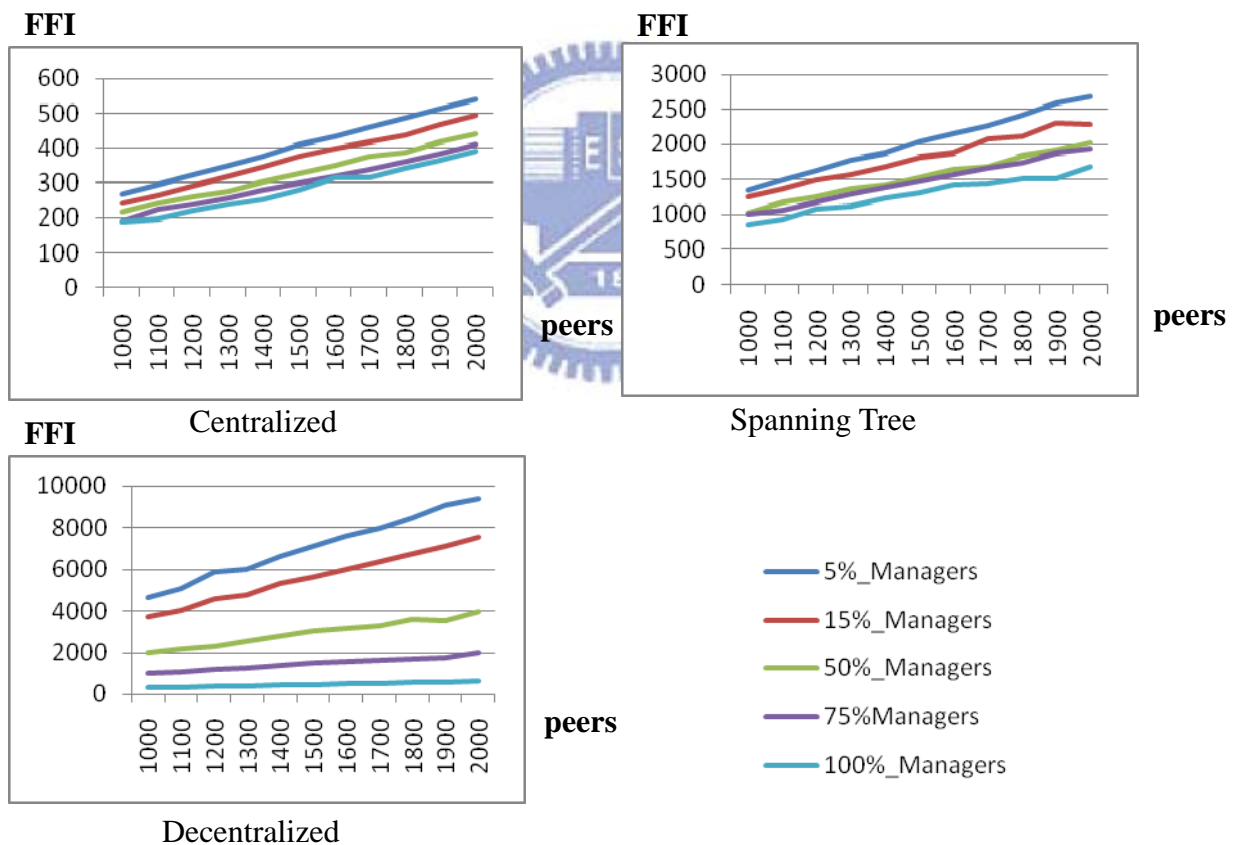
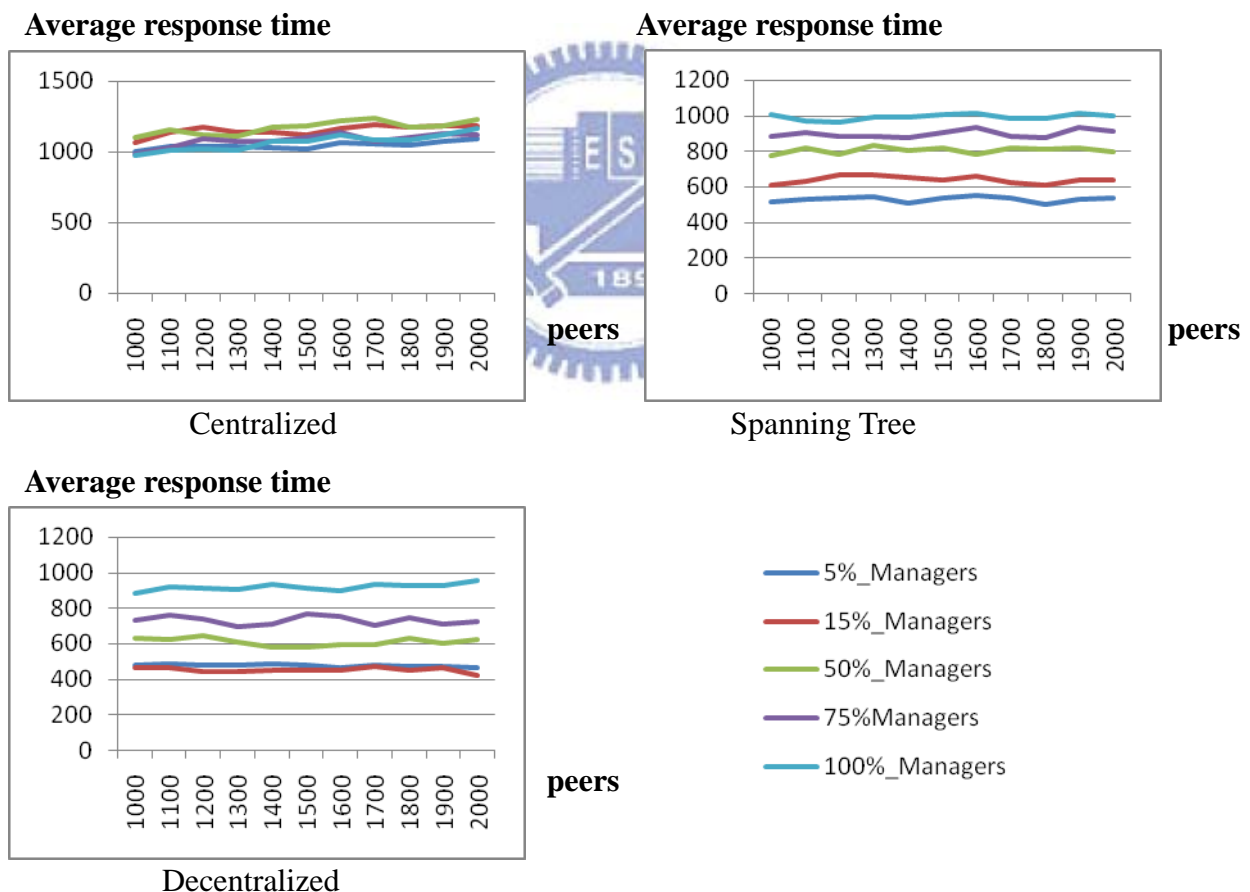


Figure 5.12: FFI for the cases of 5%, 15%, 50%, 75%, 100% managers

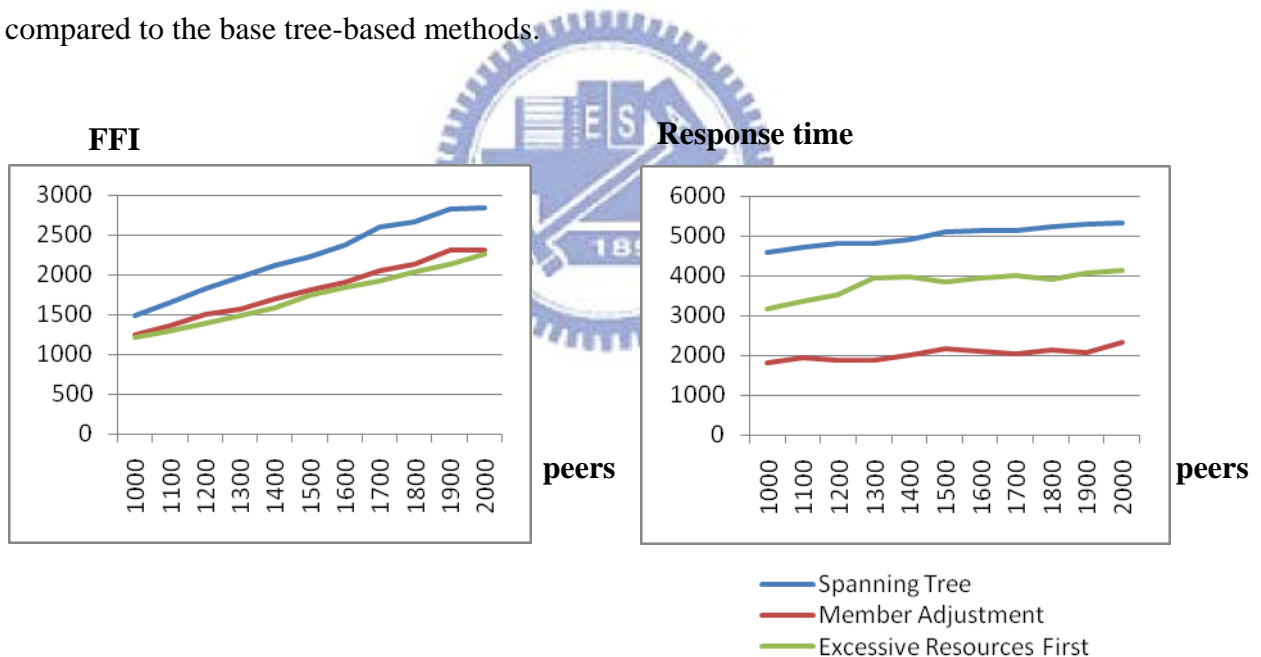
Figure 5.13 shows the simulation results of the response time. It is shown that the centralized method has similar response time under different manager percentages. The centralized method has roughly same response time. This is due to the fact that the average path length from managers to the root is roughly the same for different node sizes. On the other hand, the tree-based and decentralized methods have the response time decreased when the manager percentage decreases. This is reflected by the fact that with more managers in the network, the longer it takes to search for available resources in these two methods.



**Figure 5.13: Average response time between 5%, 15%, 50%, 75%, 100% managers**

In the following experiments we are interested in the effectiveness of some self-adaptation strategies on the scheduling. We investigate two independent approaches that have been described in the previous chapter for the tree-based method: by changing the search order among children, and by re-assigning members between a parent and its child. To better exploit the effectiveness, we also change the request generation pattern such that 10% of the requesters have higher request generation rate than normal. Here the managers are set to 15% and requesters are set to 20%.

Figure 5.14 shows both the FFI and response time for the tree variations of tree-based methods. The result shows that both self-adaptation schemes improve the FFI and response time, and in the case of member reassignment the response time dropped significantly when compared to the base tree-based methods.



**Figure 5.14: FFI and average response time**

To further appreciate the effect of the two self-adaption strategies, Figure 5.15 shows the communication overhead over time (2000 nodes, 15% managers), where the overhead represents the amount of messages for both request/resource fulfillment and network

maintenance. The results show that both self-adaption approaches can reduce unsuccessful searches, with the member reassignment approach improves the most.

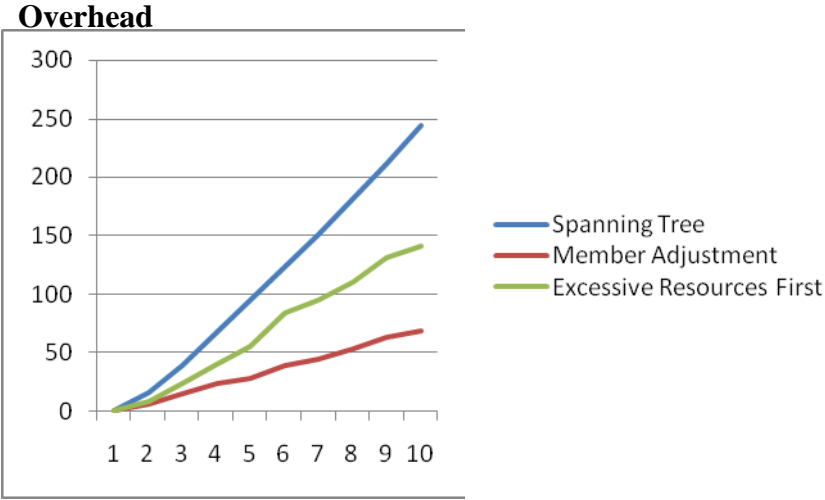


Figure 5.15: Change of overhead over time





## Chapter 6 Discussion and Future Work

It is interesting to compare DHTs and virtual queues at an abstract level. A DHT implements a virtual hash table using a set of multiple collaborating peers, and there are many approaches to implementing DHTs – the main differences being the mechanisms for requesting routing and object-peer assignment. Similarly, a virtual queue also implements a (doubly-ended) FCFS queue using multiple, distributed peers, and the goal is to meet the key requirements mentioned previously.

FCFS queues or the associated scheduling policies are not a new concept *per se* and they have been an important research topic in operating systems, parallel computing, networking, and other research fields. However, its use as a fairness measure for resource scheduling in P2P networks is uncommon. This is quite expectable for several reasons. First, to be effective, FCFS policies and other closed related policies such as least-used-first policy (when deciding which item to kick out off the cache) or earliest-starting-time-first heuristics (when scheduling jobs over multiple processors) more or less need global and timely status about the resources to be scheduled. Implementing FCFS policies in P2P networks will most likely incur unnecessary overhead.

Secondly, and probably more importantly, network-wise FCFS fairness is irrelevant in application areas such as file sharing supported by P2P systems, where resources offered by providers (who earn some credits as reward) being shared are expected to last for some time. Even for P2P networks sharing generic, uniform resources such as machine cycles, as mentioned, the usual goal is to improve job processing rate, in which case imposing FCFS fairness seems to reduce the processing rate, especially when the network grows larger and the request pattern is highly skewed.

When resource scheduling is concerned, it is interesting to compare SETI@Home with traditional systems such as operating systems, clusters, or grids, which often need to predict the performance characteristics of the participating resource providers painstakingly in order to derive a suitable execution plan, yet only to find that the predicted performance model disagrees with actual machine statistics due to machine dynamism. In the SETI@Home architecture, instead, the scheduling is done automatically by the resource providers since their act of registration indicates that they are available for the moment, fully respecting machine dynamism. Our resource sharing model bears the same idea as SETI@Home's, but generalized it in some aspects. First, unlike in SETI@Home where the central server is the one who keeps the work to be done, ours leaves what to be done to resource requestors. In addition, the central server is replaced with a set of collaborating managers that implements the virtual FCFS queue, hoping to improve locality, load balance, fault tolerance, and ultimately scalability. Despite the fact that our virtual queue may incur unavoidable communication delay due to FCFS requirements.

Interestingly, however, the FCFS fairness can play an important role in designing sound incentive mechanisms. For example, if the resource consumers and resource providers are the same set, and whether a consumer can receive resources it needs only after it has earned corresponding credits by providing matching resources. This scenario is not uncommon, and similar works have been done on file-sharing P2P systems ([1] and [11]) where a peer gets paid for providing a specific file to a remote peer, and earned credits are subsequently used for the peer to request a file at another peer. Clearly, without proper FCFS fairness, such incentive mechanisms cannot guarantee that participants with equal capability and willingness to contribute (and consume) may receive unfair treatment.

Our investigation focuses nevertheless on a narrow scope that can be outlined as follows. First, in our simplified model, jobs are uniform, that is, they are of the same type and same

processing complexity statistically. Secondly, providers are of the same processing power so that the execution time for a given job is the same when run by different providers. These assumptions are made to avoid some pathological cases. Although it is possible to drop these assumptions, doing so may raise new issues of fairness again, but they are nevertheless interesting questions that can be pursued further.

As an example, what is considered a proper pricing of processing a job? It is natural to associate prices with number of instructions and/or space used rather than by mere job counts. By distinguishing job counts from job pricing, however, the notion of fairness needs to be re-evaluated. By ensuring that each awaiting provider receives fair treatment in terms of job counts, as demanded by our fairness model addressed above, can some providers eventually earn much more than the others under certain request patterns? The problem becomes more challenging when providers can have quite diverse processing power. Although it is natural to demand a “capitalism-oriented” policy that capable providers should receive requests proportional to their processing capabilities. Again, assume all providers participate eagerly in a P2P network and all other aspects being equal, can some providers eventually earn much more than their peers in a way disproportional to their processing capabilities?

Another aspect that needs further investigation is to ensure FCFS fairness for both resource requestors and providers simultaneously. Certainly, a straightforward centralized implementation can achieve this goal, but more effective scheduling algorithms not only need to minimize additional management overhead, but also to resist as many pathological patterns of requests from both consumers and providers.

## Chapter 7 Conclusion

We have proposed a fairness model that measures the degree of preemption seen from each resource provider and consumer, and showed that this more stringent requirement makes existing approaches to resource scheduling in P2P networks unsatisfactory. To investigate the impact of this fairness criterion, we have compared several scheduling approaches, including centralized and decentralized algorithms as well as several variants of round-based algorithms based on spanning trees. The decentralized algorithms we implement choose resource providers based on local information, which can lead to unfair scheduling results for resource providers. In contrast, our decentralized algorithm and its variants search for providers along some spanning tree established among schedulers. Fairness in our approach is improved because all providers who “register” to their associated schedulers, respectively, for a given round will be served before next round begins. Simulation result shows that average response time is acceptable compared to straightforward decentralized algorithms while providing better fairness in the FCFS sense.

It should be noted that this thesis is meant as a first step towards the development of P2P networks with improved FCFS fairness. We investigate only the cases where spanning trees are used to search for resource providers, and our scheduling methods are quite reasonable and efficient, and a good tradeoff between response time and fairness. There may be other better approaches that differ from ours dramatically, possibly with (slightly) different request, resource, and network models. Further research into these possibilities is needed.

## References

- [1] P. Antoniadis, C. Courcoubetis, R. Mason, “Comparing economic incentives in peer-to-peer networks”, *Computer Networks* 46 (1) (2004) 133–146.
- [2] Ghalem Belalem and Farouk Bouhraoua, “Dynamic Strategy of Placement of the Replicas in Data Grid” Springer-Verlag Berlin Heidelberg, pp.496-506, 2007
- [3] H. Chen, S. Hariri, B. Kim, Y. Zhang, and M. Yousif, “Self-Deployment and Self-Configuration of Pervasive Network Services,” *Proceeding of the IEEE/ACS International Conference on Pervasive Services (ICPS’04)*, 2004
- [4] Clarke, I., Sandberg, O., and Wiley, B. “Freenet: A distributed anonymous information storage and retrieval system”. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*. Berkeley, CA.
- [5] Rob Cutlip, “Self-managing systems,” <http://www.ibm.com/developerworks/library/ac-selfo/>, 2005
- [6] Prithviraj Dasgupta, “Building Small Worlds in Unstructured P2P Networks using a Multi-agent Bayesian Inference Mechanism,” *AAMAS 07*, May, 2007
- [7] R. Duan, R. Prodan, and T. Fahringer, “Run Time Optimization for Grid Workflow Applications,” *Grid Computing, 2006. 7th IEEE/ACM International Conference on*, 2006
- [8] Drougas, Y. Kalogeraki, V., “A fair resource allocation algorithm for peer-to-peer overlays”, *INFOCOM 2005, Proceedings IEEE*, March 2005, Vol. 4, pp 2853-2858.
- [9] Kolja Eger, Ulrich Killat, “Fair resource allocation in peer-to-peer networks” (extended version), *Computer Communications*, 30 (2007) 3046–3054.
- [10] Globus toolkit, <http://www.globus.org/>
- [11] P. Golle, K. Leyton-Brown, I. Mironov, M. Lillibridge, “Incentives for sharing in peer-to-peer networks”, *Lecture Notes in Computer Science* 2232 (2001) 75–86.
- [12] Jinsong Han and Yunhao Liu, “Dubious Feedback: Fair or Not? ,” *Proceedings of the First International Conference on Scalable Information Systems*, 2006
- [13] S. Hariri, B. Khargharia, H. Chen, J. Yang, and Y. Zhang, “The Autonomic Computing Paradigm,” pp.5-16, 2006

- [14] T. Heinis, C. Pautasso, and G. Alonso, "Design and Evaluation of an Autonomic Workflow Engine," Proceedings of the Second International Conference on Automatic Computing, pp. 27-38, 2005
- [15] Rajendra K. Jain, *et al*, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems", Technical Report DEC-TR-301, Digital Institution Corporation, Hudson, MA 01749, September 16 1984.
- [16] D. R. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," SPAA'04, 2004.
- [17] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," IEEE Computer Society Press, pp.41-50, 2003
- [18] G. Kwon and K. S. Candan, "DANS Decentralized, Autonomous, and Network-wide Service Delivery and Multimedia Workflow Processing," Proceedings of the 14th annual ACM international conference on Multimedia MULTIMEDIA '06, pp.549-558, 2006
- [19] K. Lee, R. Sakellariou, N.W. Paton, and A. A. Fernandes, "Workflow Adaptation as an Autonomic Computing Problem," High Performance Distributed Computing Proceedings of the 2nd workshop on Workflows in support of large-scale science, pp.29-34, 2007
- [20] H. Liu, V. Bhat, M. Parashar, and S. Klasky, "An Autonomic Service Architecture for Self-Managing Grid Applications," Grid Computing, 2005. The 6th IEEE/ACM International Workshop on, pp.132-139, 2005
- [21] T.-W. J. Ngan, Animesh Nandi, and Atul Singh, "Fair bandwidth and storage sharing in peer-to-peer networks". In First IRIS Student Workshop, Cambridge, MA, Aug. 2003.
- [22] J. Nichols, H. Demirkan, and M. Goul, "Autonomic Workflow Execution in the Grid," Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions, pp.353-364, 2006
- [23] I. Foster, Argonne & U.Chicago, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich, "The Open Grid Services Architecture, Version 1.0," 2005
- [24] K. Ohnishi, H. Yamamoto, K. Ichikawa, M. Uchida, and Y. Oie, "Storage Load Balancing via Local Interactions Among Peers in Unstructured P2P Networks," Proceedings of the First International Conference on Scalable Information Systems, 2006.
- [25] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies", *ACM Computing Surveys*, Vol. 36, No. 4, December 2004, pp. 335–371.

- [26] A. Ranganathan and R. H. Campbell, "Self-Optimization of Task Execution in Pervasive Computing Environments," Proceedings of the Second International Conference on Autonomic Computing (ICAC'05), 2005
- [27] RATNASAMY, S., FRANCIS, P., HANDLEY, M., AND KARP, R. "A scalable content-addressable network". In Proceedings of SIGCOMM 2001.
- [28] Raynal, M. and Singhal, M., "Logical time: capturing causality in distributed systems", *Computer*, Vol. 29, Issue 2, Feb 1996, pps: 49-56
- [29] ROWSTRON, A. AND DRUSCHEL, P. 2001. Pastry: Scalable, "distributed object location and routing for large-scale peer-to-peer systems". In Proceedings of IFIP/ACMMiddleware. Heidelberg, Germany.
- [30] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," SIGCOMM'01, 2001
- [31] M. Strohmaier and E. Yu, "Towards Autonomic Workflow Management Systems," IBM Centre for Advanced Studies Conference, Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, 2006
- [31] J. Yan and Y. Yang, "SwinDeW-A p2p-Based Decentralized Workflow Management System," IEEE TRANSACTIONS ON SYSTEMS, MMAN, AND CYBERNETICS-PART A: SYSTEMS AND HUMANS, 2004
- [32] M. Yu, P. Liu, and W. Zang, "Self-Healing Workflow Systems under Attacks," Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04) ICDCS '04, pp.418-427, 2004
- [33] Yingwu Zhu and Yiming Hu, "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems", Transaction on Parallel and Distributed Systems, V. 16, No. 4, 2005. pp 349-361.