Fair Resource Allocation in P2P Systems: Theoretical and Experimental Results

by

Paraskevi Raftopoulou

A thesis submitted in fulfillment of the
requirements for the degree of

Master of Computer Engineering

Technical University of Crete
Department of Electronic and Computer Engineering

Intelligent Systems Laboratory

June 2003

Abstract

We study the problem of fair allocation of resources among agents with a special emphasis to the case of peer-to-peer systems. At first, we study three measures of fairness for resource allocation (the fairness index, majorization and max-min fairness) and analyze their relationship. Then, we consider a very simple model of resource allocation where we have a single resource that comes in indivisible quantities represented by integers, and needs to be allocated fairly among n agents. We adopt the fairness index as our measure of fairness for this problem, and study centralized algorithms for its solution. It turns out that our problem is very close to number partitioning, one of the six basic NP-complete problems in the standard book by Garey and Johnson. We discuss the relationship of these two problems and devise a greedy algorithm that runs in polynomial time and a complete anytime algorithm that runs in exponential time. We study the phase transition behaviour of these algorithms and demonstrate that the greedy one actually performs very well and returns almost perfectly fair allocations.

Finally, we consider the hierarchical P2P content-sharing system CLUSTER and revisit its resource allocation algorithms in the light of the above results. We show how to devise a centralized complete anytime algorithm for a more complicated resource allocation problem in CLUSTER: fair allocation of load. We study the phase transition behaviour of this algorithm and compare it with the greedy algorithm MaxFair, proposed by the inventors of CLUSTER.

Acknowledgements

I would like to thank my supervisor Manolis Koubarakis for his guidance and precious advice. Manolis introduced me to new research areas and being always there helped me to broaden my knowledge boundaries. His comments and corrections were always up to the point showing me the way. His belief in me was a continuous motivation for improvement.

I would also like to thank Richard Korf for sending me his C code on 2-way and 3-way number partitioning. I thank Ian Gent and Toby Walsh for sending their Lisp code and being always available to answer my questions regarding number partitioning phase transitions.

I would also like to acknowledge the financial support received from project DIET in the context of which this work was carried out.

I am grateful to the rest of my fellow students in the group, Theodoros Koutris, Christos Tryfonopoulos, Anni Xiruhaki, Georgia Adamopoulou, Stamatis Andrianakis and Matoula Magiridou, for always making the hours at the office so pleasant. Their encouragement was precious to me. Special thanks to Thodoros and Christos for their collaboration and valuable comments, and especially Matoula for her contribution in Chapter 3.

My soundly thanks go to my family for their endless love and encouragement throughout this long but rewarding journey. I own them my existence, my way of thinking and my will for continuous progress. I dedicate to them all my achievements and I hope they are proud of me.

I want to profoundly thank a special person. Nothing would be the same if Christos was missing out of my life. He was always there lightening my days, giving breath to my moments... I thank him for offering me so generously his love.

Finally, my deepest thanks go to Amalia, Costas, Georgia, Maria, Basiliki, Nikos, Sotiris, Petros and Stamatis. They offered me ungrudgingly their friendship. I thank them all for being there to all my joys and anxieties.

This dissertation is dedicated to my mother Aggeliki,
my father Costas and my sister Katerina.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem of resource allocation in peer-to-peer (P2P) systems is a difficult and important one. If we take a macroscopic view of P2P networks, it is not difficult to appreciate the degree of *complexity* involved in the organization and management of their resources and the services they provide. There is a huge number of autonomous peers making up the network. The network is open and peers can connect and disconnect from it at any point in time. Peers are very heterogeneous as witnessed by a recent study of Napster and Gnutella [SGG02] and belong to users with possibly conflicting interests. P2P network users are self-interested and should not be expected to be cooperative. They will act in order to satisfy their own goals and these goals might conflict with various notions of "common good". P2P users might collude to take advantage of the system resources; they might even be malicious and try to attack the system by exploiting its vulnerabilities.

Some of these observations have been made in the nineties for other complex distributed systems and networks and various algorithms for resource allocation have been proposed. Existing approaches to resource allocation in distributed systems and networks can be usefully classified into various dimensions by asking the following questions:

- What are the goals of resource allocation?
- What are the technical characteristics of the solution proposed?

Typically, work on resource allocation has several goals including:

- Fair access to system resources.
- Scalable access to resources by users and applications.
- High quality of service as given by parameters such as average response time, throughput, etc.

- Efficient, scalable, robust and fault-tolerant approaches for performing the resource allocation itself.

Regarding the technical details, one can distinguish between *centralized* and *decentralized* approaches to resource allocation. In the former case researchers usually assume that complete information about the system is available and propose an optimization algorithm for the task at hand. Such algorithms are usually based on mathematical models and techniques from *theoretical computer science* and/or *operations research* and, in a real system, they will be run by a centralized dedicated server. Relying on a single dedicated server introduces a single point of failure into the system and does not scale well. Then, in the typical distributed systems solution to this problem, one introduces a cluster of servers organized in a hierarchical or P2P fashion. Various sophisticated protocols facilitate communication between cluster members and ensure that the whole system appears to its users as a single server but at the same time offers scalability and performance guarantees.

In the decentralized approach to resource allocation, researchers recognize that resources in a distributed system are owned by autonomous nodes that make local decisions under limited information, thus a decentralized solution that takes this autonomy of nodes explicitly into account is preferable. From a researcher's point of view it is usually a good strategy to investigate both centralized and decentralized approaches so that "perfect" centralized solutions offer a good benchmark against which "imperfect" decentralized solutions can be measured.

Distributed algorithms for resource allocation problems can further be distinguished into the ones based on *consensus* [Lyn96] and the ones based on *microeconomics* or *game theory* [FNSY95]. Consensus-based algorithm (e.g., for a replicated file system) essentially concentrate on protocols and algorithms for reaching agreement among processes (e.g., storage nodes) in the presence of failures. There has been considerable work in this area which is nicely surveyed in Lynch's standard textbook [Lyn96].

Microeconomic algorithms for resource allocation exploit the fundamental similarities between distributed computer systems and human economies. In both cases, we have *producer agents, consumer agents* and *resources* to be allocated. Agents are autonomous and they base their decisions on local information only. Resources are scarce and need to be allocated to these agents that value them most. There has been much interesting research on microeconomic algorithms for resource allocation in distributed systems and networks since the eighties by researchers in distributed systems and networks [FNSY95] and distributed AI [SK97]. These algorithms can be categorized along various dimensions: co-operative vs. competitive, game-theoretic vs. based on prices (market mechanisms) and so on. The interested reader can consult [FNSY95] for a nice survey. The trend to use ideas from economics in the study of resource allocation

in distributed systems and networks has continued until recently. See for example the paper [GCL02] where ideas from game theory are used to cope with the problem of job allocation among heterogeneous computer systems, or the paper [CDS01] where bandwidth over paths in a network is sold to interested parties using auctions.

In the last few years there has also been a lot of interest in exploring the boundaries of computer science, game theory and economics by researchers with backgrounds in theoretical computer science, artificial intelligence and distributed systems [NR99, Pap01, San02]. This interest has been largely generated due to the importance of the Internet, the Web and E-commerce. In this context the notion of *algorithmic mechanism design* [NR99] is particularly important and provides a new foundation for resource allocation algorithms in distributed computer systems. The proponents of this approach suggest that in the Internet era computers should be modelled as *non-cooperative selfish agents* expected to manipulate network protocols and algorithms for their owner's benefit. To account for this non-cooperative selfish behaviour, Nisan and Ronen propose to redefine traditional optimization problems for resource allocation to include an explicit notion of *payments* that acts as an incentive for agents to behave cooperatively [NR99].

1.1 Economics-Inspired Resource Allocation in P2P Networks

The approaches to resource allocation in distributed systems and networks sketched above can be used as a basis for developing similar techniques in P2P networks. This could actually be seen as a re-birth of distributed systems and networks research: P2P interaction was the original main idea of distributed systems and networks that somehow got lost with the emphasis on client-server architectures.

In this dissertation we would like to advocate the use of ideas from *human economies* as a useful metaphor for studying resource allocation in P2P systems. Our arguments are exactly the same as the ones used by proponents of these ideas in distributed systems and networks e.g., [Fer89, FNSY95]. Reiterating what we said above P2P networks consist of a huge number of autonomous peers at the "edges of the Internet". All the resources of interest to the network are made available by the peers: storage space, bandwidth, computing cycles, human presence, etc. Peers are very heterogeneous in their capabilities and belong to users with possibly conflicting interests. Peers are self-interested and should not be expected to be cooperative. They will act in order to satisfy their own goals and these goals might conflict with various notions of "common good". Peers might collude to take advantage of the system resources; they might even be malicious and try to attack the system by exploiting its vulnerabilities.

The following open research questions remain to be tackled in this area:

- What mathematical models are appropriate for characterizing formally resources and their use in P2P networks?
- What fundamental concepts of micro-economics and game theory are appropriate for modelling resource allocation problems in P2P networks?
- What algorithms for resource allocation are appropriate in these contexts? What is the computational complexity of these algorithms? How do these algorithms perform on real or simulated data? What phase transition behavior do they exhibit?
- How can these algorithms be implemented efficiently in various architectural alternatives for P2P systems? How can these algorithms be implemented efficiently in nature-inspired environments for developing P2P systems (e.g., the DIET core [HWBM02] or Anthill [BMM02])?
- How does trust and reputation information affect the behavior of a selfish peer regarding the sharing of its resources? For example, should information about trust and reputation be used to price peer resources or to determine the amount of resources a peer makes available to another peer? What are the dynamics of the resulting system? Will co-operative "communities of trust" emerge?

Some of these questions are studied in this thesis.

1.2 Contributions

The contributions of this dissertation are the following:

- We survey recent related work in the area of P2P systems. This allows the reader of this thesis to see our work in an appropriate context. In our survey, we pay particular attention to measurement studies that reveal interesting characteristics of existing P2P systems, work on load balancing for P2P networks, and the use of ideas from economics to solve resource allocation problems in P2P networks.
- We study three measures of *fairness* for resource allocation in a general context where we have agents and resources to be allocated among them. The measures we consider are the following: the fairness index [JCH84], majorization [MO79] and max-min fairness [BG87]. We give detailed presentations of these measures and comment on their relationships.

- We consider a very simple model of resource allocation where we have a *single resource* that comes in indivisible quantities represented by integers, and needs to be allocated *fairly* among n agents. We adopt the fairness index as our measure of fairness for this problem, and study centralized algorithms for its solution. It turns out that our problem is very close to *number partitioning*, one of the six basic NP-complete problems in the standard book by Garey and Johnson [GJ79]. We discuss the relationship of these two problems and devise a greedy algorithm that runs in polynomial time and a complete anytime algorithm that runs in exponential time. Both algorithms are based on number partitioning algorithms originally proposed by Karmakar and Karp [KK82] (greedy) and Korf [Kor98] (complete). We study the phase transition behaviour of these algorithms and demonstrate that the greedy one actually performs very well and returns almost perfectly fair allocations.

- We consider the hierarchical P2P content-sharing system CLUSTER proposed in [TXK02] and revisit its resource allocation algorithms in the light of the above results. We show how to devise a centralized complete anytime algorithm for a more complicated resource allocation problem in CLUSTER: fair allocation of load. We study the phase transition behaviour of this algorithm and compare it with the greedy algorithm MaxFair presented in [TXKN03].

1.3 Organization

The organization of the thesis is as follows. The next section surveys related work. Section 3 presents three measures of fairness. Section 4 discusses algorithms for a simple fair resource allocation problem. Then, Section 5 builds on the developments of Section 4 and studies fair allocation of load in the P2P system CLUSTER. Finally, Section 6 presents our conclusions and discusses future research.

Chapter 2

Related Work

In this section we survey work carried out in the area of P2P systems that is mainly connected to the topic of this dissertation. We start with a quick introduction to P2P computing by discussing the first three influential P2P file sharing systems (Napster, Gnutella, and Freenet) together with more recent attempts to build P2P system infrastructure based on distributed hash-tables.

We also survey measurement studies of existing P2P systems to understand the "laws" that govern their structure and dynamics, the patterns of user-system intersections, the characteristics of participating hosts etc. The results of these studies are taken into account in Chapter 5, where detailed simulations of a particular P2P system design are conducted.

Finally, we survey recent economics-inspired research addressing various open issued in P2P networks. This thesis should be seen to fall most closely into this area of related research.

2.1 P2P Computing

In P2P systems a very large number of autonomous computing nodes (the *peers*) pool together their resources and rely on each other for *data* and *services*. P2P systems are application level *virtual* or *overlay networks* that have emerged as a natural way to share data and resources. Popular P2P systems such as Napster[1] (now in a transition state), Gnutella[2], Freenet[3], Kazaa[4], Morpheus[5] and others have made this model of interaction popular. Ideas from P2P computing can also be applied to other areas beyond data sharing such as Grid computation (e.g.,

[1] `http://www.napster.com`
[2] `http://gnutella.wego.com`
[3] `http://freenet.sourceforge.net`
[4] `http://www.kazaa.com`
[5] `http://www.musiccity.com`

SETI@Home[6] or DataSynapse[7]), collaboration networks (e.g., Groove[8]) and even new ways to design Internet infrastructure that supports sophisticated patterns of communication and mobility [SAR+02].

The wealth of business opportunities promised by P2P networks has generated much industrial interest, and has resulted in the creation of various research and industrial projects[9], startup companies, and special interest groups[10]. Researchers from distributed computing, networks, multi-agent systems and databases have also become excited with the P2P vision, and papers tackling open problems in this area have started appearing in high-quality venues (such as ICDCS, SIGCOMM, INFOCOM, CIDR, SIGMOD, VLDB, etc.). Indicatory of the recent interest in P2P computing are specialised conferences and workshops collocated with major conferences from different areas[11].

2.1.1 Three Influential P2P Systems: Napster, Gnutella and Freenet

In this section we discuss the first three systems that popularized the P2P paradigm: Napster, Gnutella and Freenet. These three systems have a very similar goal: to facilitate the discovery and sharing of files (e.g., images, audio and video) among a large set of *peers* (user computers) located at the "edge of the Internet". The files to be shared are stored at the peers, and after being discovered by an interested party, they are downloaded using a protocol similar to HTTP. But beyond this basic goal, there are important differences among the three systems regarding the *metadata* kept at each network node, the *topology* of the P2P network, the *placement* of the shared files, the *routing algorithms* for queries and replies, and the degree of privacy offered to its users.

Napster

Napster was the first of the big three P2P systems to be released and is now in a transition state due to legal problems. In Napster, a large cluster of dedicated servers owned by the respective company maintain a *metadata index* that keeps

[6] `http://www.setiathome.ssl.berkeley.edu`

[7] `http://www.datasynapse.com`

[8] `http://www.groove.net`

[9] See the European projects DIET (`http://www.dfki.de/diet`) , BISON (`http://www.cs.unibo.it/bison/`), MMAPPS (`http://www.mmapps.org`), SWAP (`http://swap.semanticweb.org/`) and the recent US project IRIS (`http://iris.lcs.mit.edu/`).

[10] For example, see the activities of the P2P working group of the Global Grid forum at `http://www.gridforum.org`.

[11] For example, see the 1st and 2nd IEEE International Conference on P2P Computing (`http://www.ida.liu.se/conferences/p2p/p2p2002`), the International Workshop on Agents and P2P Computing (`http://p2p.ingce.unibo.it`), the 1st and 2nd International Workshop on Peer-to-Peer Systems (`http://iptps03.cs.berkeley.edu`) and so on.

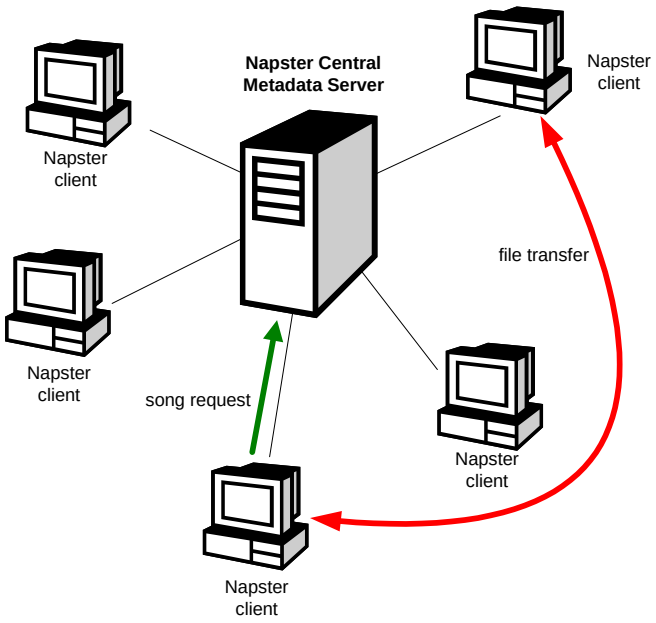

Figure 2.1: An overview of Napster's architecture

track of active peers, descriptions of the files they are willing to share, and certain quality of service parameters such as bandwidth and duration of the connection. Peers (clients!) connect to the system by connecting to *one* of these dedicated servers and publishing a description of the files they want to share with other peers. Peer queries are sent to their selected server which returns a list of matching files, the address of the peer that each file is stored to, and other important information known about the peer (e.g., bandwidth as reported by the peer). Peers can then *directly* download selected files from the peer of their choice.

The Napster architecture is thus partially centralized (see Figure 2.1 for a high level view of Napster's architecture). The network of Napster servers acts as a single point for all user queries; it points users to the peers who have the files they want. Once the files are downloaded into the requesters machine, they are automatically shared, making them more available to the overall user community.

Gnutella

On the opposite end of the spectrum of decentralization, Gnutella has *no centralized servers*. Each node in a Gnutella network is a peer that can act as a client and as a server at the same time (see Figure 2.2). Gnutella peers form an *overlay network* by setting up connections to peers of their choice. Addresses for connecting to the Gnutella network initially can be found by the interested user e.g., by consulting web pages such as `gnutellahosts.com` or `router.limewire.com`). Gnutella offers primitives *ping* and *pong* for discovering parts of the network and

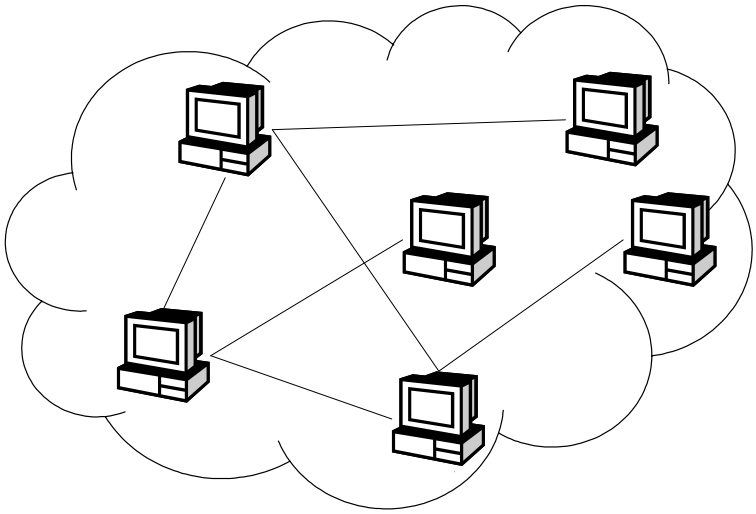

Figure 2.2: A high level view of Gnutella's architecture

facilitate its maintenance while peers enter and leave the system.

To discover a file, a Gnutella source node issues a query such as "I am interested in MPEGs with video-clips of Jennifer Lopez" to its neighbors with whom it has open connections. The query is accompanied by a time-to-live (TTL) counter that specifies how many hops this query is allowed to travel in the Gnutella network. Each node that receives this request processes it using its local file collection and returns URLs pointing to matching files to the requesting node. Then this node decrements the TTL counter of the request by one. If the value of the TTL counter is more than 0, then this node forwards the query to its neighbours. This process repeats itself and eventually more pointers to matching files are returned to the source of the request. Thus, Gnutella uses the classical *flooding* technique known from Computer Networks [BG87] for routing queries. Replies reach the source node by travelling along the reverse path followed by the query because a node in a Gnutella network does not know the identity of the source node issuing a request that it might process. However, the privacy of information requesters and providers is not really protected in any serious way (e.g., Gnutella messages contain IP addresses, and URLs are returned to information requesters so that they can retrieve the files they desire). Since its original proposal, the inefficiencies of this basic Gnutella protocol have carefully being studied and many proposals for more efficient search in P2P networks are now in the literature (see [DGMY03] for a survey).

Freenet

Freenet[12] is a P2P network of nodes connected to each other for the purpose of sharing information in the form of data files [CHM+02]. Like Gnutella, Freenet keeps a *completely decentralised architecture* which ensures scalability, robustness and fault-tolerance. At the same time, Freenet goes a long way in ensuring the survivability of published information, the adaptability to usage patterns and the protection of the anonymity of information providers and consumers and holders (these features are what distinguish Freenet very much from Napster and Gnutella).

Every Freenet user runs a node that provides the network with some storage space. To add a file, a user sends the network a message containing the file and an assigned location-independent *globally unique identifier (GUID)* which is computed using a SHA-1 secure hashing function. Each GUID consists of two parts: a *content-hash key* which is obtained by hashing the contents of the file and it is used for low-level data storage, and a *signed-subspace key* intended for higher-level human use like traditional filenames.

To retrieve data, a user of Freenet sends to the network a request message and the GUID of the file. Whenever a node receives a request, checks its local data-store first. If the file is found, the node returns it to the requester together with a tag identifying itself as the holder. If the file is not found, the routing table is consulted, *one* of the neighbour nodes with the closest matching key is chosen, and the request is forwarded to it. This is a basic difference with the Gnutella algorithm which does not perform local search and would send the query to *all* neighbour nodes. When the data file is finally found, it is returned to the requester via the same path. Additionally, intermediate nodes save an entry in their routing table associating the requested key with the data source. Depending on their distance from the holder, each node might also cache a local copy of the data file.

The anonymity of a file producer is ensured by having intermediate nodes occasionally altering the holder tags to point to themselves as data holders. This does not compromise discovery of the file later because the identity of the true data holder is kept at the node's routing table and routing tables are never revealed. The anonymity of an initiator of a query is also ensured since a node cannot know whether its neighbour node is the one interested in the results of the query or is simply forwarding a message.

Each data request in Freenet is given a TTL count (like in Gnutella), which is decremented at each node the request goes through successfully in order to reduce message traffic. To prevent requests from going into an infinite loop, Freenet assigns a unique identifier to each request so that a node will never forward a request that goes through it for a second time.

Insert messages follow the same procedure that a request message for that file

[12] `http://freenet.sourceforge.net`

would take thus routing tables are updated in the same way and files are stored in exactly the nodes where queries will go looking for them [CHM+02].

2.1.2 Distributed Hash-Tables

A fundamental problem that is confronted to all P2P systems is the efficient location of a data item. Locating the nodes that stores the video clip of my favourite singer, seems to be a core operation in all file-sharing P2P systems. Several solutions were proposed by the designers of these systems, including a centralised location mechanism for the case of Napster and a flooding approach for the decentralised Gnutella. It is clear that neither of these techniques scales well to face the problem created from the hard task to locate the owner of a file. With this in mind, two systems that claim to solve this location problem were proposed. Both systems use *distributed hash tables* to create a scalable indexing mechanism that supports one basic operation: map a given *key* to a node in the P2P system.

In this section we will discuss CAN and Chord, two representative examples of DHTs, and look into the details that are directly connected to the interests of this dissertation.

CAN

CAN (Content-Addressable Network) uses a *virtual d-dimensional* Cartesian coordinate space on a d-torus for some fixed d. This virtual coordinate space is dynamically partitioned among all nodes in an overlay network that are used to store data items of the form $(key, value)$. To store a pair $(key, value)$, key is deterministically mapped onto a point p in the coordinate space using a uniform hash function. Then $(key, value)$ is stored at the node that owns the zone within which point p lies.

Each node in CAN holds a routing table that gives the IP address and virtual coordinate zone of each of its neighbors (neighborhoods are defined appropriately in a geometric fashion). When a CAN node receives a request for a particular data item that it doesn't have, it greedily forwards this request towards any node whose zone coordinates are closest to this item. Thus, each peer in CAN maintains $O(d)$ states, and the lookup cost is $O(dn^{1/d})$ for d dimensions and n nodes.

If a new node A wants to join CAN, it must first find the IP address of any node B currently in the system (e.g., as in YOID [Fra00]). Then, it randomly chooses a point p in the coordinate space and sends a JOIN request destined for point p through B (and CAN's routing machinery). When the JOIN message reaches the node C responsible for the zone in which p lies, C splits its zone in half and assigns one half to node A. The exact way to achieve this splitting is

discussed in [RFH+01]. Finally, routing tables in the network are updated to take account of the new node A for a cost of $O(d)$.

Beyond the basic functionality discussed above, [RFH+01] presents mechanisms for dealing with node departure and recovery under certain failure scenarios. Since the above complexity bounds refer to hops in the CAN overlay network and not to IP-level hops, [RFH+01] discusses some techniques for minimizing lookup latency with the goal to make it comparable to the underlying IP path latencies.

Regarding this thesis, we are particularly interested in the contribution of CAN and other DHT proposals discussed below to load balancing. In Chapters 4 and 5 we discuss load balancing in P2P systems as an application of our metrics and algorithms for fair resource allocation. [RFH+01] briefly discusses some very simple load balancing mechanisms e.g., more uniform partitioning of the coordinate space by splitting the node with the maximum volume among p and its neighbors when a new node joins (see the discussion on joins above). To deal with the added complexity of differing popularities of data items, [RFH+01] proposes standard caching and replication techniques as studied for the case of Web "hotspots" [RS02].

Chord

Chord [SMK+01, SMLN+03] is a scalable P2P lookup service that tries to simplify the design of content sharing systems. Similarly to CAN, data items in Chord are represented by a key-value pair, using a hash function to map the key of an item to a peer. Thus, when a user is interested for a data item, Chord provides a lookup mechanism that retrieves the peer that is responsible for this item. In contrast to CAN, the information is maintained by a Chord peer and the lookup mechanism depends only on the number of peers in the system.

To map data items to network peers, Chord assigns each node and key an m-bit identifier using a base hash function such as SHA-1 [KLL+97, Lew98] (node hashes are generated using their IP address). The parameter m must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible. This type of hashing is specified as *consistent hashing* within Chord. Consistent hashing works as follows. Identifiers are ordered in an *identifier circle* modulo 2^m. A data item with key k is then assigned to the first node whose identifier is equal to or follows k in the identifier space. Nodes are connected to their neighbors in the identifier circle and searching for a key amounts to traversing this circle. In Chord, faster searches are enabled through additional edges in the identifier circle. This is implemented by having each node maintain a small amount of such routing information in a so-called "finger table". If there are n nodes in the network, each one of them holds information about $O(\log n)$ other nodes in its finger table. This enables lookup operations to require only $O(\log n)$ messages. Additionally, the Chord network can be efficiently adjusted

while nodes join or leave the system: only an $O(1/n)$ fraction of items are moved to different nodes and $O(\log^2 n)$ messages are required to update routing tables.

Similarly to CAN, Chord makes only a limited effort to address the load balancing problem. Consistent hashing in its simplest form might spread data items *unevenly* over the network nodes [SMK+01, SMLN+03]. The inventors of consistent hashing [KLL+97] and the Chord authors [SMK+01, SMLN+03] propose the use of *virtual nodes* to eliminate this problem. Using this technique each node is assumed to run a number of virtual nodes and keys are mapped to virtual nodes by the hash function. [KLL+97] show that having $O(\log n)$ virtual nodes per real node results in almost perfect load balance. But since it is hard to know the value of n beforehand, [SMK+01, SMLN+03] propose to assume an a-priori bound on the number of nodes of the system. The experiments in [SMK+01, SMLN+03] show that good load balancing is achieved with the use of virtual servers but the problem of differing popularities of data items is not considered.

Load Balancing efforts in DHTs

[BCM03] suggest the application of the "power of choices" paradigm [MRS01] to load balancing in DHTs. According to this approach, nodes are mapped to identifiers in Chord's identifier circle using a hash function h_0. However, data items are mapped to positions in the circle by considering d choices given by hash functions $h_1, \ldots, h_d$. Thus if a node wants to insert a data item x in the network, it first calculates $h_1(x), \ldots, h_d(x)$. Then, it performs d lookups in parallel to query the load of nodes responsible for these hash values. Finally, the data item x is assigned to the node having the smallest load and the rest of the nodes keep a *redirection pointer* to this node. If we want to search for an item x, we can choose a hash function $h_j,\ 1 \leq j \leq d$ at random and contact node p_j responsible for the value $h_j(x)$. If p_j does not have x, the the query is redirected to the correct node p_i using the redirection pointer for x stored at p_j.

[BCM03] presents experiments where they compare the load balancing strategy of Chord based on $O(\log n)$ virtual nodes or on an unlimited number of virtual nodes with the load balancing strategy based on the "power of two choices" (i.e., $d = 2$). These experiments show that the two choices scheme presented above achieves small variations in the load assigned to each node just like the other two strategies. However, this scheme also achieves lower *maximum loads* and it is thus preferable to avoid situations where nodes fail due to excessive load.

[RLS+03] is the most comprehensive attempt to address the load balancing problem in DHTs adopting the concept of virtual nodes. The main idea in [RLS+03] is to balance the load by moving virtual nodes from heavy nodes to light nodes. A node i is called *heavy* if its load L_i is greater than its *target load* T_i. Otherwise, it is called *light*. If a node i has capacity C_i then its target load is

set to

$$T_i = (\frac{\Sigma_{i=1}^{n} L_i}{\Sigma_{i=1}^{n} L_i} + \delta)C_i$$

where δ is a slack variable showing how close to the ideal value

$$\frac{\Sigma_{i=1}^{n} L_i}{\Sigma_{i=1}^{n} L_i} C_i$$

we would like to get. When a virtual node v is moved from a heavy node h to a light node l, one should be careful to choose v so that the minimum amount of load is moved that makes h light without turning l into a heavy node.

Under the above assumptions, [RLS+03] discusses the relative merits of three load balancing schemes:

- *One-to-one* scheme. In this scheme, a light node chooses a random key and performs a lookup operation to find the node responsible for that key. If this node is a heavy node, then a transfer of load takes place among the two nodes. Light nodes are the only ones that perform this probing operation periodically; no such operation is expected from heavy nodes.

- *One-to-many* scheme. In this scheme, a heavy node h actively seeks out $k > 1$ light nodes $l_1, \ldots, l_k$ in the network. For each light node l_i, we carefully choose a virtual node v_i to be exchanged as we discussed above. Among the nodes $v_1, \ldots, v_k$ chosen, we select the heaviest one to exchange with one from h.

  This scheme is more complex to implement than the one-to-one scheme. [RLS+03] proposes to introduce d *directories* (where $d << n$) that store load information about the light nodes. Directories are also stored in the DHT.

- *Many-to-many* scheme. This scheme consists of three phases. In the *unload* phase, each heavy node starts transferring its heavy virtual nodes greedily into a global *pool* until it becomes a light node. Then the system enters the *insert* phase with all its nodes being light and a global pool of heavy virtual nodes awaiting to be transferred. In this phase, we first order the heavy virtual servers in descending order of weight. Then we repeatedly choose the heaviest virtual node in the pool and transfer it to a light node k chosen so that $T_k - L_k$ is minimized subject to the condition that $T_k - L_k \geq load(v)$ (this is a standard *best-fit* heuristic). We continue this phase until the pool becomes empty or no more nodes can be transferred. In the former case the algorithm terminates while in the latter the algorithm enters the *dislodge phase.* In the dislodge phase we take the heaviest virtual node v in the pool and try to find a node with a virtual node v' of a light node i such that

  $$load(v') < load(v) \text{ and } L_i + load(v) - load(v') \leq T_i.$$

If no such node can be found the algorithm terminates, otherwise it returns to the insert phase (hoping that insert will work for the next node in the pool which is lighter than the one just swapped).

[RLS+03] propose to use directories to implement this algorithm in a distributed way.

[RLS+03] compares the above three schemes using simulations under Gaussian and Pareto distributions of load. The experiments presented concentrate on measuring the *total load* that needs to be reallocated to reach a balanced state and the *number of rounds* needed to converge to a state where all nodes are light. The results show that the total load moved does not depend on the scheme used, but rather on the distribution of the load, with all schemes performing worse under the Pareto distribution. The many-to-many scheme is shown to be superior to the other two, converging to a successful state within approximately 50 rounds in as many as 94% of the cases tried.

Before we closing this section, let us point out that the above study of the three pioneer systems Napster, Gnutella and Freenet and of the DHT proposals CAN and Chord, hardly represents a good survey of the area of P2P networks. We have not discussed the well-known popular super-peer networks [YGM03], other hierarchical designs such as [TXK02, TXKN03], how to implement more complex services such as multicast on top of lookup mechanisms such as the ones provided by CAN and CHORD [RHKS01] etc. For more detailed surveys the reader should consult [MKL+01, DGMY03].

2.2 Modelling and Measuring: Uncovering the "Laws" of P2P networks

Recently, researchers from computer science and physics have tried to understand the "laws" governing the current state of *information networks* such as the Internet and the Web, and compare them with *social networks* studied by psychologists and social scientists [FFF99, BA99, Kle00, KRRT02]. This flurry of research activity has resulted in various useful characterizations of the Web starting from the basic modelling assumption that the Web is a *directed graph* with pages representing nodes and links among pages representing edges. The results of these studies are currently actively exploited for the design of better crawlers, search engines and portals, and for understanding the emergence of new social phenomena in the Web. We believe that similar large-scale modelling and measurement exercises must be undertaken for P2P networks. With P2P traffic accounting for around 30% of the total Internet traffic according to recent studies[13], these exercises are timely and can result in a better understanding of these

[13] `http://wwwstats.net.wisc.edu`

popular information technologies and their role in our society. They can also facilitate new advances in algorithms, architectures and implementations that will lead to a new generation of P2P systems.

2.2.1 Measurements and Graph Models for the Web

Barabasi and Albert have studied a 325 thousand page subset of the Web and discovered that the distribution of degrees of nodes (especially *in-degrees*, the number of links pointed to a certain page) follows a *power-law*: the probability that a node has in-degree i is directly proportional to $1/i^k$ where $k > 1$. In fact k is approximately 2.1 according to [BA99] and [BKM+00]. In other words, there is an abundance of nodes with only a few links, and a small - but significant - minority that have a very large number of links [BA99]. This means that the traditional random graph model of Erdos and Renyi [Bol85] where nodes with very high connectivity are very rare is not appropriate for characterizing the Web. Motivated by this observation, Barabasi and Albert developed a stochastic model that captures the experimental data better than the random graph model [BA99] by incorporating two observations: (i) the Web is constantly growing, and (ii) the rate of acquisition of new links is proportional to the links that a page already has. Adamic and Huberman [AH99] later on pointed out that the second observation of [BA99] is correct provided that the connectivity growth rate is allowed to vary for each Web page to capture differences in popularity. They also suggested that the stochastic model of [HA99] (which demonstrates that the number of pages a Web site has follows a power-law) could also be used for the connectivity problem.

Identical power-law distributions for the degree of nodes in the Web graph have been observed independently by Kumar et. al. who studied a 40 million page data set [KRRT99], and more importantly by Broder et. al. [BKM+00] who used a much larger crawl with approximately 200 million pages and 1.5 billion links (this crawl comes from the search engine AltaVista). Similar power-law distributions have also been observed in other man-made networks e.g., the Internet [FFF99] or the citation graph of the Research Index Computer Science literature database [AJM01].

The analysis of Broder et. al. [BKM+00] is very significant because it has also revealed a very interesting picture of the Web's macroscopic structure. Over 90% of the approximately 200 million nodes considered form a single connected component if links are treated as undirected edges. This connected graph consists of four subgraphs reminiscent of a bow-tie (see Figure 2.3). The first subgraph is a strongly connected component comprising about 56 million pages (it is labelled with SCC in Figure 2.3). This component represents probably the most valuable part of the web and includes most portals, universities and corporations. The second and third pieces are labelled IN and OUT in Figure 2.3. IN is composed by pages that can reach the SCC, but cannot be reached from it (e.g, these can

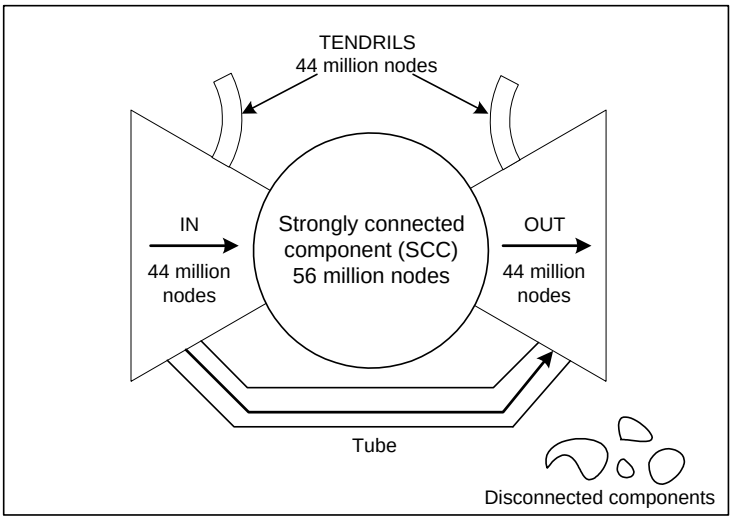

Figure 2.3: The Web's bow-tie structure from [KRRT02]

be new sites that people have not yet discovered and created links to). OUT is composed by pages that are accessible from the SCC but do not link back to it (e.g., these can be corporate websites that do not point to outside pages). Finally, the part labelled TENDRILS in Figure 2.3 contain pages that cannot reach the SCC, and cannot be reached from the SCC. The parts IN, OUT and TENDRILS contain about 44 million pages i.e., all four subgraphs have roughly the same size.

Broder et. al. have also showed that the *diameter* of SCC is at least 28, and that the diameter of the graph as a whole is over 500 [BKM+00]. They have also shown that for randomly chosen source and destination pages, the probability that any path exists from the source to the destination is only 24%. Under some simplifications a *small-world phenomenon* [Mil67, WS98, Wat99] also manifests itself: if a directed path exists, its average length is about 16. Similarly, if an undirected path exists, its average length is about 6. This particular experimental result is very close to previous predictions by Barabasi and Albert [BA99].

Researchers have also been interested in discovering *web communities* i.e., collections of web pages that deal with a common topic. Under various graph-theoretic definitions of what a community is and with the utilization of appropriate graph-theoretic techniques on small parts of the web, papers [KRRT99, FLGC02] have shown that despite its decentralized, unorganized and heterogeneous nature, the web *self-organizes* so that one can discover *communities* based *purely on link structure and connectivity* of the web graph.

Recent work has also addressed the question of *self-similarity* of the Web i.e., does the Web consist of subgraphs that are "mini Webs"? In other words, is the Web a fractal? With this question in mind Dill et. al. have studied a large Web crawl and have concluded that self-similarity in the web is pervasive and robust

[DKM+02]. It is pervasive in the sense that "meaningful" subsets of the Web (e.g., a corporate intranet or pages about a topic) are themselves "mini Webs" and their graph-theoretic properties are the same with the entire web. This result holds regardless of the particular way these "meaningful" subsets of the web are defined. Additionally, self-similarity is robust in the sense that the parameters corresponding to the graph-theoretic properties do not change significantly for these subsets (e.g., for many "mini Webs" the power-law exponent is approximately 2.1). Finally, if the web is decomposed into these "meaningful" subsets (under a variety of definitions for "meaningful"), there is a *navigational backbone* that makes these subsets strongly connected. Users can then navigate from one "mini Web" to another following this navigational backbone. The web therefore, according to [DKM+02], is the product of essentially independent stochastic processes that evolve at various scales all roughly following the model of [KRR+00].

The results of the above studies can be very useful in the following situations as summarized nicely in [BKM+00, FLGC02, DKM+02]:

- Analyzing and improving search algorithms that make use of link information e.g., Google's PageRank algorithm [BP98] or Kleinberg's HITS algorithm [Kle00].
- Designing better web crawlers, and better browsing and information foraging strategies.
- Designing vertical (theme-based) portals, information dissemination and e-commerce services automatically by relying on discovered web communities. These developments will take us well beyond current technology such as static taxonomies of portals like Yahoo.
- Understanding the social phenomena underlying content creation on the web and predicting the emergence of important new phenomena in the web. For example, a great number of links between web pages of various scientists can indicate emerging connections between their respective disciplines.

Previous results, similar concepts and well-understood formal tools of various disciplines (mathematics, computer science, physics, social science and biology) have been very useful in the above studies. For example, researchers have utilized insights from small-world networks as they appear in social or biological contexts [Mil67, WS98] in the study of small-world phenomena in the web, or borrowed ideas from applied mathematics to predict accurately the structure of networks by looking only at a finite sample [BA99]. For classification purposes one might also want to distinguish efforts in this area between more theoretical ones that use stochastic models to explain the graph structure of the web [BA99, KRR+00] and more experimental ones that are used to validate and refine proposed models [BKM+00, KRRT99, FLGC02, DKM+02].

2.2.2 Measurement Studies for P2P Networks

More recently similar measurement efforts in the area of P2P systems have been carried out concentrating mostly on Napster [SGG02] and Gnutella [Mar02, SGG02, AH00, ABJ01, Jov01, RFI02].

Adar and Huberman [AH00] were the first to study the number of files shared and downloaded by Gnutella peers by sampling messages for a 24-hour period in the life of a Gnutella network. [AH00] discovered that 66% of all Gnutella peers shared *no* files (i.e., they were *free riders*) and that only 1% of the peers served 37% of the files shared. Additionally, 67% of the peers did not answer any queries presumably because they had no files somebody else was interested in. Free riders were found to be distributed evenly among Internet domains. Furthermore, [AH00] found that there is no correlation between the quantity of files a peer has and their quality (i.e., whether they can be used to answer user queries). Gnutella queries seem to concentrate on popular topics and at the same time there are peers, possibly with few files, that specialize on these topics. As a result, these peers are able to respond to many queries.

The study of [AH00] is very interesting and it poses some interesting questions. How detrimental is the (negative) effect of free-riding to the performance of a Gnutella network? How can we eliminate free-riding to end up with a co-operative community of file sharing peers? Is there a danger of legal consequences for Gnutella users sharing copyrighted material? The results of [AH00] show that the Gnutella network does not really offer the promised anonymity and privacy benefits to participating peers because it is easy for an interested party to detect the peers that champion piracy by serving most of the shared files.

Jovanovic has studied the Gnutella topology by implementing a distributed network crawler that enabled measurements between the months of May and December of 2000 [Jov01]. His analysis reveals the presence of small-world properties and power-law distributions the Gnutella network. Specifically, [Jov01] points out that the Gnutella network exhibits the typical small diameter and clustering properties characteristic of small-world networks and compares the obtained values for those parameters with respective values in other small-word models (e.g., the model of Watts and Strogatz [WS98] and the model of Barabasi and Albert [BA99]) and the random graph model of Erdos and Renyi [Bol85]. Additionally, [Jov01] gives out that Jovanovic also demonstrated that the Gnutella network obeys the four power laws originally discovered in the context of the Internet by [FFF99].

More recently, [SGG02] carried out the most detailed study of Napster and Gnutella networks currently available. They traced four days of activity in the Napster network and eight days of activity in a Gnutella network capturing various characteristics of 500 thousand peers for the former system and 1.2 million peers for the latter. [SGG02] reports that the set of peers participating in Napster and Gnutella is *very heterogeneous* with respect to many characteristics: Internet

connection speeds, latencies, availability and quantity of shared files. In many cases, the difference in these characteristics is three to five orders of magnitude across the peers in the whole system. For example, only about 25% of the users in Napster and 30% of the users in Gnutella have Internet connections with bandwidth 3Mbps or more. The most popular form of access in these systems are cable modems and DSLs. Thus, although Gnutella fans stress the fact that every peer in Gnutella is capable of being both a client and a server, reality shows that some peers are simply *incapable* of playing their server role effectively. Thus, future P2P systems should take into account capabilities of peers when delegating authority or responsibility. In this way, [SGG02] have in some sense prophesied the coming of deployed super-peer networks such as Morpheus and Kazaa, and related academic proposals such as [TXKN03, YGM03].

Additionally [SGG02] confirms the rampant free-riding originally pointed out by [AH00] and they also point out that many peers (with moderate bandwidth connections) purposefully *lie about their bandwidth* to avoid becoming the target of a high number of downloads. Thus, future P2P systems should either offer incentives to its participants to encourage co-operation, or have a way to directly measure and verify any peer data needed.

Moreover, [SGG02] points out that because Gnutella networks follow a power-law with exponent 2.3 (a result that they attribute to a currently unavailable Web site `http://www.clip2.com`), they are very robust in the phase of random removal of nodes (or peer breakdowns) as can be shown by utilizing the models of Cohen et. al. [CEbAH00]. However, a Gnutella network can become fragmented in the phase of a carefully orchestrated attack that removes a small percentage of "strategically placed" (highly connected) nodes.

Markatos has also studied traffic characteristics on a Gnutella network by installing two peers in Europe (Crete and Norway) and one in the US (Rochester) and monitoring queries and responses for one hour [Mar02]. His study shows that traffic in Gnutella due to queries and query responses is very bursty even when measured at various time scales (similar results have been reported for Internet and Web traffic in the past). He also observed that queries in Gnutella exhibit locality properties with 40% of the queries observed having been submitted more than once. He proposes a caching technique that takes locality into account and results in a reduction in network traffic.

Ripeanu et. al. have also performed a detailed measurement study of the topology and dynamics of the Gnutella network observed over the period November 2000 to June 2001 [RFI02]. Their measurements shows the Gnutella network to grow by about 25 times during the seven months of observation starting from around 2,000 nodes. They also confirm the belief that network membership is very dynamic with a great number of nodes staying in the Gnutella network for very little time (in this study 40% of the nodes stay for less than 4 hours).

[RFI02] have also found found that 95% of all pairs of Gnutella nodes are connected by shortest paths of length less than 7 hops. Given that the Gnutella

protocol limits messages to a TTL=7, one might hasten to conclude that all broadcasted messages will reach all nodes. However, the very opposite fact may be true in Gnutella networks operating under TTL bounds and unique message identification mechanisms (see Section 2.1.1) as pointed out in [Jov01] and [ABJ01]. [ABJ01] studies P2P networks where broadcasts are implemented by flooding governed by TTL bounds and unique message identification mechanisms (the typical case is Gnutella). [ABJ01] shows that in the presence of *heterogeneous latencies* it is possible that a message broadcasted from some node a will never reach some node b even though the length of the shortest-path connecting a with b is less than or equal to the TTL bound (they call this effect *short-circuiting*). [ABJ01] demonstrated experimentally by using the data collected by Jovanovic in [Jov01] that short-circuiting can actually decrease the nodes reachable from another node in a Gnutella network by as much as 50%.

[RFI02] have also observed that most network traffic in Gnutella is due to query broadcasting and have estimated that the generated traffic corresponds to a significant percentage of the total Internet traffic in the U.S. (around 1.7%). Obviously, this presents an important obstacle to the scalability of Gnutella networks.

Moreover, [RFI02] have discovered that the Gnutella networks started as a pure power-law network in November 2000, but moved away from this distribution in March and May 2001. In their collected data for the latter period, the pure power-law distribution is only preserved for nodes with more that 100 links, while nodes with fewer links follow an almost constant distribution. [RFI02] concludes that this multi-modal distribution might have a good impact on network reliability by limiting the need to depend on highly-connected nodes. However, [RFI02] does not present any detailed experiments to support this conclusion.

In another part of their study, [RFI02] have sought to investigate how well the Gnutella overlay network matches the underlying Internet infrastructure. For the purposes of their study, the Internet is considered to be a collection of interconnected *autonomous systems* i.e., groups of local area networks under shared technical administration. [RFI02] point out that only 2% to 5% of Gnutella connections link nodes within an autonomous system, while more than 40% of the traffic among Gnutella nodes crosses autonomous system borders. Because traffic inside an autonomous system can be handled more easily, the mismatch of the two networks clearly indicates inefficient use of Internet resources in Gnutella networks. Finally, [RFI02] have also performed a detailed clustering experiment and concluded that Gnutella nodes cluster independently of the Internet structure imposed by DNS.

The above studies are very useful and clearly point out many interesting avenues for future research. The first obvious question is to identify and classify the various forms of self-organization present in P2P networks (e.g., do we observe structures very similar to the ones observed in the Web?). To answer this question, more work needs to be done in the area of measurement and modelling

for P2P networks. Regarding the studies already available, we should also point out that Napster is currently unavailable and Gnutella is not the most popular P2P network any more (this position is now taken by the super-peer network Morpheus according to [YGM03]). In addition, a multitude of other P2P systems have been proposed where the original architectural decisions made by the designers of Napster and Gnutella have not been followed. Typical examples of these systems include the ones based on distributed hash-tables such as Chord [SMK+01] or CAN [RFH+01], super-peer networks [YGM03] and other structured P2P designs [TXK02, TXKN03]. It would be interesting to study in detail the different design choices of these more recent architectural alternatives, and carry out large-scale measurement efforts to uncover the hidden structures that have evolved as a result of these choices. The results of these measurement and modelling studies will tell us a lot about the P2P networks of interest:

- They can give valuable insights into the structure and topology of various types of P2P systems and their local and global properties. This can lead to better classifications of P2P system architectures than the ones currently available [MKL+01, DGMY03], and, as a result, improve our understanding of the functionality of each architecture. It can also influence the development of new architectural alternatives that mix features from two or more types of existing P2P systems in order to get the benefits of all.

- They can be used to evaluate current search and routing algorithms and influence the design of new ones.

- They can provide explanations for social phenomena that characterize the evolution of P2P networks or allows us to predict the emergence of new phenomena.

We also urgently need good mathematical models for performance analysis of P2P networks, and for modelling their evolution and dynamics. These models could possibly be more complicated than the ones for the Web or traditional distributed systems. For some recent work on performance modelling for P2P networks the reader should consult the work by Yang and Garcia-Molina [YGM03]. From our point of view, it would also be interesting to see what ideas and mathematical models from various subareas of complex adaptive systems research (e.g., biology, social science and economics) could be profitably used in this area.

2.3 Economic Ideas in P2P Systems

Although P2P systems are a prime candidate for the utilization of tools from microeconomics and game theory, very few researchers have so far taken this route.

Cooper and Garcia-Molina [CGM02a] discuss the case of a P2P network of cooperating archives that *trade space* in order to replicate and preserve their digital collections. For example, an archival site A storing research papers might contact an archival site B storing software and propose to store its collection of research papers at B's servers. In return, B will be allowed to store its software collection at A's servers. This allows both sites to replicate their collections and increases global reliability. Moreover, they define two measures of reliability: *local reliability* (the probability that no collection owned by a certain site will be lost) and global reliability (the probability that no collection owned by any site will be lost). The goal of the distributed trading algorithms in [CGM02a] is to maximize global reliability. Thus they introduce introduce the notion of *deed* for modelling trades of storage space (in analogy to property deeds). Deeds keep track of space owed by one site to another (equivalently, the latter's site right to use a block of space owned by the former). Finally [CGM02a] discusses trading whole collections or trading equal blocks of space. Going a step further, [CGM02b] discusses the issue of having to choose between multiple offers of space by different peers. For example, site A might need space for its 100GB collection but site B agrees to store A's collection only if A stores in return the 150GB collection of A. In addition, there is a site C which can also store A's collection but it needs 120GB of available storage in return. What is an appropriate trade for site A? To answer this question, [CGM02b] discusses how to use *auctions* and *bidding* to facilitate these trading scenarios.

In other recent work, the problem of trading storage space in P2P networks is also considered. In [NWD03], each peer brings with it some storage space that is made available to the network for storing files published by other peers (storage is the only resource considered). Peers are allowed to consume as much of the network's storage space as they provide in their local node. Moreover, [NWD03] proposes that peers maintain *records of their resource usage* (e.g., how many GBs of storage they make available to the community, a list of files stored locally for other nodes and a list of own files stored at other nodes) and make these records available for auditing by other peers. Because nodes are expected to behave selfishly and might not tell the truth about their resource usage, [NWD03] concentrates on protocols designed to enforce cooperative behavior by not allowing nodes to *collide* with other nodes to take advantage of the system resources in an unfair way.

Ideas from economics could also be utilized to provide incentives for users and discourage free-riders in P2P systems as originally suggested in [AH00]. The first P2P system to provide a solution to this problem is Mojo Nation[14]. Mojo Nation provides a digital currency, the "mojo", used to count resources (disk space, bandwidth and processing cycles) contributed to or requested by the system. Each interaction in Mojo Nation involves some exchange of mojos, and peers

[14] `http://www.mojonation.net/`

perform bookkeeping using a background payment system. There is also a trusted third party which ensures honest transfers between peers. Mojo Nation could also monitor content usage and facilitate new business models for P2P networks that discourage piracy and enable royalties to be paid to the owner of content.

[HPS01] have also proposed to use micro-payments and a trusted third party (called an *escrow server*) to encourage P2P network users to join legitimate P2P networks and play by the rules (e.g., do not create spam, do not serve illegal content and avoid free-riding). Their work concentrates mostly on the cryptography issues involved for realizing their proposal.

[GLBM01] is a theoretical paper which also addresses the problem of incentives in P2P networks using a game theoretic analysis. According to this study, P2P network users can be distinguished in two categories: *altruistic users* for whom being altruistic has higher utility than the sum of the costs associated with sharing files (e.g., bandwidth and storage costs), and *non-altruistic* users for whom the opposite holds. Under some reasonable assumptions, [GLBM01] shows that there is a dominant strategy for altruistic users: share and download as many files as possible. There is also a dominant strategy for non-altruistic users: download as many files as possible and share none. These results give a simple mathematical explanation for the behavior observed by [AH00].

Finally [GLBM01] studies also the question of incentives in centralized systems (e.g., Napster) so that free-riding behaviour is discouraged. They propose that central servers keep track of downloads and uploads and charge users a fee for downloading files while paying them a fee for uploading. Three kinds of payment schemes are discussed and compared: micropayments, quantized micro-payments (i.e., micropayments for blocks of files instead of single files), and payments in terms of some internal currency that can be bought with real money but cannot be exchanged with real money. Interestingly enough one can now find equilibria in which it is a dominant strategy for every agent to share files and not be a free-rider.

2.4 Conclusions

In this chapter we discussed three prominent file sharing P2P systems and also gave an overview on more recent research aiming the improvement of P2P system infrastructure with the use of distributed hash-tables. We also looked into measurement studies for the Web and put focus on similar studies that reveal interesting characteristics of existing P2P systems. Finally we surveyed economics inspired research that addresses various open issues in P2P networks.

Chapter 3

Measures of Fairness in Resource Allocation

In this section we study resource allocation in an abstract setting. We deal with allocations of resources to agents and explore various notions of fair resource allocation. The concepts that we develop are very general and can possibly be useful in various contexts where resource allocation is an important topic: (e.g., human economies or computer systems and networks). In later chapters of this report we apply the concepts studied in this chapter to the problem of resource allocation in P2P systems.

Allocations of resources will be represented as vectors of real numbers. In this respect the following notation of [MO79] comes very handy so we introduce it immediately.

Definition 3.1 *Let $\overline{x} = (x_1, \ldots, x_k)$ be a vector of reals. We will denote by $x_{[i]}$ the i-th element of $\overline{x}$, when sorted in descending order. Similarly we will denote by $x_{(i)}$ the i-th element of $\overline{x}$ when sorted in ascending order.*

In what follows we study three measures of fairness: fairness index[JCH84], max-min fairness [KRT01] and majorization [MO79]. Initially the fairness measures are defined formally and then their properties are discussed thoroughly. Fairness index is related to other statistical measures, and max-min fairness is correlated to majorization. Finally, we explain the reason why we choose to use the measure of fairness index for allocating the resources in a P2P system.

3.1 Fairness Index

The first quantitative measure we study is the fairness index as proposed in [JCH84].

Definition 3.2 *Let A be a non-negative real number representing an amount of some given resource R in terms of units of some measure. Let $\overline{x} = (x_1, \ldots, x_k)$ be a vector of reals representing an allocation of amount A to k agents such that agent i is allocated x_i units of R and $A = \sum_{i=1}^{k} x_i$. Then, the* fairness index *of this allocation is given by a function $\mathcal{F} : \mathbb{R}_+^k \to (0, 1]$ that is defined as follows:*

$$\mathcal{F}(\overline{x}) = \frac{(\sum_{i=1}^{k} x_i)^2}{k \sum_{i=1}^{k} x_i^2}$$

The fairness index measures "how equal" the amounts x_i are in the allocation $\overline{x}$. If all agents get the same amount of resources (i.e., all x_i's are equal) then the fairness index is 1, implying that the allocation is 100% fair. As the disparity increases and the vector $\overline{x}$ becomes more "skewed", the fairness index value decreases giving a notion of how far this allocation is from the perfect. It is easy to see that an allocation is perfect if and only if A is divided evenly by k. For a non-perfect allocation, there are cases where the fairness index would give us the fraction of favored agents. The following example from [JCH84] illustrate this intuitive interpretation.

Example 3.1 *Suppose one is asked to distribute 20 Euros among 100 persons. Let us consider the following two ways to distribute the money:*

- *First Allocation: Give 20 cents to each of the 100 persons. If this allocation is represented by $\overline{x}$, then*

  $$x_i = 0.2 \ for \ i = 1, 2, \ldots, 100.$$

  The fairness index of this allocation is obviously 1, thus $\overline{x}$ is perfect.

- *Second Allocation: Depending upon some criterion, choose 10 persons and give them 2 Euros each. The other 90 persons get no money. If we represent this allocation by $\overline{y}$, then:*

  $$y_i = \begin{cases} 2, & i=1,2,3,\ldots,10 \\ 0, & i=11,12,\ldots,100 \end{cases}$$

  The fairness index of this allocation is:

  $$\mathcal{F}(\overline{y}) = \frac{[\sum x_i]^2}{n \sum x_i^2} = 0.10$$

  This allocation is only 10% fair because only 10% of the agents are treated equally.

The fairness index has some nice intuitive properties as pointed out in [JCH84]. Some of these properties are summarized below:

1. The fairness index is *population size independent* i.e., it can be applied to any number of agents.

2. The fairness index is *independent of scale and metric* i.e., the unit of resource measurement does not matter. More formally, the fairness index of allocation $\overline{x}$ and $c\overline{x}$ (where $c > 0$) is the same.

3. The fairness index is *bounded* i.e., its values are in the interval $(0, 1]$. Thus, fairness can be expressed as a percentage, and this helps in the intuitive understanding of the differences among allocations.

4. The fairness index is *continuous* so that any slight change in x_i changes the index.

5. If we take an amount of resource $\delta x > 0$ from one agent i with resource x_i and give it to another agent j with resource x_j, then the fairness index of the resulting allocation:

   - increases (the new allocation is better) if $\delta x < |x_i - x_j|$.
   - decreases (the new allocation is worse) if $\delta x > |x_i - x_j|$.
   - remains the same if $\delta x = |x_i - x_j|$.

   This property is very intuitive. It is known as the *principle of transfers* and it is also discussed in [MO79] and attributed to Dalton [Dal20].

6. Let $\overline{x} = (x_1, \ldots, x_n)$ be an allocation of some resource to n agents. If each agent is given the same amount of additional resource c, then $\mathcal{F}(\overline{x} + c) \geq \mathcal{F}(\overline{x})$ where $\overline{x} + c$ denotes the allocation $(x_1 + c, \ldots, x_n + c)$.

7. If only one agent is given additional resources, then the fairness index is decreased if this agent is a favored one. The fairness index is increased otherwise.

8. The fairness index has a bell-shaped behavior curve with respect to the allocation to each individual agent. Thus, the fairness index firstly increases when an individual's allocation increases. This behavior happens up to a critical point. From this point on, any additional increase to the individual allocation results in a decrease to the fairness index.

Example 3.2 *Suppose we want to distribute a single unit of a resource among two agents. If we assume that the first agent gets a fraction x_1 of the unit, then the other agent gets $1 - x_1$. Then the allocation vector is $\overline{x} = (x_1, 1 - x_1)$ and the fairness index is*

$$\mathcal{F}(\overline{x}) = \frac{(x_1 + 1 - x_1)^2}{2(x_1^2 + (1 - x_1)^2)} = \frac{1}{2(x_1^2 + (1 - x_1)^2)}.$$

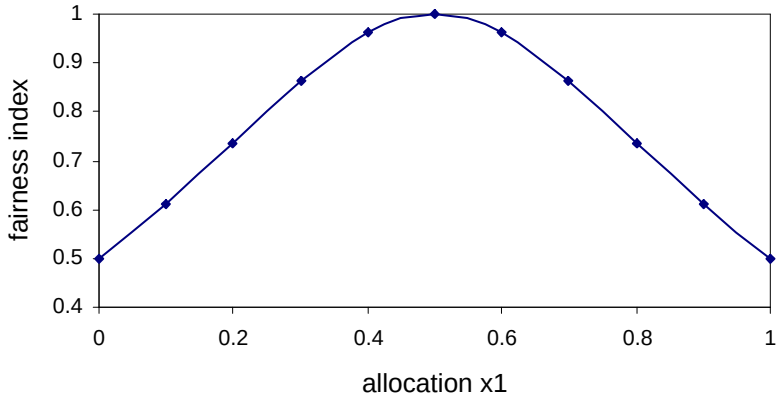

Figure 3.1: A bell-shaped fairness index curve

Figure 3.1 presents the fairness index as a function of x_1. We observe the bell-shaped behavior described above. The minimum value of $\mathcal{F}$ is 0.5 when one of the agents gets the whole of the unit and the maximum value is reached when we have a perfect allocation (both agents get 1/2 and $\mathcal{F}(\overline{x}) = 1$).

9. If there is no limit on allocations, then the worst case of fairness can be near zero. By putting upper and lower bounds on allocations, the fairness index can guarantee a minimum level of fairness.

3.2 The Relation of Fairness Index to Other Statistical Measures

Let us now consider $\overline{x}$ to give us the values of some random variable X, where X takes each value x_i from $\overline{x}$ with equal probability. In what follows we will use $\overline{x}$ to refer to X so that we do not complicate our notation.

Let μ, Var and Cov denote the mean, variance and coefficient of variation of the random variable X. Then

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i,\ Var = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2 \text{ and } Cov = \frac{Variance}{Mean}.$$

We can also use the first two moments of $\overline{x}$ to get a new expression for the fairness index. The first and second moment of $\overline{x}$ are

$$m_1 = \frac{1}{n}\sum_{i=1}^{n} x_i \text{ and } m_2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2.$$

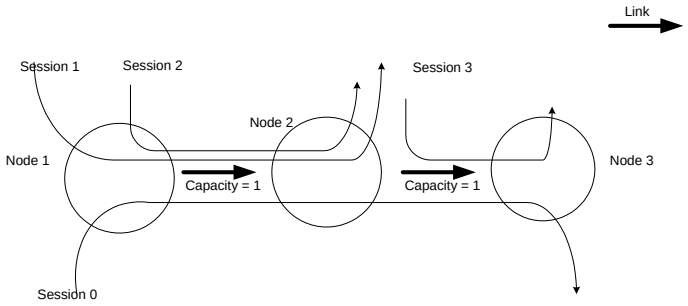

Figure 3.2: A data network

Thus we can express $\mathcal{F}(\overline{x})$ as follows:

$$\mathcal{F}(\overline{x}) = \frac{[\sum x_i]^2}{n \sum x_i^2} = \frac{[\frac{1}{n} \sum x_i]^2}{\frac{1}{n} \sum x_i^2} = \frac{m_1^2}{m_2} \tag{3.1}$$

The coefficient of variation can also be expressed in terms of moments as follows:

$$Cov = \frac{(m_2 - m_1^2)^{\frac{1}{2}}}{m_1}$$

Now using the expression of fairness index in terms of moments (Equation 3.1), we can relate the fairness index with covariance as follows:

$$\mathcal{F}(\overline{x}) = \frac{m_1^2}{m_2} = \frac{m_1^2}{m_1^2 + m_2 - m_1^2} = \frac{1}{1 + \frac{m_2 - m_1^2}{m_1^2}} = \frac{1}{1 + Cov^2} \tag{3.2}$$

This relationship is important because it shows why the fairness index is a better measure of fairness than the coefficient of variation. The advantage of the fairness index is that it is bounded in the interval $(0, 1]$ while the coefficient of variation is not. This makes fairness index a more intuitive measure of fairness. Notice also the inverse relation between the fairness index and the coefficient of variation; when we have a perfect allocation the coefficient of variation is zero. If we increase unfairness then the coefficient of variation increases and the fairness index decreases.

3.3 Max-Min Fairness

In the networking community the notion of *max-min fairness* has been very popular. Max-min fairness forms the basis for bandwidth allocation in both data networks and other Internet applications.

Figure 3.2 taken from [BG87] shows a data network with 3 nodes and 2 links and can be used to explain the concept of max-min fairness. In this network session 0 goes through both links while the others go through only one link each. We try to maximize the network bandwidth allocated to each session. Thus, giving sessions 0, 1 and 2 a rate of 1/3 each is the fairest solution given the capacity of the link from node 1 to node 2. If we also give session 3 a rate of 1/3, this results in a waste of extra capacity available at the rightmost link. The solution that takes advantage of the capacity at the rightmost link without limiting session 0 is to give session 3 a rate of 2/3.

This example motivates us to consider the following approach to fairness. Initially, give the most poorly treated session as much as possible bandwidth. Then, ignoring this session, make sure that the next poorest session is given as much as possible bandwidth, etc. Another way to express the same idea is the following: taking into account the constraints of the network, maximize the bandwidth b_i given to the session i without decreasing the bandwidth b_j allocated to any other session j such that $b_j \leq b_i$. We also have a very natural definition of *fairest* in this case: there is no way to increase the allocation b_i to a session i without decreasing the allocation b_j to a session j, such that $b_j \leq b_i$.

Let us now define formally the concept of max-min fairness. We do this through the definition of an appropriate order relation[1].

Definition 3.3 *Let $\overline{x}$ and $\overline{y}$ be two allocation vectors. Then we will say that $\overline{x}$* lexicographically dominates *$\overline{y}$, denoted by $\overline{x} \preceq_l \overline{y}$, if $\overline{x} = \overline{y}$, or there is some index j for which $\overline{x}_{(j)} > \overline{y}_{(j)}$ and $\overline{x}_{(i)} = \overline{y}_{(i)}$ for all $i < j$.*

We will say that $\overline{x}$ and $\overline{y}$ are *equivalent* if both $\overline{x} \preceq \overline{y}$ and $\overline{y} \preceq \overline{x}$.

The $\preceq_l$ defines a total order on the equivalence classes of allocation vectors [KRT01]. This implies that the fairest allocation according to $\preceq_l$ is the one whose vector is the unique maximal under $\preceq$.

3.4 Majorization

In 1905 Lorenz proposed a *fair allocation policy* that simultaneously gives a high value of resources to every agent, trying to achieve fairness on average as well as to each individual agent [Lor05, MO79]. Lorenz proposed to plot the points that represent the amount of resources allocated to the poorest agent, then the sum of the 2 poorest agents and so on, and consider the curve obtained by connecting this points. If we have a uniform distribution of resources (perfect allocation) then this curve is a straight line, otherwise the curve is convex and lies under the straight line. We can also notice that an allocation whose curve lies over

[1]The networking literature also gives formal machinery to represent network graphs, capacities, etc. However, we will not need these in our work.

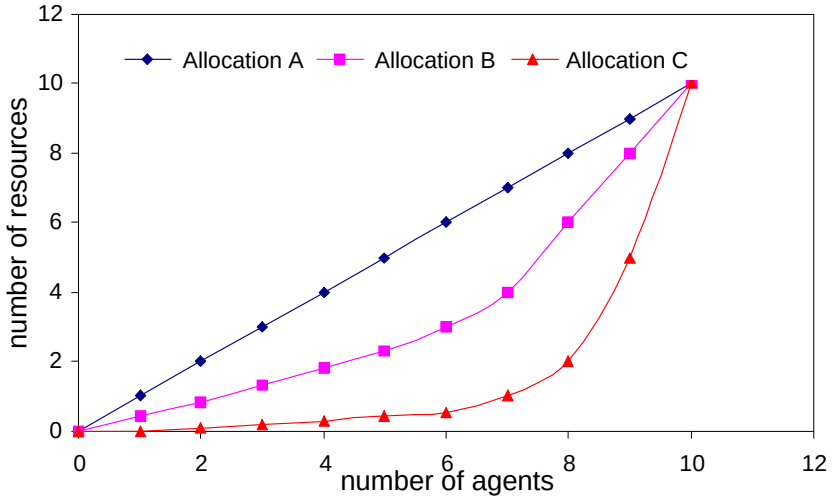

Figure 3.3: Lorenz curves

the curve of another allocation is fairer, i.e., it has a more equal distribution of resources.

Example 3.3 *Suppose we want to distribute 10 Euros among 10 persons. Let us consider the following three allocation vectors:*

- $(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$
- $(0.4, 0.4, 0.5, 0.5, 0.5, 0.7, 2, 2, 2)$
- $(0, 0.1, 0.1, 0.1, 0.1, 0.1, 0.5, 1, 3, 5)$

that represent allocations A, B and C.

Figure 3.3 represents these allocations according to Lorenz's proposal. A is a perfect allocation and B is fairer than C according to Lorenz. This is exactly the result we were intuitively expecting, since in allocation B resources are more equally distributed than in C.

Later on, Lorenz's intuitive found its formalization in the the concept of *majorization* proposed by Hardy, Littlewood and Polya in the 1920s to study properties of inequalities [HLP52]. The classic textbook on majorization is now [MO79], and we follow it closely in this section.

Definition 3.4 *Let* $\overline{x} = (x_1, x_2, \ldots, x_k)$ *and* $\overline{y} = (y_1, y_2, \ldots, y_k)$ *be two allocation vectors. Then we will say that* $\overline{x}$ *is* majorized *by* $\overline{y}$ *(or* $\overline{y}$ majorizes $\overline{x}$ *or* $\overline{x}$ is less skewed *than* $\overline{y}$*) and denoted by* $\overline{x} \prec \overline{y}$*, if:*

1. $\sum_{i=1}^{n} \overline{x}_{[i]} \leq \sum_{i=1}^{n} \overline{y}_{[i]}$, for all $n = 1, \ldots, k-1$.

2. $\sum_{i=1}^{k} \overline{x}_{[i]} = \sum_{i=1}^{k} \overline{y}_{[i]}$.

In the study of majorization the following class of functions studied by Schur is really important [MO79].

Definition 3.5 *A real-valued function ϕ defined on a set $\mathcal{A} \subset \mathbb{R}^n$ is said to be* Schur-increasing *on $\mathcal{A}$ if*

$$\overline{x} \prec \overline{y} \text{ on } \mathcal{A} \Rightarrow \phi(\overline{x}) \leq \phi(\overline{y}). \tag{3.3}$$

If, in addition, $\phi(\overline{x}) < \phi(\overline{y})$ whenever $\overline{x} \prec \overline{y}$ but $\overline{x}$ is not a permutation of $\overline{y}$, then ϕ is said to be strictly Schur-increasing *on $\mathcal{A}$. If $\mathcal{A} = \mathbb{R}^n$, then ϕ is simply said to be* Schur-increasing *or* strictly Schur-increasing[2].
Similarly, ϕ is said to be Schur-decreasing *on $\mathcal{A}$ if*

$$\overline{x} \prec \overline{y} \text{ on } \mathcal{A} \Rightarrow \phi(\overline{x}) \geq \phi(\overline{y}), \tag{3.4}$$

and ϕ is strictly Schur-decreasing *on $\mathcal{A}$ if the strict inequality $\phi(\overline{x}) > \phi(\overline{y})$ holds when $\overline{x}$ is not a permutation of $\overline{y}$.*

The following theorem is from [MO79] and it is due to Schur and Ostrowski.

Theorem 3.1 *Let $\mathcal{I} \subset \mathbb{R}$ be an open interval and let $\phi : \mathcal{I}^n \to \mathbb{R}$ be continuously differentiable. Necessary and sufficient conditions for ϕ to be Schur-increasing on $\mathcal{I}^n$ are*

$$\phi \text{ is symmetric on } \mathcal{I}^n \tag{3.5}$$

and

$$\frac{\partial \phi}{\partial z_i}(\overline{z}) \text{ is decreasing in } i = 1, \ldots, n \quad \text{for all } z \in \mathcal{D} \cap \mathcal{I}^n. \tag{3.6}$$

Alternatively, ϕ is Schur-increasing on $\mathcal{I}^n$ if and only if (3.5) and for all $i \neq j$

$$(z_i - z_j)[\frac{\partial \phi}{\partial z_i}(\overline{z}) - \frac{\partial \phi}{\partial z_j}(\overline{z})] \geq 0 \text{ for all } \overline{z} \in \mathcal{I}^n. \tag{3.7}$$

The above theorem can be reformulated; for Schur-decreasing functions *decreasing* is replaced by *increasing* in (3.6) and inequality (3.7) is reversed.

An equivalent form of the above theorem, with the help of remark A.5 in Page 58 of [MO79], is given by the following corollary:

[2]In the literature the terms *Schur-convex or concave* are more often used but we will use *Schur-increasing or decreasing* instead because we believe that they are more intuitive. The textbook [MO79] clearly agrees with us although prefers to follow the literature.

Corollary 3.1 *Let $\mathcal{I} \subset \mathbb{R}$ be an open interval and let $\phi : \mathcal{I}^n \to \mathbb{R}$ be continuously differentiable. Then ϕ is Schur-increasing on $\mathcal{I}^n$ if and only if*

$$\phi \text{ is symmetric on } \mathcal{I}^n \tag{3.8}$$

and

$$(z_1 - z_2)[\frac{\partial \phi}{\partial z_1}(\overline{z}) - \frac{\partial \phi}{\partial z_2}(\overline{z})] \geq 0 \text{ for all } \overline{z} \in \mathcal{I}^2. \tag{3.9}$$

The following result has been first stated in [GMP01].

Proposition 3.1 *The fairness index metric $\mathcal{F}$ is a Schur-decreasing function.*

Proof: The fairness index of allocation $\overline{x}$ is given by

$$\mathcal{F}(\overline{x}) = \frac{(\sum_{i=1}^k x_i)^2}{k \sum_{i=1}^k x_i^2}.$$

1. $\mathcal{F}$ is *continuously differentiable* iff $\mathcal{F}$ is differentiable and that its first partial derivatives are continuous. This follows easily from the properties of polynomials and continuity.

2. $\mathcal{F}$ is *symmetric* because:

$$\mathcal{F}(-\overline{x}) = \frac{(\sum_{i=1}^k -x_i)^2}{k \sum_{i=1}^k (-x_i)^2} = \frac{(-\sum_{i=1}^k x_i)^2}{k \sum_{i=1}^k (-x_i)^2} = \frac{(\sum_{i=1}^k x_i)^2}{k \sum_{i=1}^k x_i^2} = \mathcal{F}(\overline{x}).$$

3. $(x_1 - x_2)[\frac{\partial \mathcal{F}}{\partial x_1}(\overline{x}) - \frac{\partial \mathcal{F}}{\partial x_2}(\overline{x})] \leq 0$ for all $\overline{x} \in \mathcal{I}^2$.
   It is easy to see that:

$$\frac{\partial \mathcal{F}}{\partial x_1}(\overline{x}) = \frac{2(2x_1 + 2x_2)(x_1^2 + x_2^2) - 4x_1(x_1^2 + x_2^2 + 2x_1x_2)}{4(x_1^2 + x_2^2)} =$$

$$\frac{4x_1^3 + 4x_1x_2^2 + 4x_1^2x_2 + 4x_2^3 - 4x_1^3 - 4x_1x_2^2 - 8x_1^2x_2}{4(x_1^2 + x_2^2)} = \frac{x_2^3 - x_1^2x_2}{x_1^2 + x_2^2}$$

   Similarly:

$$\frac{\partial \mathcal{F}}{\partial x_2}(\overline{x}) = \frac{x_1^3 - x_1x_2^2}{x_1^2 + x_2^2}$$

   Thus:

$$(x_1 - x_2)[\frac{\partial \mathcal{F}}{\partial x_1}(\overline{x}) - \frac{\partial \mathcal{F}}{\partial x_2}(\overline{x})] = (x_1 - x_2)[\frac{x_2^3 - x_1^2x_2}{x_1^2 + x_2^2} - \frac{x_1^3 - x_1x_2^2}{x_1^2 + x_2^2}] =$$

$$\frac{x_1x_2^3 - x_1^3x_2 - x_1^4 + x_1^2x_2^2 - x_2^4 + x_1^2x_2^2 + x_1^3x_2 - x_1x_2^3}{x_1^2 + x_2^2} =$$

$$\frac{2x_1^2x_2^2 - x_1^4 - x_2^4}{x_1^2 + x_2^2} = -\frac{(x_1^2 - x_2^2)^2}{x_1^2 + x_2^2} \leq 0$$

■

The above proposition and the definition of Schur-decreasing functions imply that majorization is *stricter* than the fairness index. Also, the most majorized allocation, when it exists, maximizes $\mathcal{F}$ (gives the highest possible fairness index value).

It is straightforward to see that majorization is stricter than max-min fairness simply by referring to Definitions 3.4 and 3.3. Thus, majorization is the stricter notion of fairness discussed in this section.

3.5 Conclusions

In this section we studied resource allocation in an abstract setting and presented in detail three basic and well-known measures of fairness: fairness index, max-min fairness and majorization. We demonstrated that majorization is the stricter of the three notions. For the rest of the work in this thesis, we adopt the fairness index as our measure of fairness. The question of how our results would change if we used majorization or max-min fairness is left to future work.

Chapter 4

Fair Allocation of Load in P2P Systems

In this chapter, we consider a P2P system and we concentrate on distributing the load fairly among the peers of the system. Initially, we describe the architecture of the system and then, we define formally the fair load allocation (FLA) problem. We use the measure of fairness index presented in Section 3.1 to balance the load among the peers.

Continuing, the FLA problem is reduced to the k-way number partitioning (NUMP) problem, and thus, we wonder whether algorithms that have been used to solve the NUMP problem can also be used to solve the FLA problem. We present the greedy heuristic, the set differencing heuristic, an algorithm based on dynamic-programming and complete algorithms for solving the NUMP problem, and then we propose a greedy (GFLA) and a complete (CGFLA) algorithm for the FLA problem. Moreover, we study the behavior of CGFLA which exhibits phase transitions. Finally, we present experimental results for both GFLA and CGFLA algorithms and we compare them in terms of their solution quality, that is the number of generated nodes and the fairness index achieved.

The rest of the chapter is organized as follows. In Section 4.1 the FLA problem is defined formally. In Section 4.2 the FLA problem is related to the k-way (NUMP) problem and algorithms solving the NUMP are presented. Section 4.3 presents a greedy algorithm for solving FLA, while in Section 4.4 a complete algorithm is presented. Section 4.5 describes the experiments carried out and discusses the results that emerged from these experiments.

4.1 The Fair Load Allocation Problem

Let us consider a P2P system that consists of k *nodes* or *peers*. We want to allocate n *documents* (e.g., text files, pictures, MPEGs etc.) to the peers so that each document is allocated to only one peer and the load of the system is

distributed fairly among the peers. Our problem is a classical *resource allocation* problem and we will call it the *fair load allocation (FLA)* problem.

We measure the load of a peer by the number of file downloads that it serves for the user community. To model this quantitatively, each document is associated with a *popularity* $p(d)$ that measures the probability that this document will be accessed or the number of hits this document will receive in some given time period that the P2P system is in operation. In the former case $p(d)$ can be a real number in the interval $[0, 1]$ while in the latter case $p(d)$ can be a natural number. The *load* of a peer i, denoted by $load(i)$, is then equal to the sum of the popularities of the documents it stores. In this study load will be measured by a natural number. This representation can be easily transformed into the probabilistic one by considering the total load of the system and expressing each natural number as a proportion of the total load.

In Chapter 3 we have already presented some measures of fairness, that have been used in problems similar to FLA. In this chapter fairness is measured quantitatively as a number in the interval $(0, 1]$ using the fairness index presented in Section 3.1.

Let us now define the FLA problem formally.

Definition 4.1 *[The Fair Load Allocation Problem or FLA] Let D be a set of n documents with popularities given by a function $p : D \to \mathbb{N}$. Let k be a set of peers. We would like to find a partition of D into subsets $D_1, \ldots, D_k$ so that the function*

$$\mathcal{F}(\sum_{d \in D_1} p(d), \ldots, \sum_{d \in D_k} p(d)) = \frac{(\sum_{i=1}^{k} \sum_{d \in D_i} p(d))^2}{k \sum_{i=1}^{k} (\sum_{d \in D_i} p(d))^2}$$

is maximized *where $\mathcal{F}$ is the fairness index function defined in Definition 3.2.*

We now give an example of the FLA problem. Notice that in this example and subsequent discussion we intentionally blur the difference between the set of documents D and its image under function P (a set of integers).

Example 4.1 *Let us consider an instance of FLA with $k = 4$ and $D = \{10, 9, 8, 7, 6, 4, 2\}$. The solution for this problem is the partition of D into the following subsets:*

$$D_1 = \{10\}, D_2 = \{9, 2\}, D_3 = \{8, 4\}, D_4 = \{7, 6\}$$

The resulting fairness index is equal to 0.99064.

4.2 The Number Partitioning Problem

The FLA problem as defined above is very similar to the *k-way number partitioning problem (NUMP)*, one of the six basic NP-complete problems proposed in [GJ79] and the only one that deals with numbers. NUMP is defined in [KK82] as follows.

Definition 4.2 *[The k-way Number Partitioning Problem or NUMP] Given a finite bag (multi-set) S of real numbers, partition S into k bags $A_1, \ldots, A_k \subseteq S$ so as to minimize the following difference:*

$$\Delta(A_1, \ldots, A_k) = \max_i \sum_{x \in A_i} x - \min_i \sum_{x \in A_i} x$$

Thus, both FLA and NUMP are combinatorial optimization problems differing only in their objective functions. The following example demonstrates that a solution to FLA is always a solution to NUMP but not vice versa. Notice also that FLA and NUMP are equivalent when $k = 2$ or $k = 3$.

Example 4.2 *We consider the instance of FLA considered in Example 4.1 as an instance of NUMP. Now there are two solutions that minimize $\Delta(A_1, \ldots, A_4)$:*

$$A_1 = \{10\}, A_2 = \{9, 2\}, A_3 = \{8, 4\}, A_4 = \{7, 6\}$$

and

$$A_1 = \{10\}, A_2 = \{8, 2\}, A_3 = \{9, 4\}, A_4 = \{7, 6\}$$

Notice that only the first solution is a solution to FLA because for the second one the fairness index is 0.98327*.*

NUMP has received enough attention in the literature and various heuristics have been proposed for it. We first describe a simple *greedy* algorithm presented by Korf in [Kor95b, Kor95a, Kor98], and then the set-differencing heuristic originally proposed by Karmarkar and Karp [KK82]. Both of these algorithms run in polynomial time but are not complete. Finally, we present two polynomial time algorithms: a pseudo-polynomial algorithm based on dynamic programming first described in [GJ79] and a complete algorithm that runs in exponential time [Kor95b, Kor95a, Kor98].

4.2.1 A Greedy Algorithm for NUMP

The greedy algorithm for NUMP, let us call it GNUMP, first sorts the numbers in S in decreasing order, and then places each one of them successively into the bag with the smallest partial sum. This algorithm is illustrated by the following example[1].

Example 4.3 *Suppose we want to partition the bag $S = \{8, 7, 6, 5, 4\}$ into two bags. GNUMP would proceed as follows.*

[1]The style of presentation follows [Kor95b].

Numbers Remaining	*Partial Partition*	Δ
$\{8,7,6,5,4\}$	-	-
$\{7,6,5,4\}$	$\{8\}\{\}$	*8*
$\{6,5,4\}$	$\{8\}\{7\}$	*1*
$\{5,4\}$	$\{8\}\{7,6\}$	*5*
$\{4\}$	$\{8,5\}\{7,6\}$	*0*
-	$\{8,5,4\}\{7,6\}$	*4*

GNUMP takes $O(n)$ time to k-partition a bag of n elements. It also requires $O(nkm)$ space, where m is the length of the maximum element of S in binary. Obviously, GNUMP is not complete, as it is shown by the above example (the minimum possible difference is 0 but GNUMP terminates with a difference of 4).

4.2.2 The Set-Differencing Algorithm for NUMP

Another *polynomial-time approximation* algorithm is the *set-differencing* method of Karmakar and Karp (also known as *KK heuristic*) presented originally in [KK82]. We present this algorithm as studied in [KK82, Kor98, Kor95a, Mer98].

Let us call this algorithm KKNUMP following our naming conventions. We consider the case that we want to partition a bag S in two bags ($k = 2$). Initially, the numbers are sorted in decreasing order. Then KKNUMP proceeds as follows. The largest two numbers are picked and replaced by the absolute value of their difference. The resulting bag is ordered and the same operation is iterated until there is only one number left. This number is actually the difference between the final two bags.

The algorithm is illustrated by the following example[2].

Example 4.4 *Suppose we want to partition the bag $S = \{8,7,6,5,4\}$ in 2 bags. KKNUMP proceeds as follows.*

Bag	a	b	$\lvert a-b\rvert$	*Partition*	
$\{8,7,6,5,4\}$	8	7	1	$\{7,5,4\}$	$\{8,6\}$
$\{6,5,4,1\}$	6	5	1	$\{5,4\}$	$\{6,1\}$
$\{4,1,1\}$	4	1	3	$\{4\}$	$\{1,1\}$
$\{3,1\}$	3	1	2	$\{3\}$	$\{1\}$
$\{2\}$				$\{2\}$	$\{\}$

We could say that in the first pass, the algorithm works down the table filling in the first four columns. Then, it proceeds bottom-up to fill in the last two columns of the table.

Up to now, the way we have presented KKNUMP leads us to the final difference, but tells nothing about how to compute the elements of the final bags. In

[2]The style of presentation follows [KK82].

Example 4.4 KKNUMP replaces each pair difference by the greatest number of the pair and puts the other number of the pair in the opposite bag. As pointed out in [Kor98] the computation of the two bags can also be implemented by building a graph with nodes corresponding to the given numbers, and links corresponding to the constraint that two numbers should not be in the same bag. Then, one can 2-color the given graph to obtain the solution.

The running time of KKNUMP is $O(n \log n)$ to k-partition a bag of n elements. It also requires $O(nkm)$ space, where m is the length of the maximum element of S in binary. Of course, KKNUMP is not complete, but it performs much better than GNUMP as shown in [Kor98]. The intuitive reason is that the continuous differencing operations of KKNUMP results in a much smaller difference that the difference produced by GNUMP which is in the order of the smallest given number.

4.2.3 A Pseudo-Polynomial Algorithm Based on Dynamic Programming

Another algorithm for NUMP is based on *dynamic programming*.

The algorithm requires an array $a[L]$ where $L = \frac{\Sigma}{2} + 1$, where Σ is the sum of the given numbers. The algorithm enumerates all possible subset sums to determine the achievable subset sum closest to half the total sum. If an entry $a[i]$ is equal to one, that means that the subset sum i is achievable. We describe here the algorithm as presented by [Kor98, GW98]. The algorithm starts with the array initialized to all zeros and sets $a[0] = 1$. Then for each number x in the original set, it scans the array, and for each element $a[i]$ equal to one, it sets $a[i + x]$ equal to one. The algorithm continues until all numbers are exhausted.

After, array a is constructed, the algorithm returns the entry closest to half the total sum that is equal to one. Thus, the algorithm finally finds the optimal subset difference it can be achieved because it enumerates all possible subsets. The algorithm is illustrated by the following example.

Example 4.5 *Suppose we want to partition* $S = \{8, 7, 6, 5, 4\}$ *in 2 bags. The algorithm computes the size of the array as the half of the sum of all the numbers devided by 2 and added one. Thus, the array will be of size* $L = \frac{(8+7+6+5+4)}{2} + 1 = 16$. *Then, the algorithm initializes all the entries to 0 and sets* $a[0] = 1$.

1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Now the algorithm starts to enumerate all possible subset sums. For number 8 *the algorithm sets the element* $a[0 + 8] = 1$.

1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

For number 7 *the algorithm sets the elements* $a[0 + 7] = 1$ *and* $a[8 + 7] = 1$.

1	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*

Up to now, all the possible subset sums are 7, 8 *and* 15*, meaning that we could put numbers* 7 *and* 8 *either to the same, or to different bags.*

Finally, the array will be as follows, indicating all the possible subset sums.

1	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*

The algorithm described above runs in time polynomial in nL. But notice that the size of the input is $O(n \log L)$ and nL is *not* bounded by any polynomial function of this quantity. Thus, this algorithm is only practical for partitioning bags with a small number of values, or problems where the individual numbers have limited precision. Additionally, it only returns the optimal subset difference, but not the subsets themselves.

4.2.4 Complete Algorithms for NUMP

Korf was the first to study in detail *complete* algorithms for NUMP. A *complete* algorithm guarantees to find a solution when there is one. In [Kor95b, Kor95a, Kor98], Korf presents two complete algorithms for NUMP; one based on GNUMP, let us call it CGNUMP, and one on KKNUMP, called CKKNUMP. Korf's algorithms traverse a search-tree corresponding to NUMP. This tree is defined in such a way that the first solution found by CGNUMP (resp. CKKNUMP) is the one returned by GNUMP (resp. KKNUMP). This solution is usually non-optimal. By allowing the algorithms to run for more time, the solution to the problem can be improved. Eventually, these algorithms always find and verify the optimal solution. Concerning their performance, CKKNUMP dominates CGNUMP over all cases. Namely, CKKNUMP appears asymptotically more efficient than CGNUMP when no perfect partition exists, and is orders of magnitude more efficient when there are perfect partitions. The exact details of these algorithms are given in [Kor98].

Various local search algorithms for number partitioning are considered in [JAMS91, RNMS96], while genetic algorithms are studied in [BM99].

In what follows we use greedy and complete algorithms proposed by Korf as a basis for designing corresponding algorithms for FLA. In Section 4.3 we present a greedy algorithm. In Section 4.4 we present an complete algorithm based on GNUMP.

This algorithm we present can also be classified as *anytime* as Korf has also suggested in [Kor98]. An *anytime algorithm* may improve the quality of the solution returned, if it is allowed to run for enough time (e.g., local search algorithms). The anytime feature of our algorithm is especially useful in our application domain, since a node in a P2P system running the complete algorithm presented below can always decide whether to run it to completion based on some criterion (e.g., fairness achieved by that time or resources available to the node, etc.).

When the algorithm is terminated, the node can perform the required reorganization of the system based on this possibly imperfect allocation.

4.3 A Greedy Algorithm for FLA

Our greedy algorithm for FLA, let us call it GFLA, corresponds to the greedy algorithm for NUMP proposed in [Kor98]. We first sort the documents in decreasing order of popularity. Then we always place the next unallocated document to the peer with the smallest load so far, until all documents have been allocated.

The algorithm always makes the choice that looks best at the moment: whenever a new document is processed, the highest increase to fairness index is achieved if the document is allocated to one of the peers with the smallest load. This is shown by an easy calculation presented in Proposition A.1 of Appendix A. Similarly with Korf's algorithm, GFLA needs $O(n)$ time and $O(nkm)$ space where m is the length in binary of the maximum document popularity.

4.4 A Complete Algorithm for FLA

To find an optimal solution to FLA one can use the obvious exhaustive algorithm: consider all possible allocations of the n documents to the k peers and finally return the allocation that maximizes the fairness index. The running time of this algorithm is $O(k^n)$, with $k \geq 2$, since it amounts to searching a tree with branching factor k and depth n.

The algorithm we present in this section for solving the FLA problem is based on making the greedy algorithm, GFLA, complete as proposed in [Kor98], and it is a substantial improvement over the exhaustive algorithm sketched above. Our algorithm, called CGFLA, works as follows. First, we sort the documents in decreasing order of popularity, as we have done for GFLA. Then, we consider each of the n documents in turn and place it in each of the k different peers, generating a *k-ary search tree* which is searched depth-first. The search is ordered and the greedy heuristic is used whenever we expand a node: the leftmost branch emanating from this node places the next document to the peer with the *smallest current load*, the next branch places it to the peer with the next higher load, etc. Thus, the first solution found is always the solution returned by the GFLA. Furthermore, the search tree is pruned in the following ways:

1. We never place a document in more than one peer with current load 0. In this way, we avoid generating allocations that are essentially permutations of each other. Therefore, we reduce the search space and produce only $O(k^n/k!)$ distinct *k-way* allocations of the n documents according to Proposition A.2 of Appendix A. More generally, we never place a document in more than one peer with the same current load.

2. The last document is only allocated to the peer with the smallest current load, as this is the best we can do.

3. At each node of the serach tree, we use *branch-and-bound*, and maintain the greatest fairness index $\mathcal{F}(\overline{y})$ found so far for a complete allocation $\overline{y}$. Given the peer with the largest current load in a *partial allocation*, the best we can do is to bring the load of each of the remaining peers up to the largest current value. To see if this is possible, we sum the current load of all peers except the one with the largest load and we add to this number, the sum of the popularities of the remaining unallocated documents (denoted by r), to calculate the load that the $k-1$ peers have to share. Thus, assuming that numbers on allocation vector $\overline{z}$ are in non-increasing order, the quotient

$$A = \frac{\sum_{i=2}^{k} x_i + r}{k-1}$$

represents the *average load* that can be allocated to any of the $k-1$ peers except the one with the largest current load.

Formally, let $\overline{x} = (x_1, \ldots, x_k)$ be the current load allocation to the k peers, where $x_1 \geq x_2 \geq \ldots \geq x_k$. If $x_1 - A > 0$ then we consider allocation $\overline{z} = (x_1, A, \ldots, A)$, i.e., the first peer is given the largest current load and the remaining $k-1$ peers are given the average load A. The fairness index for this allocation is the following:

$$\mathcal{F}(\overline{z}) = \frac{(x_1 + (k-1)A)^2}{k(x_1^2 + (k-1)A^2)}$$

If $\mathcal{F}(\overline{z})$ is greater than the maximum fairness index value achieved so far (denoted by $\mathcal{F}(\overline{y})$), then this is the best complete allocation we could achieve. Of course, there is no guarantee that we could actually achieve this allocation, since it represents a perfect solution to a $k-1$-way FLA problem, but it gives us an upper bound.

Thus, if the fairness index $\mathcal{F}(\overline{z})$ computed above is smaller than the best fairness index found so far, we can prune this branch since we cannot improve the existing suballocation.

4. Once an optimal allocation is found, the algorithm is terminated. We compute the optimal solution to the problem as follows. A perfect allocation has a fairness index value equal to 1 whenever the sum of the popularities of all the documents is divisible by k. If the sum of the popularities of all documents is not divisible by the number of peers, we take the integer part of the division, calling it q, and the remainder, calling it ρ. Then, the highest optimal fairness index value we can achieve, according to Proposition

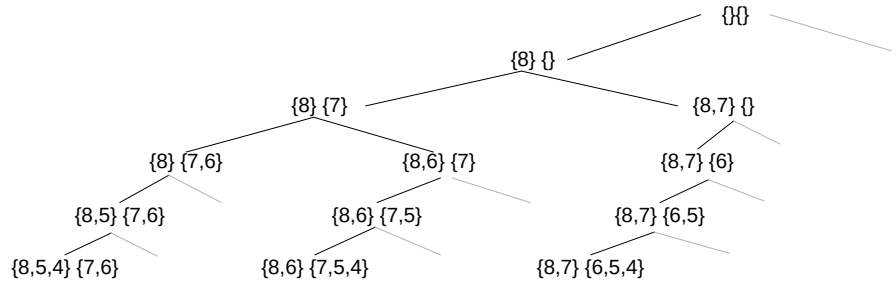

Figure 4.1: Complete algorithm based on the greedy heuristic for FLA

A.3 of Appendix A, is:

$$\frac{[(k-\rho)q+\rho(q+1)]^2}{k[(k-\rho)q^2+\rho(q+1)^2]}$$

In summary, CGFLA searches a tree depth-first, from left to right, requiring exponential time and $O(nkm)$ space, where m is the length in binary of the maximum popularity. The first allocation found is the one computed by GFLA, and the worst case is to generate at most $O(k^n/k!)$ allocations (see Proposition A.2 of Appendix A).

Returning to our example, where the list of documents is $(8, 7, 6, 5, 4)$ and we have 2 peers, CGFLA would only generate the nodes shown in Figure 4.1. Dashed lines indicate a pruned part of the tree, in accordance with the heuristics presented above.

The pseudocode of the algorithm is presented in Figure 4.2.

4.5 Experimental Results

In this section, we present the results of the evaluation carried out for both GFLA and CGFLA. We, firstly, determine the behavior of CGFLA and following, we compare the algorithms in terms of their solution quality, that is the number of generated nodes and the fairness index achieved.

4.5.1 Phase Transitions

While NP-complete decision problems are general understood to be very hard to solve, it is known that for many of these problems, typical instances are easy to solve. There is no contradiction here, since computational complexity theory gives us only a *worst case analysis* for a whole class of problems. This situation raises the question what instances of NP-complete decision problems are really hard.

```
algorithm CGFLA
input P = {p_1, p_2, ..., p_k}                                  //a set of peers
      D = {d_1, d_2, ..., d_n}                                  //a set of documents
      n                                                         //the number of documents

sort D in decreasing order
return SEARCH(P, n, D)

function SEARCH (P, n, D)
        compute f(b̄)                                            //the best fairness index we expect to have
        allocations = 0                                         //counts all possible final allocations
        Load(p_1) = p(d_1)                                      //allocate the first document to the first peer
        x̄ = {Load(p_1), 0, ..., 0}
        currentNode = root                                      //initialize the search using x̄
        BranchFactor(currentNode) = 2
        VisitedBranches(currentNode) = 0
        if D = ∅ then return x̄                                  //return the current allocation
        while (true) do
                p_i = {p_μ∈[1,k] : min(x_μ) ∧ x_μ ≠ 0 ∧ Load(p_μ) = x_μ}
                while VisitedBranches(currentNode) < BranchFactor(currentNode) do
                        for each document d_j ∈ D do //d_j to p_i according to the greedy heuristic
                                if (VisitedBranches(currentNode) = 0) and (∃x_μ∈[1,k] = 0) then
                                   p_i = {p_μ : Load(p_μ) = x_μ}
                                   Load(p_i) = Load(p_i) + p(d_j)
                                else
                                   Load(p_i) = Load(p_i) + p(d_j)
                                   p_i = {p_μ : Load(p_μ) = x_i−1}
                                   VisitedBranches(currentNode) = VisitedBranches(currentNode) + 1
                                if ∃F(m̄) then      //a complete allocation already exists
                                   A = (Σ_{i=2}^{k} Load(p_i) + Σ_{i=j+1}^{n} p(d_i))/(k − 1)
                                   ȳ = {Load(p_1), A, ..., A}
                                   compute f(ȳ)
                                   if f(ȳ)< f(m̄) then
                                           p_i = {p_μ : Load(p_μ) = x_i−1}
                                           VisitedBranches(currentNode) + +
                                   else
                                           sort(x̄)
                                           currentNode = currentNode.child
                                           if |D| = 1 then BranchFactor(currentNode) = 1
                                           else
                                              if ∃x_μ∈[1,k] = 0 then
                                                 BranchFactor(currentNode) + +
                                              else BranchFactor(currentNode) = k
                        allocations = allocations + 1
                        x̄ = {Load(p_1), Load(p_2), ..., Load(p_k)}
                        compute F(x̄)
                        if f(x̄) = F(b̄) then return x̄
                        if ∄F(m̄) or F(x̄) > F(m̄) then m̄ = x̄
                        if (D = ∅) ∧ (allocations ≠ k^n)then
                                d_j = d_j−1
                                currentNode = currentNode.parent
                        else return m̄
```


Figure 4.2: Complete algorithm for FLA based on the greedy heuristic

In papers such as [CKT91, GMPW96, GW98], researchers from theoretical computer science, Artificial Intelligence, operations research and statistical physics have given characterizations of hard problems in terms of their *constrainedness*. The state of the art results in this area can now be summarized as follows. Problems that are *over-constrained* are *insoluble* and it is usually easy to check that this is the case. Problems that are *under-constrained* are soluble and a solution can be found easily. A *phase transition* tends to occur when problems are *critically constrained* and it is difficult to determine whether they are soluble or insoluble [CKT91]. Thus, the phase transitions of some NP-complete decision problems have been shown to follow *easy-hard-easy* patterns. Phase transition behavior of NP-complete optimization problems e.g., NUMP has also been studied [GW98] and an *easy-hard pattern* has been observed.

Let us assume that we have an *ensemble* of instances of an NP-complete problem. We assume that each instance has a state space $\mathcal{S}$ with $|\mathcal{S}|$ elements and a number Sol of these states are solutions. Points in the space can be represented by N-bit binary vectors where $N = \log_2 |S|$. Let $\langle Sol \rangle$ be the *expected number of solutions* averaged over the ensemble. In various NP-complete problems (e.g., constraint satisfaction problems) the phase transition between insoluble and soluble problems occurs when $\langle Sol \rangle \approx 1$ [SD96].

In [GMPW96], Gent et. al. have defined the *constrainedness*, κ, *of an ensemble of instances* as follows:

$$\kappa =_{def} 1 - \frac{log_2(\langle Sol \rangle)}{n} \tag{4.1}$$

[GMPW96] demonstrate that in many NP-complete problems a phase transition will occur when $\kappa \approx 1$. If $\kappa < 1$, problems are under-constrained and typically soluble. When $\kappa > 1$, problems are over-constrained and typically insoluble. The parameter κ is very useful especially because its value at the phase transition varies very little. Additionally, whatever variation there is can be modelled using finite size scaling techniques.

In [GW95, GW96, GW98] Gent and Walsh have studied the phase transition behavior of NUMP in detail concentrating on variations of the algorithms proposed by Karmakar and Karp [KK82] and Korf [Kor98]. Additionally, they have shown the way that phase transition behavior can be used to provide precise and quantitative comparisons between algorithms and heuristics. [Mer98] have presented a statistical mechanics analysis for studying phase transition behavior in number partitioning problems. In what follows we are going to illustrate the phase transition behavior of FLA using the results of [GW98] to guide us. As it turns out, in many cases we can immediately use their results in our case.

[GW98] first compute the expected number of solutions in an instance of the decision problem version of NUMP i.e., the *expected number of perfect partitions*. If we consider finding a perfect 2-partition for a bag of n numbers drawn uniformly

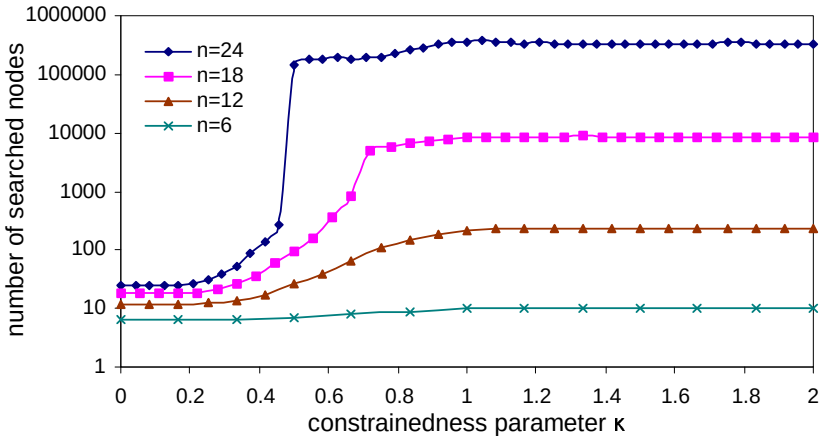

Figure 4.3: Average nodes searched by CGFLA against κ

and at random from $(0, l]$ then [GW98] show that

$$\langle Sol \rangle = \frac{2^n}{l}$$

Substituting the annealed value of $\langle Sol \rangle$ into Equation 4.1 and simplifying it follows:

$$\kappa = \frac{log_2(l)}{n}$$

The analysis of [GW98] regarding κ for NUMP holds *as is* for FLA. Thus, we will use the parameter κ in order to identify whether CGFLA exhibits phase transition behavior, and if so, to identify the phase transition region. Afterwards, we are going to use κ, if determined, in order to compare our algorithms.

Phase Transition Experiments

Let us now present the details of our experiments that show the phase transition behaviour of FLA. We generated n documents and allocate them to $k = 2$ peers using CGFLA, with n varying from 6 to 30. The popularities of the documents are integers distributed uniformly at random in the interval $(0, l]$, for $\log_2 l$ from 0 to $2n$. Were generated 1000 problem instances for each value of l and n.

In Figure 4.3, the average number of nodes searched by CGFLA is plotted against the constrainedness parameter κ. The y-axis is in logarithmic scale. The easy-hard phase transition pattern typical of optimization problems is easily identifiable. At the beginning, where κ is small, problems are in the soluble phase and are, on average, easy. Problem hardness increases as we approach the phase boundary. As expected, problems appear to remain uniformly hard in the insoluble phase away from the phase boundary.

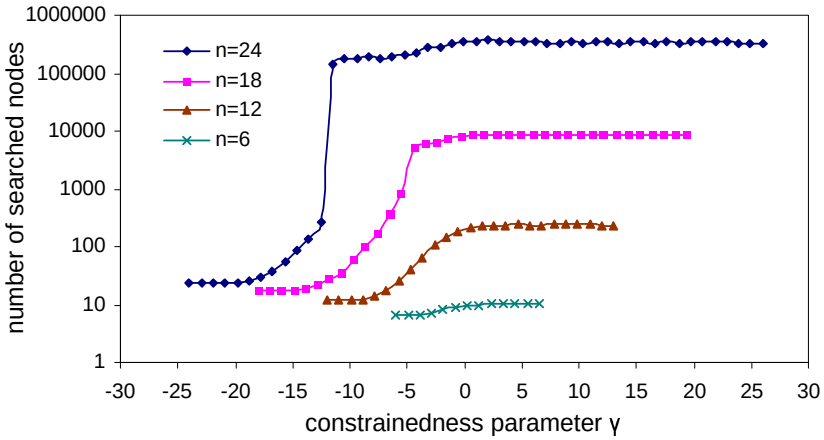

Figure 4.4: Average nodes searched by CGFLA against γ

Additionally, the transition sharpens as n increases. Thus, when $n = 6$ the phase boundary is identifiable with difficulty. As n increases, the transition becomes more obvious, and it is finally very steep for $n = 24$.

Another interesting point arising is the fact that the transition from the easy to the hard phase does not start at the same value of κ for all values of n. Thus, for $n = 6$ the average number of generated nodes starts to increase when $\kappa = 0.7$, while for $n = 24$ the number of searched nodes is increased noticeably when $\kappa = 0.5$. However, for all values of n, the insoluble phase where CGFLA exhibits uniform behavior occurs after $\kappa \approx 1$.

In Figure 4.3, it is clear that in the easy phase the nodes generated by CGFLA are comparable as n increases. For example, when $\kappa < 0.4$ CGFLA generates approximately 7 nodes for allocating 6 documents, while for allocating 24 documents it generates approximately 30 nodes. However, when $\kappa > 1$ CGFLA generates approximately 10 nodes to allocate 6 documents, and 330000 nodes to allocate 24 documents.

Let us now consider the *finite-size scaling* methods of [GW96] to see how the probability to have a perfect partition in the decision problem version of NUMP scales with problem size. [GW96] predicted that problems of all sizes will be indistinguishable around some *critical point* except for a change of scale given by a simple power-law. [GW96] suggest that

$$Prob(\text{perfect partition}) = f((\frac{\kappa - \kappa_c}{\kappa_c})n^{1/\nu}) \tag{4.2}$$

where f is a fixed function, κ_c is the critical point, and $n^{1/\nu}$ provides the change of scale. Equation 4.2 has a fixed point where κ equals κ_c and for all n the probability is the constant value $f(0)$. [GW96] has estimated κ_c as 0.96 ± 0.02. Additionally, ν has been computed as 1 ± 0.3. Finally, [GW96] defines a rescaled

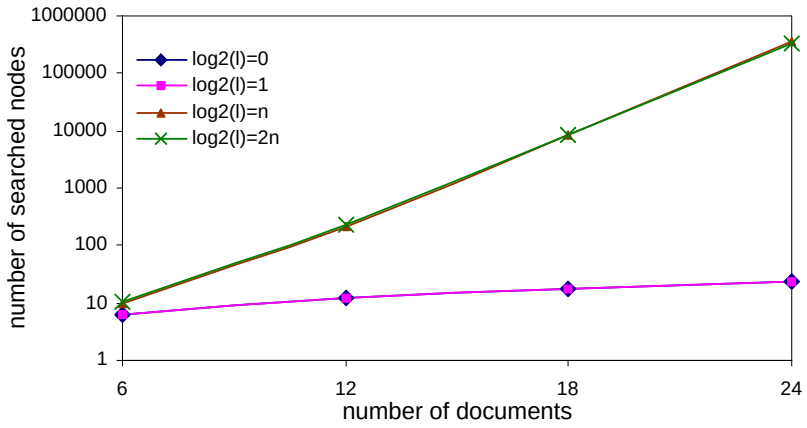

Figure 4.5: Average nodes generated by CGFLA against n

parameter γ as follows:

$$\gamma =_{def} \frac{\kappa - 0.96}{0.96} n$$

In Figure 4.4, the average number of nodes generated by CGFLA against parameter γ is plotted. This graph follows by the data in Figure 4.3 and the definition of γ and it is in correspondence with Figure 3 of [GW96].

Counting the number of generated nodes in a search algorithm is a nice machine-independent measure of algorithm performance. Let us now give a flavor of the actual time needed to solve some of the FLA instances generated above. The algorithms were implemented in C and we run our experiments on a Pentium 4 with CPU at 1.6 GHz and with 1 GByte of main memory running Linux. GFLA solved all instances of the FLA problem very quickly, in less than 1 msec. Concerning CGFLA in the easy phase, for example when $\kappa \simeq 0.3$, FLA is solved in time less than 1 msec for every value of n. In the hard region the time increases substantially. For example, when $\kappa \simeq 1.65$ CGFLA needs on average less than 1 msec when $n = 6$, 1 msec when $n = 12$, 50 msec when $n = 18$, and 1.6 sec when $n = 24$ to find the optimal solution. Thus, the actual time needed for finding the optimal solution is very small, even if 350000 nodes have to be generated.

4.5.2 Varying the Number of Documents

To compare GFLA and CGFLA regarding their time complexity, we could consider GFLA as equivalent to a search algorithm traversing the search-tree corresponding to FLA and always returning the first solution found. Then, the number of nodes generated by GFLA is the same with the number of documents, n, being allocated.

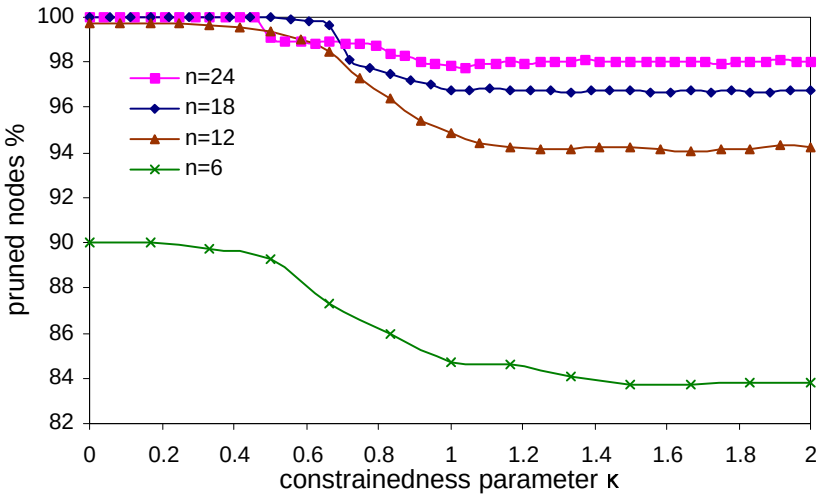

Figure 4.6: Percentage of pruned nodes against κ

CGFLA is expected to have a more complicated behavior, as it is affected by both the number of documents and the size of the popularities of the documents generated.

In Figure 4.5 we plot the average nodes searched by CGFLA to allocate n documents to $k = 2$ peers, with n varying from 6 to 24. The y-axis is in logarithmic scale. It is clear that the number of nodes searched by CGFLA increases exponentially with n. The experiment was run for some instances from both easy ($\log_2 l = 0$ or 1) and hard phases ($\log_2 l = n$ or $2n$) of our problem. As expected, CGFLA has a uniform behavior during the easy phase, as all problems there are under-constrained and they are expected to be easy. The same uniformity occurs during the hard phase, since the problems there are over-constrained.

Although the number of nodes generated by CGFLA still increases exponentially with n, the heuristics used (see Section 4.4) prune a *large part* of the search tree. The percentage of pruned nodes against parameter κ is plotted in Figure 4.6. It is clear that for all values of n only a small number of nodes is generated relatively to the initial search space. Particularly, in the easy phase of the problems, where κ is small, around 99% of the nodes are pruned when the number of documents is more than 12. In the hard phase of the problem, the percentage of pruned nodes is again very high (95%), despite the fact that the pruned nodes are less than those in the easy phase. When the number of documents is equal to 6, the curve of pruned nodes demonstrates similar behavior.

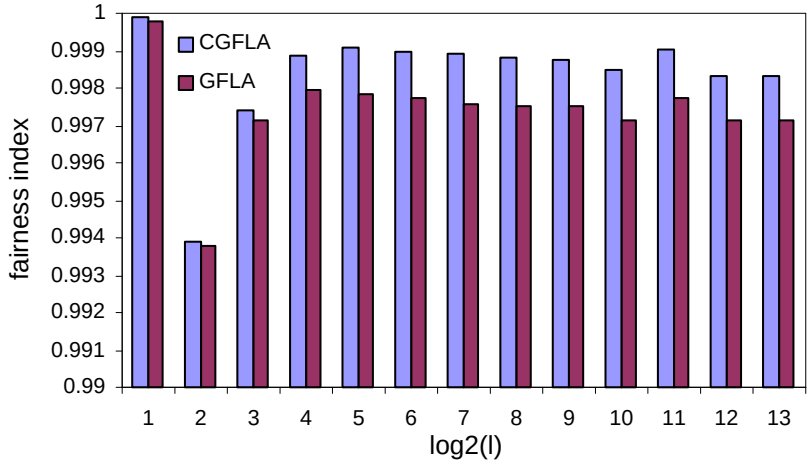

Figure 4.7: Average fairness index achieved against $log_2(l)$

4.5.3 How Fair is GFLA and CGFLA?

Now we study the values of the fairness index achieved by GFLA and CGFLA. In Figure 4.7, we plot the average fairness index achieved, for $n = 6$, $k = 2$ and $\log_2 l$ from 0 to 12, while the popularities of the documents are distributed uniformly in the interval $(0, l]$. 1000 problem instances were generated for each value of l. Note that the difference in the fairness index between the allocations returned by both algorithms is always less than 0.0015, meaning that the solution returned by GFLA is always very near to the solution returned by CGFLA.

Figure 4.8 gives us another interesting picture. We plot the difference in the fairness index achieved by GFLA and CGFLA, for $n = 6$ and $\log_2 l$ from 0 to $2n$, against the constraindness parameter κ. 1000 problem instances were generated for each value of l. We observe that the difference is *one order of magnitude less* in the soluble phase compared to the difference in the insoluble phase. The difference between the solutions increases as we approach the phase boundary and it remains relatively constant (≈ 0.0013) in the insoluble phase. It is again very easy to see that the difference is *very very small*.

Another point shown in Figure 4.7 which deserves explanation is the relatively low fairness index achieved for $log_2(l) = 1$. While the fairness index is higher than 0.998 for all other values of $log_2(l)$, it drops to 0.9938 for $log_2(l) = 1$. When $log_2(l) = 1$, $l = 2$ and thus, the popularities of the documents are generated in the interval $(0, 2]$, meaning that the popularity of each document is either 1, or 2. In this case, there will be instances where the sum of the documents is divisible by 2, but the solution to the FLA problem will not be equal to 1. For example, if we have two peers and a bag of documents $\{2, 2, 2\}$, the best that it could be done is to partition these documents into bags $\{2, 2\}$ and $\{2\}$. Thus, considering

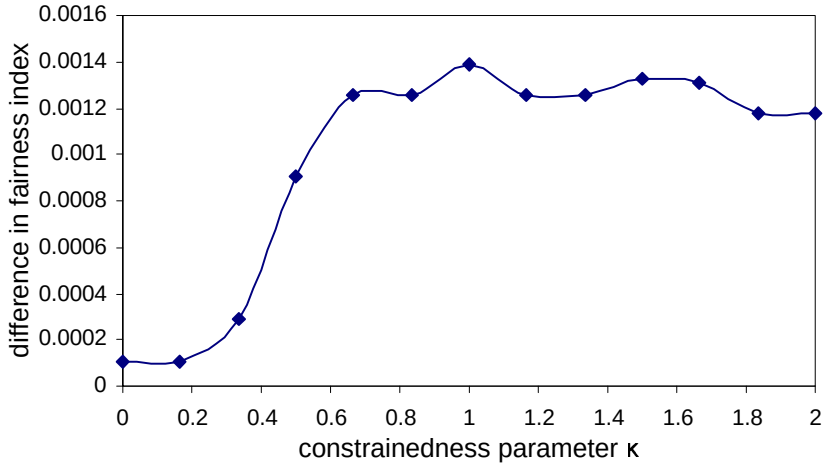

Figure 4.8: Difference in the fairness index achieved against κ

that this situation will occur in half of the problem instances, it becomes clear why the fairness index appears relatively decreased when $log_2(l) = 1$.

4.5.4 Varying the Number of Peers

Up to now, all the experiments run were allocating the set D of n documents in 2 peers. Let us now consider the case where the number of peers is $k > 2$. We expected the average number of nodes searched by CGFLA to be affected by the number of peers, as the whole search space is $O(k^n)$ (Section 4.4). Additionally, due to the analogy of the k-way FLA and the NUMP problems, a similar phase transition behavior was again expected. More specifically, we again expect a phase transition to occur around $\kappa \approx 1$.

[GW98] defines the constrainedness parameter, κ, for multi-way partitioning using the same technique presented in Section 4.5.1. The expected number of perfect partitions in this case is the following:

$$\langle Sol \rangle = k^n (\frac{1}{2})^{(k-1)\log_2(l)}$$

Substituting this value into Equation 4.1 gives the following expression:

$$\kappa = \frac{(k-1)log_k(l)}{n}$$

In Figure 4.9, we plot the average number of nodes searched by CGFLA against the constrainedness parameter, κ, with the number of peers varying from 2 to 6 and for 12 documents. The y-axis is in logarithmic scale. The popularities of the documents were distributed again uniformly in the interval $(0, l]$, with $log_2(l)$ from 0 to $2n$. 1000 problem instances were generated for each value of l.

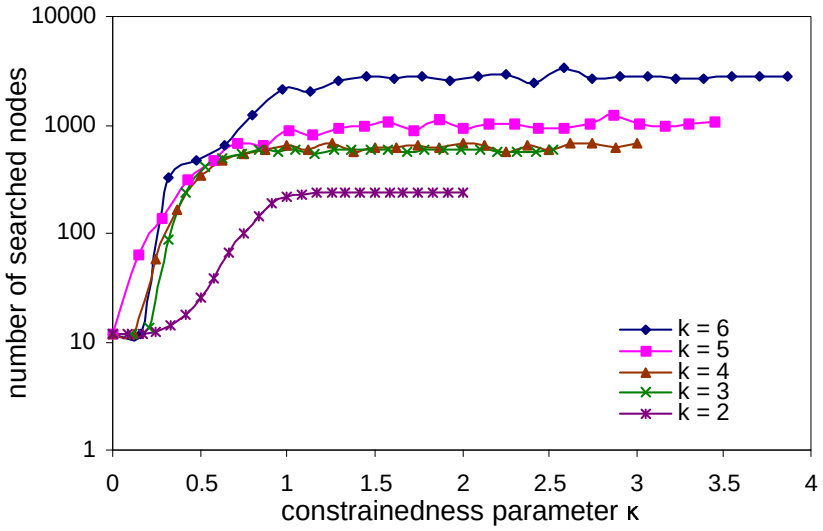

Figure 4.9: Average nodes searched by CGFLA against κ

As it was expected, we take a similar behavior for all values of k. The phase transition is easily identifiable. Problems are in the easy phase when κ is small, while their hardness increases as we approach the phase boundary. However, the greater the number of peers is, the earlier the phase transition begins. Thus, for $k = 3$ the number of average nodes searched starts to increase when $\kappa \approx 0.42$, while for $k = 4$ the increase in the number of nodes starts when $\kappa = 0.375$. Additionally, the transition becomes sharper as the number of peers increases. As expected, all problems remain uniformly hard in the insoluble phase.

4.5.5 Using Zipf Distributions

In this section we present experiments, where the popularities of the generated documents follow both Zipf and uniform distributions. As we have said in Chapter 3 the probability of a request for the i-th most popular object on the Web is proportional to $1/i$. The values of the Zipf parameter θ, which have been found to adequately capture the distributions of accesses for web objects, range between 0.6 and 0.8 [ABCdO96, BCF$^+$99]. Based on these claims we have generated popularities following the Zipf distribution with $\theta = 0.8$.

In Figure 4.10, we plot the average number of nodes searched by CGFLA against the constrainedness parameter, κ, when allocating 12 documents to $k = 2$ peers. The y-axis is in logarithmic scale. The popularities of the documents were distributed either uniformly, or following the Zipf distribution in the interval $(0, l]$, with $log_2(l)$ from 0 to $2n$. 1000 problem instances were generated for each value of l.

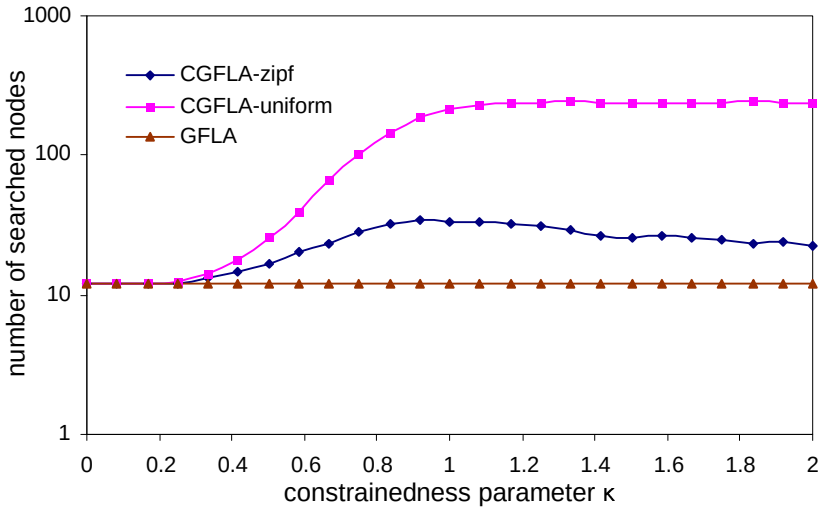

Figure 4.10: Average nodes searched against κ

The behavior of the average number of nodes searched by CGFLA in the case that the generated popularities obey Zipf's law is very similar to the case that the popularities are distributed in a uniform way. In Figure 4.10 the search space displays a phase transition that sharpens around $\kappa \approx 1$. However, the number of generated nodes is surprisingly less in the case that we used the Zipf distribution than in the case where the popularities were spread uniformly in the same interval. Specifically, the average number of nodes generated by CGFLA is comparable to GFLA when we used the Zipf distribution. This discovery makes us agree with the claim of [GW98] that instances of NUMP obeying Zipf's law are much easier to partition than those drawn uniformly.

Additionally, we expect that the maximum value of the fairness index achieved is less in the case that the popularities of the documents follow the Zipf distribution than in the case they are uniformly distributed. By definition, using the Zipf distribution means that there will be some very popular documents, while the most of them will have a very small probability to be asked. Thus, there will be generated some great numbers (corresponding to the very popular documents), whereas the big mass of the generated numbers will be relatively small (corresponding to the not so popular documents). This kind of unevenness to the generated popularities results in rather spottiness in the final subsets. The experiments strictly verify our assumptions. However, even with the Zipf distribution the values of fairness index are still very very high ($> 0,991$).

4.6 Conclusions

In this chapter we concentrated on distributing the load fairly among the peers in a P2P system. We described the architecture of our system, defined formally the FLA problem and finally, proposed two algorithms for solving it: GFLA and CGFLA. The algorithms were evaluated and compared experimentally in terms of their solution quality.

Initially, effected by the algorithms in the NUMP paradigm, we tried to detect whether CGFLA presents a phase transition behavior. All our experiments resulted in the fact that CGFLA moves from an easy to a hard phase depending on the problem size and on the interval within document popularities lie. Additionally, the experimental results comparing GFLA and CGFLA indicate that although the solution returned by GFLA is not the optimal, it is always very close to it. Of cource, GFLA is, as expected, quicker than CGFLA no matter the size of the problem.

Chapter 5

More Sophisticated Models of Resources

In the previous chapter we defined formally the FLA problem and we presented algorithms and experimental results for solving this problem. Up to now, we have dealt with very simple models of P2P systems where the only concepts have been the *peers* and the *documents* contributed by them. Each document has been associated with a popularity that gives us a measure of the load to be suffered by a peer contributing this document. In this section we will use more realistic models of P2P system resources proposed originally in the context of the hierarchical P2P content sharing system CLUSTER proposed by our research group in [TXK02, TXKN03, Xir03]. CLUSTER imposes a *hierarchical system structure* based on the concept of *peer clusters* and takes into account the *heterogeneity of peers* by using sophisticated models of resources allowing for peers with varying capabilities. The fact that P2P system nodes consist of very heterogeneous computers was first discussed in [SGG02], but so far very few papers have taken this into account. In this chapter we will introduce the main concepts of CLUSTER and its resource models as discussed in [TXK02, TXKN03, Xir03]. Then, we will utilize algorithms similar to the ones developed in Chapter 4 to study a particular kind of load balancing problem emphasized in this context.

The rest of the chapter is organized as follows. In Section 5.1 we introduce CLUSTER and define the load balancing problem to be studied. Section 5.2 presents the resource models of CLUSTER. Section 5.3 presents a greedy and a complete algorithm for load balancing. In Section 5.4 we present the results of the experimental evaluation carried out for these algorithms. Finally, Section 5.5 concludes this chapter.

5.1 CLUSTER: A Hierarchical P2P Content Sharing System

We now introduce a formal model of CLUSTER as presented in [TXKN03]. CLUSTER can be used to store a set D of n documents with popularities given by a function $p : D \rightarrow \mathbb{N}$. The documents are contributed by k peers. Each peer i may contribute more than one document to the system. The documents of CLUSTER may belong to a predefined set C of λ *semantic categories* (e.g., topics). This can be captured by a function $t : D \rightarrow C$ that maps each document to a semantic category. The *popularity of a category* c, denoted by $p(c)$, is defined to be the sum of the popularities of the documents belonging to c i.e.,

$$p(c) = \sum_{d \in c} p(d).$$

In general, the popularity of a set of documents D_i will be denoted by $p(D_i)$ and defined as

$$p(D_i) = \sum_{d \in D_i} p(d).$$

Finally, the *popularity of a set of categories* S_i, denoted by $p(S_i)$, is defined as

$$p(S_i) = \sum_{c \in S_i} p(c) = \sum_{c \in S_i} \sum_{d \in c} p(d).$$

Table 5.1 lists our notation for the various parameters of CLUSTER that will be used in the rest of this chapter.

The basic concept of CLUSTER that makes it different from other P2P content sharing systems, is that it relies on a *two-level hierarchical organization* of its peers. Peers in CLUSTER are logically organized into m *clusters* with the possibility of having peers that belong to more than one cluster. In the architecture proposed in [TXKN03], clusters of nodes form storage collectives/repositories, and each cluster can store and thus serve requests for documents belonging to one or more document categories. Each category may belong to *only one cluster.* The nodes are assigned to clusters *according to the categories* of the documents they contribute. So, depending on how the categories are assigned to clusters, a peer may belong to more than one cluster if it contributes documents associated with more than one category.

Given this logical system model, the load balancing task in CLUSTER can be seen as a two-level problem. Initially, fair load distribution needs to be ensured among all clusters of the system. [TXKN03] refer to this property as *inter-cluster load balancing*. Additionally, fair load distribution must be ensured among all peers belonging to the same cluster. [TXKN03] refers to this as *intra-cluster load balancing*. We have achieved *global load balancing*, when we have achieved

Symbol	Meaning
P	set of peers
D	set of documents
C	set of categories
P_i	i-th cluster of peers ($P_i \subseteq P$)
S_i	set of categories assigned to cluster i
T_j	set of categories of documents contributed by peer j
D_j	set of documents stored by peer j
D_{jc}	set of documents stored by peer j and belonging to category c ($D_{jc} \subseteq D_j$)
k	number of peers
n	number of documents
λ	number of categories
m	number of clusters
k_i	number of peers in cluster i
u_j	processing capacity of peer j
U_i	processing capacity of cluster i
v_{jc}	part of processing capacity of peer j that is carried by category c
v_c	total processing capacity carried by category c

Table 5.1: CLUSTER model parameters

inter-cluster and intra-cluster load balancing at the same time. In this chapter, we concentrate on the problem of inter-cluster load balancing. Intra-cluster load balancing is achieved in CLUSTER using various well-understood replication techniques that are exploited by the query answering protocols and will not be considered here. The interested reader should consult [TXKN03] for more details.

5.1.1 Inter-cluster Load Balancing: Formal Problem Formulation

The inter-cluster load balancing (ICLB) problem in CLUSTER can be defined as follows [TXKN03]. We want to partition a set P of k peers into m clusters so that the following two constraints hold:

1. Documents belonging to the same category are assigned to a single cluster.
2. Load is allocated fairly among clusters.

The first constraint enables us to group together all documents of the same category and treat them as a *single logical entity*. If the first constraint is satisfied then the second constraint can be taken into account by designing protocols operating at the level of categories (as opposed to the level of documents). Thus, a rewording of ICLB is: "Partition λ document categories into m clusters achieving fair allocation of load among clusters". This statement of ICLB makes it very

similar to the FLA problem studied in Chapter 4. If the *load* of a cluster P_i is denoted by $load(P_i)$, then formally ICLB can be defined as follows.

Definition 5.1 *[ICLB] Let P, D and C be a set of peers, documents and categories in CLUSTER with the properties and constraints given above. We would like to find a partition of P into m clusters $P_1, P_2, \ldots, P_m$ so that the function*

$$\mathcal{F}(load(P_1), load(P_2), \ldots, load(P_m)) = \frac{(\sum_{i=1}^{m} load(P_i))^2}{m \sum_{i=1}^{m} (load(P_i))^2} \tag{5.1}$$

is maximized*, where $\mathcal{F}$ is the fairness index function of Definition 3.2.*

The next section discusses various ways to define the concept of load for a cluster of peers (i.e., the quantities $load(P_i)$ in the above formula) culminating in the most realistic model of Section 5.2.4 where peer heterogeneity is fully taken into account. Then we will use search algorithms like the ones developed in Chapter 4 to perform fair load allocation in CLUSTER.

5.2 Accounting for Heterogeneous Peers in CLUSTER

In this section we present the detailed models of CLUSTER that allow us to account for heterogeneous peers. Heterogeneity is captured by modelling explicitly the different processing power and different storage capacity of peers. Our presentation follows [TXKN03] almost verbatim, but our notation is slightly different (we believe it is more intuitive; see Table 5.1).

5.2.1 Peers with Identical Capabilities Contributing Documents of a Single Category Each

For reasons of presentation, let us first make the following restrictive assumptions:

1. Peers in CLUSTER are identical (i.e., they have equal processing and storage capacity).

2. All documents contributed by each peer belong to the same category.

The second constraint in the informal definition of ICLB (Section 5.1.1) now implies that all documents contributed by a peer will be logically organized into the same cluster. Therefore no peer will belong to two different clusters.

As in Chapter 4 where we deal with single peers, the load of a cluster will be determined by the *popularity* of the documents contributed by its peers. Then, at a first approximation, the load of a cluster P_i can be the sum of the popularities

of the documents belonging to the peers of P_i. This sum is the same with the sum of the popularities of the *set of categories* S_i of the documents of cluster P_i. In other words, $load(P_i) = p(S_i)$. However, not all clusters are of equal size. Thus, we choose to *normalize* the above expression for $load(P_i)$ with respect to the number of peers k_i that are members of P_i and arrive at the following expression:

$$load(P_i) = \frac{p(S_i)}{k_i} \tag{5.2}$$

The following subsections relax the above restrictive assumptions one by one, and generalize the above formula for load of a cluster.

5.2.2 Peers with Different Processing Capacity

In the general case, the peers of a P2P system will not have the same processing capabilities as shown by the measurement study of [SGG02]. To accommodate this fact (i.e., relax the first assumption of Section 5.2.1), we model each peer as having *processing capacity* u_j, which reflects the processing power as well as the networks capabilities of a peer (bandwidth). We can now define the *processing capacity* U_i of cluster i to be the sum of the processing capacities of all the peers contributing documents to i. Thus

$$U_i = \sum_{j \in P_i} u_j$$

where u_j is the processing capacity of peer j and P_i is the i-th cluster.

Now we can refine Equation 5.2 giving the load of a cluster P_i by dividing $p(S_i)$ by P_i's processing capacity instead of P_i's number of peers:

$$load(P_i) = \frac{p(S_i)}{U_i} \tag{5.3}$$

5.2.3 Peers that Contribute Documents Belonging to Multiple Categories

In general, a peer in a content sharing system can contribute a number of documents, and these documents can belong to different categories. If these categories are assigned to the same cluster, then there is no need to change Equation 5.3 defining the load of a cluster in Section 5.2.2.

If a peer contributes documents belonging to categories that are assigned to *different clusters*, the peer can be modelled as belonging *simultaneously to all these clusters* and contributing different parts of its processing capacity to them. The processing capacity of a peer j that is contributed to a specific cluster i is analogous to the popularities of the categories stored by j and assigned to cluster i. Consider, for example, a peer j having processing capacity u_j and contributing

documents belonging to categories c_1 and c_2, which are assigned to clusters P_1 and P_2 respectively. For the needs of intra-cluster load balancing, [TXKN03] initially assumes that peers store all documents of the categories assigned to their cluster. In this case, because i belongs to P_1 and P_2 it will store all documents in the categories assigned to P_1 (i.e., the categories of set S_1 in the notation of Table 5.1) and all documents in the categories assigned to P_2 (i.e., the elements of S_2 in our notation). Thus, peer j will contribute

$$u_j \frac{p(S_1)}{p(S_1) + p(S_2)}$$

of its processing capacity to cluster P_1, and

$$u_j \frac{p(S_2)}{p(S_1) + p(S_2)}$$

of its processing capacity to cluster P_2.

We can now generalize the above discussion. If u_j is the processing capacity of peer j and T_j is the set of categories of documents stored by peer j, then the load of this cluster can be defined as follows:

$$load(P_i) = \frac{p(S_i)}{\sum_{j \in P_i} u_j \frac{p(S_i)}{p(T_j)}} = \frac{1}{\sum_{j \in P_i} \frac{u_j}{p(T_j)}}$$

5.2.4 Peers with Different Storage Capacities

In P2P systems, peers (i.e., user computers) may have very different storage capacities and thus, they may not be able to store (or they may not wish to store) all documents of the categories to which they contribute. In this case, our formula for the load of a cluster can be derived as in Section 5.2.3 but now our analytical models of load have to take into account *individual documents* and not refer simply to categories.

As previously, a peer j will belong to a cluster i whenever j *contributes* documents to a category assigned to i. We can therefore say that categories *carry* peers and their processing capacity to clusters. Peer j will *store* all the documents it contributes, and it may also choose to store some additional documents belonging to categories assigned to the clusters that j belongs to. The storage capacity of peer j is *implicit* in this choice of additional documents.

Thus, if we know that a peer j stores the set of documents D_{jc} belonging to category c[1], we can compute the part of the processing capacity of j that is *carried* by c as in Section 5.2.3. This processing capacity will be denoted by v_{jc}

[1]Peer j has to own and contribute some of these documents to the community while the rest are stored by it for reasons of being cooperative.

and is given by the following formula:

$$v_{jc} = \sum_{c \in T_j} u_j \frac{p(D_{jc})}{p(D_j)}$$

where D_j is the complete set of documents stored by peer j (see the definitions of Table 5.1). Similarly, we can define the *total processing capacity* v_c *carried* by category c as follows:

$$v_c = \sum_{j \in P} v_{jc} = \sum_{j \in P} \sum_{c \in T_j} u_j \frac{p(D_{jc})}{p(D_j)}$$

The processing capacity U_i of a cluster P_i can now be defined as the sum of the processing capacities carried to P_i by all categories assigned to it. In our notation:

$$U_i = \sum_{c \in S_i} v_c$$

Thus, the load of cluster P_i can now be defined as follows:

$$load(P_i) = \frac{p(S_i)}{U_i} = \frac{p(S_i)}{\sum_{c \in S_i} v_c}$$

5.3 Algorithms for ICLB

Let us now discuss two algorithms for solving the ICLB problem in CLUSTER for the most general case of resource models given in Section 5.2.4. To solve ICLB, [TXKN03] proposes a greedy algorithm, called MaxFair. In what follows we are going to present MaxFair and also propose a complete algorithm based on GFLA presented in Section 4.4.

5.3.1 A Greedy Algorithm for ICLB

MaxFair takes as input a set P of peers, a set D of documents contributed by the peers in P and the number of clusters m in which the peers will be organized. MaxFair starts by grouping together all documents belonging to the same category and calculating the popularity for each category and the processing capacity carried with each category. Then, MaxFair considers each category c in turn and allocates it to one of the m clusters. This cluster is chosen to be the one that gives the highest value for the fairness index when the category is hypothetically allocated to it.

Thus, MaxFair always makes the choice that looks best at the moment: whenever a new category is processed, MaxFair computes all m possible values of the fairness index, by assigning the category hypothetically to all m clusters and

algorithm MaxFair
input $P = \{p_1,\ p_2,\ \ldots,\ p_k\}$ //a set of peers
$D = \{d_1,\ d_2,\ \ldots,\ d_n\}$ //a set of documents
$C = \{c_1, c_2,\ \ldots,\ c_n\}$ //a non-empty set of categories
$t : D \rightarrow C$ //mapping of documents to categories
m //the number of clusters
$\{u_1,\ u_2,\ \ldots,\ u_k\}$ //the processing capacity of each peer p_i

compute the popularity $p(c_j)$ of each category c_j in C
compute the processing capacity v_j for each categories c_j in C
sort the categories of C in decreasing order of popularity

for each cluster P_i initialize $load(P_i)$ to 0 **end for**

for each category $c_j \in C$ **do**
for $i := 1$ to m **do** //suppose c_j is allocated to cluster P_i and compute
$p'(S_i) = p(S_i) + p(c_j)$ //the hypothetical cluster popularity
$U'_i = U_i + v_j$ //and processing capacity
$load'(P_i) = p'(S_i)/U'_i$
$\overline{y} = \{load(P_1),\ \ldots,\ load'(P_i),\ \ldots,\ load(P_m)\}$
compute $\mathcal{F}(\overline{y})$ //need to traverse an m-positions array
$f_i = \mathcal{F}(\overline{y})$
end for
assign c_j to P_k where f_k is $max_i f_i$
$\overline{x} = \{load(P_1),\ \ldots,\ load'(P_k),\ \ldots,\ load(P_m)\}$
end for
return $\overline{x}$

Figure 5.1: A greedy algorithm for ICLB

choosing the allocation that results in the maximum value of the fairness index. MaxFair needs $O(\lambda m^2)$ time and obviously may not provide the optimal solution to the ICLB problem.

The pseudocode of MaxFair is presented in Figure 5.1.

5.3.2 A Complete Algorithm for ICLB

In order to find the optimal solution to the ICLB problem we can use the following exhaustive algorithm: consider all possible allocations of the λ categories to the m clusters and finally return the allocation that results in the maximum value of the fairness index. The running time of this algorithm is $O(m^\lambda)$, where $m \geq 2$.

We now present CICLB, a complete anytime algorithm for the ICLB problem, which is an improvement over the exhaustive algorithm. CICLB is very similar to the complete algorithm CGFLA presented in Section 4.4, but considers categories instead of documents, clusters instead of peers and cluster load instead of peer load. The input of the algorithm is as follows:

1. The set of peers P.

2. The set of documents D contributed by the peers of P. Each document is assigned to a semantic category.

3. The number of clusters m in which we want to logically organize the peers of P.

4. The set of categories C.

5. A function $t : D \rightarrow C$ mapping each document to a category.

CICLB works as follows. Initially, CICLB groups together all documents of the same category and computes the popularity of each category and the processing capacity carried with each category. Then, it sorts the categories in C in decreasing order of popularity. Afterwards, it considers each of the λ categories in turn, and attempts to place it in each one of the m different clusters, generating an m-ary search tree which is searched depth-first. The search is ordered and CICLB uses the greedy heuristic MaxFair whenever it expands a node: m values of fairness index are computed and the leftmost branch emanating from this node places the next category to be allocated to the cluster that results in the highest value for the fairness index, the next branch places this category to the cluster that results in the second greatest value of the fairness index, etc. Thus, the first solution found is always the greedy solution returned by MaxFair and the time complexity of CICLB is $O(m^{\lambda+2})$.

Additionally, we use a very simple heuristic in order to reduce the search space, in analogy to the first heuristic for CGFLA presented in Section 4.4. We never place a category in more than one empty cluster, and, thus, we avoid generating duplicate allocations that differ only by a permutation of the clusters. Therefore, we reduce the search space and produce only $m^{\lambda}/m!$ distinct m-way allocations of the λ categories (see Proposition A.2). More generally, we never place a category in more than one cluster with the same popularity and the same processing capacity. Once the complete tree is searched, the optimal allocation has been found and CICLB terminates.

Summarizing, CICLB searches a tree depth-first, left to right and the worst-case time complexity is $O(m^{\lambda+2}/m!)$. The first allocation found is the one computed by MaxFair.

5.3.3 Further Discussion

Although, GFLA presented in Chapter 4 and MaxFair presented in this section look similar, the careful reader should have noticed that MaxFair is more complicated. GFLA always places the next unallocated document to the peer with the smallest load at each step of the allocation procedure based upon Proposition

Parameter	Symbol	Baseline value
Number of documents	n	1000000
Maximum document popularity	l	300000
Number of peers	k	500000
Number of categories	λ	100
Number of clusters	m	20
Skewness for the Zipfian distribution	θ	0.8
Maximum number of processing capacity of a peer	u_{max}	10000

Table 5.2: Baseline values for the parameters used in the experiments

A.1. However, MaxFair computes at each step m values of the fairness index by allocating hypothetically categories to each one of the clusters before it makes a final decision. The same difference holds for the corresponding complete algorithms. Unfortunately, the ICLB problem has more complicated formulas than FLA for computing load so MaxFair appears to be the simplest greedy heuristic that one could devise.

5.4 Experimental Results

In this section, we present the results of the experimental evaluation carried out for both MaxFair and CICLB. We firstly discuss the framework in which all experiments were carried out, and then we analyze and compare the two algorithms.

5.4.1 Framework

Because of the hierarchical system architecture that CLUSTER imposes, we have to deal with several parameters that may affect the performance of the algorithms. The *key parameters* are the number of documents, the document popularities, the number of peers, the number of categories and the number of clusters. From the values of these parameters, we calculate all the other parameters needed. To observe the impact of each one of the above parameters to the overall performance of the algorithms under evaluation, we use a baseline set of values (shown in detail in Table 5.2). For each experiment carried out, we vary one of the parameters and see how the performance of each algorithm varies.

In the experiments we create n different documents with popularities chosen from the interval $(0, l]$. In analogy with [TXKN03] the popularities of the documents follow the Zipf distribution with skewness factor 0.8 as shown also in Table 5.2. We then generate k peers and assign to each peer u processing capacity using a Zipf distribution with skewness factor 0.8. In the context of our

experiments the processing capacity of a peer is represented by an integer in the interval $[1, u_{max}]$, where u_{max} is 10000. We choose a rather large value for u_{max} in order to model the big heterogeneity of processing capacity among peers in existing content sharing P2P system [SGG02].

Having generated the documents and the peers of the system, we assign each document to a peer using the uniform distribution. Finally, we randomly assign each document to a category using the uniform distribution. Each time a document is assigned to a category, the popularity and the portion of the processing capacity of the peer contributing the document are added respectively to the overall popularity and processing capacity of the category. The baseline parameters of Table 5.2 can be used to compute all the other parameters shown in Table 5.1, such as the set of categories T_i contributed by peer i, the processing capacity U_i of cluster i, etc.

Initially, we set up an experiment using the baselines values of the parameters shown in Table 5.2 and examine the difference in search time between MaxFair and CICLB. As expected, MaxFair terminated in a few milliseconds, whereas CICLB needed 11 hours to return the optimal solution, since it conducts an almost exhaustive search.

In the experiments of the following section we use the values of Table 5.2 for the key parameters. In each experiment we vary the parameter under consideration, while keeping the rest of the parameters constant, and study the effect to the algorithm performance. We then explain the behavior of the algorithm in terms of the fairness achieved and the number of nodes generated until the solution was found. All the experiments of the next section were run 1000 times and their results were averaged to produce a point in the graph.

5.4.2 Phase Transitions

Let us remind you that some optimization problems may exhibit phase transition behavior [GW96]. In Section 4.5 we studied and evaluated experimentally the easy-hard transition behavior of FLA. A very important parameter for detectinf this behavior was the upper bound l of the interval in which the popularities of the documents were generated. In this section we will try to identify whether ICLB exhibits a similar behavior.

Provided that all documents of the same category are grouped together and treated as a single entity, we want to partition the λ document categories into m clusters (see Section 5.1.1). We suppose that category popularities lie in the interval $(0, u]$, where the upper bound u of the interval is computed as follows.

The documents are generated in the interval $(0, l]$ following the Zipf distribution with skewness parameter 0.8. The Zipf distribution suggests that the i-th element of the distribution appears with frequency equal to $(\frac{1}{i})^{\theta}$. Provided that we generate n elements, the number of appearances of the i-th most frequent

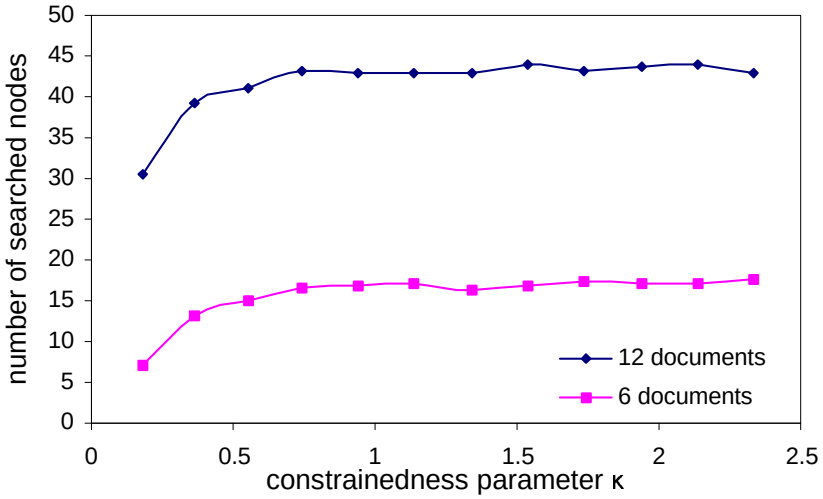

Figure 5.2: Average nodes searched by CICLB against κ

number of the distribution [Knu73] is

$$O_i = \frac{n}{i^\theta H_n} \tag{5.4}$$

where H_n is the Harmonic mean of order n i.e.,

$$H_n = \sum_{i=1}^{n} \frac{1}{i^\theta}$$

We use the function proposed in [GSE+94] to generate the document popularities; according to this function integer i gets weight proportional to $(\frac{1}{i})^\theta$. From this we understand that the i-th most frequent number is in general number i. Thus, to compute the sum of the generated popularities in the interval $(0, l]$ we take the following sum:

$$\sum_{i=1}^{l} iO_i \tag{5.5}$$

Assigning the documents to categories, the worst case scenario is that all documents will eventually belong to the same category. Thus, the maximum possible category popularity, that is the upper bound u of the interval within which the category popularities lie, equals the sum of the generated document popularities, that is computed in Equation 5.5. Substituting the baseline value of θ into Equation 5.5, we have the following:

$$u = \sum_{i=1}^{l} i \frac{n}{i^{0.8} \sum_{i=1}^{n} (\frac{1}{i})^{0.8}}$$

Having computed the upper bound u of the interval in which the category popularities lie, we can now use the same technique presented in Section 4.5.1 to detect whether ICLB exhibits a phase transition behavior. Thus, we first compute the expected number of perfect partitions of λ categories into m clusters and then the resulting constrainedness parameter of the ICLB problem.

The expected number of perfect partitions is

$$\langle Sol \rangle = m^{\lambda}(\frac{1}{2})^{(m-1)log_2 u}$$

The corresponding constrainedness parameter is

$$\kappa = \frac{(m-1)log_m u}{\lambda} \tag{5.6}$$

Substituting the value of u in Equation 5.6, the constrainedness parameter of ICLB becomes equal to:

$$\kappa = \frac{(m-1)log_m(\sum_{i=1}^{l} i\frac{n}{i^{0.8}\sum_{i=1}^{n}(\frac{1}{i})^{0.8}})}{\lambda}$$

Let us now present the details of our experiment which shows that ICLB exhibits an easy-hard pattern. In order to produce the required amount of runs we considered a case with less documents and smaller number of categories and clusters than those appear in Table 5.2. Thus, $\lambda = 6$ categories are allocated to $m = 2$ clusters, with $n = 6$ or 12, $k = 10$ and $log_2(u)$ from 0 to 2λ. The documents were generated in the interval $(0, l]$. For each value of κ and n, 1000 problem instances were generated.

In Figure 5.2 the average number of nodes searched by CICLB is plotted against the constraindness parameter κ. A phase transition behavior is detected; initially, the number of generated nodes is relatively small, while increasing and staying high for the greater values of κ.

5.4.3 Effect of Number of Documents and Peers

From the description of MaxFair and OICLB in Section 5.3 it is clear that the number of documents n has an insignificant effect in the behavior of these algorithms.

The same thing happens with the parameter k (number of peers). The number of peers has no effect in the algorithm because the real parameter of interest is the capacity of each peer as shown by the resource models of Section 5.2.

5.4.4 Varying the Number of Categories

In this experiment we analyzed the behavior of CICLB under different numbers of categories. We choose λ to vary from 2 to 10, with the number of clusters $m = 2$

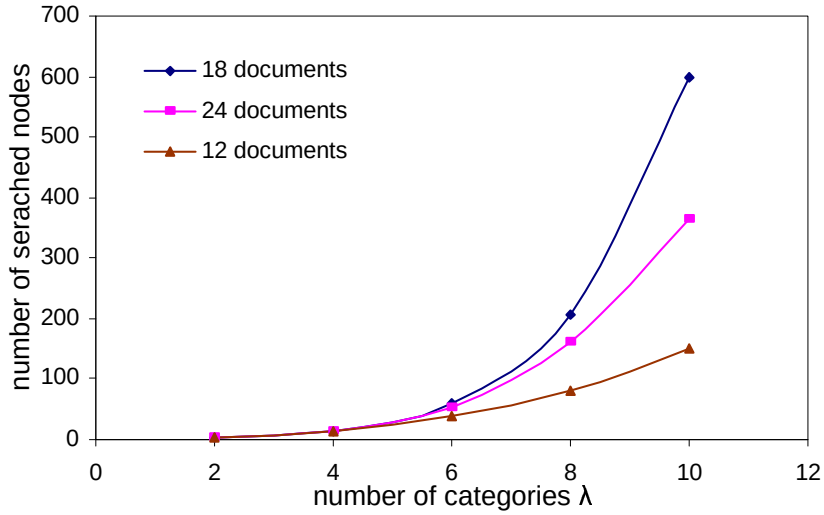

Figure 5.3: Average nodes searched by CICLB against λ for different number of documents

and $n = 12, 18$ or 24. For each value of λ and n, 1000 problem instances were generated. Figure 5.3 plots the number of searched nodes against the number of categories for three different instances of parameter n.

Figure 5.3 shows that the search time is exponential to the number of categories, something expected since the worst case complexity of the algorithm, calculated in Section 5.3.2 has λ in the exponent of the formula.

5.4.5 Varying the Number of Clusters

The number of clusters in which the peers of the system are logically organized is a crucial parameter that is expected to affect the search time needed for finding the optimal allocation for ICLB.

In Figure 5.4 the number of searched nodes by CICLB is plotted against the number of clusters m. Each data point is the average of 1000 problem instances. The system contains $k = 20$ peers that contribute $n = 24$ documents. We generate the popularities of the documents following the Zipf distribution with $\theta = 0.8$ and we consider they are assigned in $\lambda = 15$ categories.

In Figure 5.4 it is obvious that the number of the searched nodes presents an easy-hard transition behavior with respect to the number of clusters. Initially, when the number of clusters is small, the average number of generated nodes is relatively small. When the number of clusters becomes more than 5, then the number of generated nodes becomes greater and stays on average constant ($\simeq 275$ nodes).

The phase transition behavior shown in Figure 5.4 is expected, since the ICLB

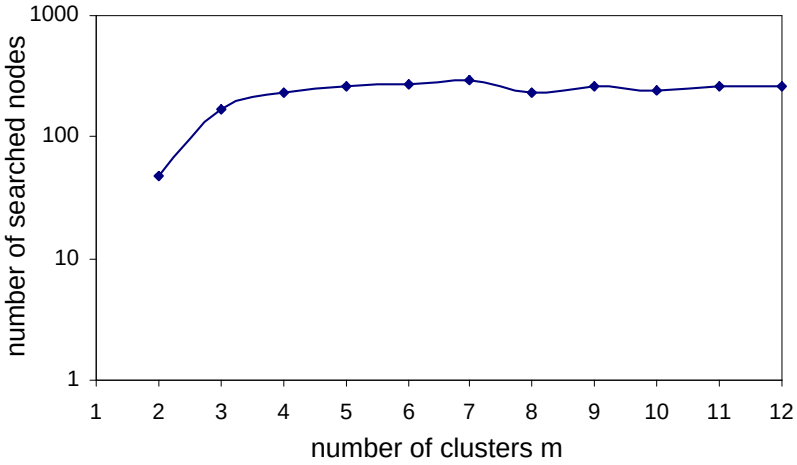

Figure 5.4: Average nodes searched by CICLB against m

problem moves from easy to hard instances. Practically, this means that as the number of clusters m becomes greater, and thus the ratio λ/m moves towards 1, fewer categories are assigned to each cluster and the value of fairness index depends more on the skewness of these category loads than on the partitioning itself. Thus, CICLB must search through a large number of possible partitions before concluding that it has found the optimal solution.

5.4.6 How Fair is MaxFair and CICLB?

MaxFair is a greedy algorithm and thus, does not return the best possible solution. In this section we want to investigate how fair MaxFair is and the way the greedy solution is affected by the various parameters of the system. Additionally, we examined the behavior of the optimal solution under the key parameters.

To examine the difference in the values of fairness index of the returned partitions returned by the two algorithms, we set up an experiment using the values of Table 5.2 for the key parameters. As expected, although MaxFair terminated in a few milliseconds returned an allocation with fairness index 0.990395, whereas CICLB needed 11 hours to return the optimal solution with fairness index 0.999995 i.e., only 0.0096 better than the greedy one. It is therefore clear that MaxFair should be the algorithm of choice in any implementation of CLUSTER.

We additionally conducted several experiments using the baseline values of the parameters shown in Table 5.2, but for a smaller number of categories and clusters. The results of the experiments were averaged to produce the graphs in Figures 5.5 and 5.6. We chose small values for the above parameters as we wanted to determine the solution quality of MaxFair and thus we needed to run

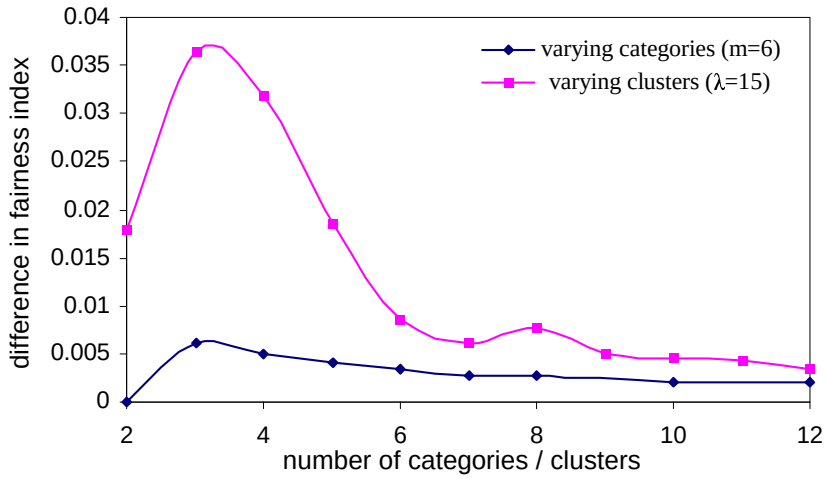

Figure 5.5: Difference in the fairness index between the greedy and the optimal solution

several instances of the same problem.

The graphs in Figure 5.5 show how close is the greedy to the optimal solution for different number of categories, and for different number of clusters. Note that only one parameter is varied each time. We expect the difference between the greedy and the optimal solution to increase or decrease relatively to the difficulty of the problem.

From the presentation of the problem in Section 5.1, we would expect the problem to be difficult when the number of clusters increases as the number of possible allocations increases too. Practically this means that the complete algorithm has more choices to identify a subtree, that the greedy approach would discard by considering it a bad choice at the time. However, the value of the fairness index depends also on the value of the fraction λ/m. The closer this quantity is to 1, the fewer categories are allocated in each cluster. Thus, having only a few categories assigned to each cluster the fairness index depends more on the skewness of these category loads than on the partitioning itself, which leads us to the conclusion that MaxFair does the best available choices for the situation moving closer to the optimal solution. This explains the dropping difference between the two algorithms shown in Figure 5.5 (notice for example that for the last point in the graph the algorithms have to partition 15 categories to 12 clusters).

Furthermore, the allocation problem is expected to be more easy when the number of categories increases as it becomes easier to generate less skewed allocations. In other words, when the number of categories increases, then (using our favorite analogy with NUMP) we have more numbers to partition, which means that they have a better chance of being partitioned more uniformly. On the other

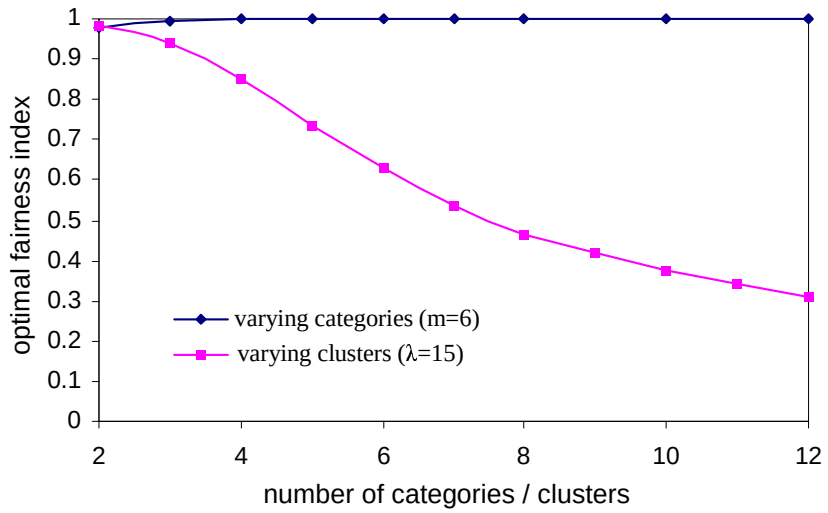

Figure 5.6: Optimal fairness index

hand, when the number of categories is relatively small, the choices of MaxFair seem to be the best thing to do, and CICLB does not have the chance to find something better simply because nothing better exists! To the above we have also to add that the distribution of the category loads does also matter, since a Zipfian distribution is less "flexible" in being partitioned (some "heavy" categories go to different clusters, and all the categories with smaller loads accumulate in the rest of the clusters), thus favoring MaxFair -that does nothing but the obvious- to eventually discover the optimal allocation. Figure 5.5 verifies our hypothesis.

Additionally, as we can observe in Figure 5.6, the optimal value of the fairness index increases with the number of categories and decreases with the number of clusters. This can be explained as follows. As the number of categories increases, the possibility of having more uniform allocation vectors increases too. Conversely, when the number of clusters increases, it is more difficult to be fair with all clusters resulting in more skewed allocation vectors.

Finally, let us point out that in all cases the greedy and the optimal solutions are very close (the difference is always less than 0.0095). This last observation was expected by the experiments and the high values of fairness index achieved by MaxFair in [TXKN03].

5.5 Conclusions

In this chapter, we discussed the problem of fair resource allocation in more sophisticated environments than the ones of Section 3. In particular, we used more realistic models of P2P system resources proposed originally in the context

of CLUSTER [TXKN03]. CLUSTER imposes a hierarchical system structure based on the concept of peer clusters and takes into account the heterogeneity of peers by using sophisticated models of resources allowing for peers with varying capabilities.

We utilized a complete search algorithm, called CICLB, similar to algorithm CGFLA developed in Chapter 4 to study a particular kind of load balancing problem emphasized in the context of CLUSTER. Initially, we specified all parameters affecting the performance of the system, such as the number of peers in the system, the number of documents contributed by the peers, the number of semantic categories to which documents are assigned and the number of clusters to which the peers will eventually belong. We then studied the phase transition behavior of CICLB and compared it experimentally with the greedy algorithm, called MaxFair, presented in [TXKN03]. The greedy algorithm performs very well and returns almost perfectly fair allocations. Finally, we presented a discussion concerning the way the optimal values of the fairness index are affected by the two basic parameters of our system: the number of categories and the number of clusters.

Chapter 6

Concluding Remarks

Let us now summarize the main contributions of this dissertation. In this work we dealt with the problem of fair resource allocation in P2P systems.

Initially, we surveyed the area of P2P computing and paid specific attention in presenting models of P2P networks, measurements concerning the load and the traffic of the network, and the use of ideas from economics and game theory to solve resource allocation problems. This allows the reader of the thesis to see our work in an appropriate context.

We focused on fairness measures to express the notion of fairness in P2P systems. We presented quantitative and qualitative fairness metrics (fairness index [JCH84], max-min fairness [KRT01] and majorization [MO79]) and relate them in a general context.

We considered a very simple model of resource allocation. We defined formally the fair load allocation (FLA) problem in this model and related it to the number partitioning problem (NUMP), one of the six basic NP-complete problems, and the only one using numbers. We presented some of the important algorithms solving NUMP, trying to identify whether we can use these algorithms to cope with FLA.

Then, we proposed two algorithms for solving the FLA problem. One greedy, called GFLA, and one complete algorithm, called CGFLA. We experimentally evaluated these algorithms and compared them in terms of their solution quality, that is the number of generated nodes and the value of the fairness index achieved. For the precise and quantitative comparison of the algorithms, we identified the phase transition behavior of FLA.

Finally, we extended our model of resources for P2P systems to the more sophisticated one originally proposed in [TXKN03] in the context of the P2P content-sharing system CLUSTER that imposes a logical system organization and takes into account the heterogeneity of the peers. We developed a complete anytime algorithm for inter-cluster load balancing in CLUSTER and compared it in great detail with the greedy algorithm MaxFair proposed in [TXKN03].

6.1 Future Work

There are many problems that need to be addressed in future work. We are already in the process of designing distributed algorithms for balancing the load among the peers in a P2P system without any centralized control.

In this case we urgently need good mathematical models for performance analysis in P2P networks and for modelling their evolution and dynamics. Additionally, we can consider a P2P system as a complex adaptive system (CAS), and thus, work with ideas and models from various areas, such as biology, social science and economics.

Bibliography

[ABCdO96] Virgílio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. In *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems (PDIS)*, Miami Beach, Florida, 1996.

[ABJ01] Fred S. Annexstein, Kenneth A. Berman, and Mihajlo A. Jovanovic. Latency Effects on Reachability in Large-scale Peer-to-Peer Networks. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 84–92, 2001.

[AH99] Lada A. Adamic and Bernardo A. Huberman. Scaling Behaviour on the World Wide Web, 1999. Technical comment on [BA99].

[AH00] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. *First Monday*, October 2000.

[AJM01] Yuan An, Jeannett Janssen, and Evangelos Milios. Characterizing and Mining the Citation Graph of the Computer Science Literature. Technical Report CS-2001-02, Faculty of Computer Science, Dalhousie University, 2001.

[BA99] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509–511, October 1999.

[BCF+99] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *INFOCOM*, volume 1, pages 126–134, 1999.

[BCM03] John Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple Load Balancing for Distributed Hash Tables. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, pages 31–35, February 2003.

[BG87] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1987.

[BKM+00] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *Proceedings of the 9th International Conference on the WWW (WWW9)*, volume 33 of *Computer Networks and ISDN Systems*, pages 309–320, 2000.

[BM99] Regina Berretta and Pablo Moscato. The number partitioning problem: An open challenge for evolutionary computation? In David W. Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*. McGraw Hill, 1999.

[BMM02] Ozalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Proceedings of IEEE International Conference on Distributed Computer Systems*, pages 15–22, 2002.

[Bol85] B. Bollobas. *Random Graphs*. Academic Press, 1985.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the 7th International Conference on the WWW (WWW7)*, volume 30 of *Computer Networks and ISDN Systems*, pages 107–117, 1998.

[CDS01] C. Courcoubetis, M. Dramitinos, and G. D. Stamoulis. An auction mechanism for bandwidth allocation over paths. In *Proceedings of the Seventeenth International Teletraffic Congress (ITC)*, May 2001.

[CEbAH00] R. Cohen, K. Erez, D. ben Abraham, and S. Havlin. Resilience of internet to random breakdowns. *Physical Review Letters*, 85, November 2000.

[CGM02a] B. Cooper and H. Garcia-Molina. Peer-to-Peer Data Trading to Preserve Information. *ACM Transactions on Information Systems*, 20(2), April 2002.

[CGM02b] B. Cooper and H. Garcia-Molina. Peer-to-Peer Data Trading to Preserve Information. In *Proceedings of the 22nd IEEE International Conference on Distributed Computer Systems*, July 2–5 2002.

[CHM+02] Ian Clarke, Theodore W. Hong, Scott G. Miller, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, January 2002.

[CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the *Really* Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91*, pages 331–337, Sidney, Australia, 1991.

[Dal20] H. Dalton. The measurement of the inequality of incomes. *Economics Journal*, 30:348–361, 1920.

[DGMY03] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data sharing peer-to-peer systems. In *Proceedings of the 9th International Conference on Database Theory (ICDT 2003)*, volume 2572 of *Lecture Notes in Computer Science*, pages 1–15. Springer, January 2003.

[DKM+02] Stephen Dill, Ravi Kumar, Kevin S. McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. *ACM Transactions on Internet Technology (TOIT)*, 2(3):205–223, 2002.

[Fer89] D.F. Ferguson. *The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms*. PhD thesis, Computer Science Dept., Columbia University, 1989.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proceedings of ACM Special Interest Group on Data Communications*, pages 251–262, 1999.

[FLGC02] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization of the web and identification of communities. *IEEE Computer*, 35(3):66–71, 2002.

[FNSY95] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic models for allocating resources in computer systems. In S. Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, September 1995.

[Fra00] P. Francis. Yoid: Extending the internet multicast architecture. Technical report, AT& T Center for Internet Research at ICSI (ACIRI), April 2000.

[GCL02] D. Grosu, A. T. Chronopoulos, and M.Y. Leung. Load Balancing in Distributed Systems: An Approach Using Cooperative Games. In *Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002)*, April 2002.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. Mathematical Sciences. W.H.Freeman and Company, New York, 1979.

[GLBM01] Philippe Golle, Kevin Leyton-Brown, and Ilya Mironov. Incentives for sharing in peer-to-peer networks. In *Proceedings of 3rd ACM Conference on Electronic Commerce*, October 2001.

[GMP01] A. Goel, A. Meyerson, and S. A. Plotkin. Approximate majorization and fair online load balancing. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01)*, pages 384–390, New York, January 2001. ACM Press.

[GMPW96] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, and Toby Walsh. The constrainedness of search. In *AAAI/IAAI*, volume 1, pages 246–252, 1996.

[GSE+94] Jim Gray, Prakash Sundaresan, Susanne Englert, Ken Baclawski, and Peter J. Weinberger. Quickly generating billion-record synthetic databases. In *Proceedings of the ACM SIGMOD*, pages 243–252, 1994.

[GW95] Ian P. Gent and Toby Walsh. The number partition phase transition. Technical Report 95-185, Department of Computer Science, University of Strathclyde, 1995.

[GW96] Ian P. Gent and Toby Walsh. Phase transitions and annealed theories: Number partitioning as a case study. In *Proceedings of ECAI-96*, pages 170–174, 1996.

[GW98] Ian P. Gent and Toby Walsh. Analysis of heuristic for number partitioning. *Computational Intelligence*, 14(3):430–451, August 1998.

[HA99] Bernardo A. Huberman and Lada A. Adamic. Growth dynamics of the web. *Nature*, 401:131, September 1999.

[HLP52] G.H. Hardy, J.E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, London and New York, 1st ed., 2nd ed. edition, 1934,1952.

[HPS01] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Proceedings of 3rd ACM Conference on Electronic Commerce*, October 2001.

[HWBM02] Cefn Hoile, Fang Wang, Erwin Bonsma, and Paul Marrow. Core specification and experiments in DIET: a decentralised ecosystem-inspired mobile agent system. In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*, pages 623–630, July 15–19 2002.

[JAMS91] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, May-June 1991.

[JCH84] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Have. A quantitive measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report DEC-TR-301, Digital Institution Corporation, Hudson, MA 01749, September 26 1984.

[Jov01] Mihajlo A. Jovanovic. Modeling Large-scale Peer-to-Peer Networks and a Case Study of Gnutella. Master of Science, University of Cincinnati, June 2001.

[KK82] N. Karmarkar and N.R. Karp. The differencing method of set partitioning. Technical Report UCB/CSD 82/113, Computer Science Division, University of California, Berkley, California, U.S.A., 1982.

[Kle00] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 2000.

[KLL+97] D. Karger, E. Lehman, F. Leighton, M. Levine, D. Lewin, and R. Panigray. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 654–663, El Paso, May 1997.

[Knu73] Donald E. Knuth. *The Art of Computer Programming*, volume 3 of *Computer Science and Information Processing*. Addison-Wesley, Massachusetts, 1973.

[Kor95a] Richard E. Korf. From approximate to optimal solutions: A case study of number partitioning. In Morgan Kaufmann, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 266–272, Montréal, Québec, Canada, August 20-25 1995.

[Kor95b] Richard E. Korf. Optimal number partitioning. Technical Report 950062, University of California, Los Angeles, Computer Science Department, December 31 1995.

[Kor98] Richard E. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 106:181–203, August 1998.

[KRR+00] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, and Eli Upfal. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*, pages 57–65, November 2000.

[KRRT99] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for cyber communities. In *Proceedings of the 8th International Conference on the WWW (WWW8)*, volume 31 of *Computer Networks and ISDN Systems*, pages 1481–1493, 1999.

[KRRT02] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. The Web and Social Networks. *IEEE Computer*, 35(11):32–36, 2002.

[KRT01] J. M. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *Journal of Computer and System Sciences*, 63(1):2–20, 2001.

[Lew98] D. Lewin. Consistent hashing and random trees: Algorithms for cashing in distributed networks. Master's thesis, Department of EECS, MIT, 1998.

[Lor05] M.O. Lorenz. Methods of measuring concentrations of wealth. *Journal of the American Statistical Association*, 9:209–219, 1905.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Mar02] Evangelos P. Markatos. Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella. In *Proceedings of the CCGrid 2002: the 2nd IEEE International Symposium on Cluster Computing and the Grid*, pages 65–74, May 2002.

[Mer98] Stephan Mertens. Phase transition in the number partitioning problem. *Physical Review Letters*, 81(20):4281–4284, November 1998.

[Mil67] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[MKL+01] D. S. Milojisic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical Report HP-2002-57, HP Laboratories, Palo Alto, 2001.

[MO79] A.W. Marshall and I. Olkin. *Inequalities: theory of majorization and its applications*, volume 143 of *Mathematics in Science and Engineering*. Academic Press, 1979.

[MRS01] M. Mitzenmacher, A. Richa, and R. Sitaraman. *The Power of Two Choices: A Survey of Techniques and Results*, pages 255–312. Kluwer Academic Publishers, Norwell, MA, 2001.

[NR99] Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC 1999)*, pages 129–140, May 1999.

[NWD03] T.-W. Ngan, D. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, 2003. Forthcoming.

[Pap01] Christos H. Papadimitriou. Game theory and mathematical economics: A theoretical computer scientist's introduction. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 4–8, October 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[RFI02] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. *IEEE Internet Computing*, 6(1):50–57, January-February 2002.

[RHKS01] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level multicast using content-addressable networks. *Lecture Notes in Computer Science*, 2233:14–??, 2001.

[RLS+03] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in structured p2p systems. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[RNMS96] Wheeler Ruml, J. Thomas Ngo, Joe Marks, and Stuart M. Shieber. Easily searched encodings for number partitioning. *Optimization Theory and Applications*, 89(2):251–291, May 1996.

[RS02] Michael Rabinovich and Oliver Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2002.

[San02] T. Sandholm. Algorithm for Optimal Winner Determination in Combinatorial Auctions. *Artificial Intelligence*, 135:1–54, 2002.

[SAR+02] I. Stoica, D. Adkins, S. Ratnasamy, S. Shenker, S. Surana, and S. Zhuang. Internet indirection architecture. In *Proceedings of ACM SIGCOMM'02*, pages 73–86, August 2002.

[SD96] Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.

[SGG02] S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Conferencing and Networking*, January 2002.

[SK97] R. Schwartz and S. Kraus. Bidding mechanisms for data allocation in multi-agent environments. In *Proceedings of ATAL'97*, 1997.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[SMLN+03] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[TXK02] P. Triantafillou, C. Xiruhaki, and M. Koubarakis. Efficient massive sharing of content among peers. In R. Wagner, editor, *Proceedings of the Workshop on Resource Sharing in Massively Distributed Systems (RESH'02) – 2002 IEEE International Conference on Distributed Computer Systems*, pages 681–685, July 2-5 2002.

[TXKN03] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards high-performance peer-to-peer content and resource sharing systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, January 2003.

[Wat99] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 1999.

[WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393:440–442, 1998.

[Xir03] Chrysanni Xiruhaki. Load Balancing In Peer-to-Peer Networks: System Architecture and Algorithms. Master of Computer Engineering, Department of Electronic and Computer Engineering, Technical University of Crete, Chania, Greece, March 2003.

[YGM03] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, March 5–8 2003.

Appendix A

Appendix

Proposition A.1 *Let us assume that we have a P2P system with k peers and vector $\overline{x}$ denotes the current load allocation. Let $c > 0$ be the popularity of a new document that arrives in the system. The highest increase to the fairness index of the system is achieved by allocating this document to one of the peers with the smallest load.*

Proof: The result follows easily. Without loss of generality, we suppose that we can arrange the components of $\overline{x}$ in non-increasing order e.g., $x_1 \geq x_2 \geq \cdots \geq x_k$.

If we allocate the new document to some peer i such that $x_i > x_k$ the load of that peer would change to $x_i + c$, the new allocation vector would be $(x_1, \ldots, x_{i-1}, x_i + c, x_{i+1}, \ldots, x_k)$ and the fairness index would be equal to

$$\frac{\overbrace{(x_1 + ... + x_{i-1} + x_i + c + x_{i+1} + ... + x_k)^2}^{A}}{k \underbrace{[x_1^2 + ... + x_{i-1}^2 + (x_i + c)^2 + x_{i+1}^2 + ... + x_k^2]}_{B}}$$

If we had allocated the document to the peer with the smallest popularity (peer k), then the allocation vector would have been $(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_k + c)$, and the value of the fairness index would have been equal to

$$\frac{A}{k \underbrace{[x_1^2 + ... + x_{i-1}^2 + x_i^2 + x_{i+1}^2 + ... + (x_k + c)^2]}_{C}}$$

Let us now consider the difference

$$\frac{A}{kB} - \frac{A}{kC} = \frac{A}{kBC}(C - B).$$

Since quantities k, A, B, C are positive, it is sufficient to consider only the difference

$$C - B =$$

$$x_1^2+...+x_{i-1}^2+x_i^2+x_{i+1}^2+...+(x_k+c)^2-[x_1^2+...+x_{i-1}^2+(x_i+c)^2+x_{i+1}^2+...+x_k^2] =$$
$$2cx_k - 2cx_i = 2c(x_k - x_i)$$

which is easily seen to be negative since $x_k < x_i$. Thus, we have the greatest increase to the fairness index if we allocate the new document to the peer with the smallest load. ■

Proposition A.2 *By never placing a document in more than one peer with current load 0, we cut the search space of algorithm CGFLA and generate only $k^n/k!$ possible allocations.*

Proof: If we were placing each document to each one of the peers with current load 0, taking eventually all possible combinations of all the n documents to the k peers of the system, we would have k^n possible allocations. However, we allocate each document only to one peer with current load 0 at each level of the search tree.

At the first level, we place the first document only to one peer with current load 0, having only one sub-allocation instead of k. Thus, we cut the above mentioned search space by $1/k$ and reduce the possible allocations to k^n/k.

At the second level, there are $k-1$ peers with current load 0, but we choose only one to place the next document, cutting the above computed search space by $1/(k-1)$, and the possible allocations are now $k^n/[k \cdot (k-1)]$.

...

At level $(k-1)$, there are 2 peers with current load 0, and we choose one of them to place the next document, cutting the search space in half, and thus we generate $k^n/[k \cdot (k-1) \cdot .... \cdot 3 \cdot 2]$ possible allocations.

¿From level k onwards, and until we allocate all the remaining documents at level n, we place each document to each one of the k peers in turn, having no further reduction to our search space.

Thus, by never placing the document in more than one peers with current load 0 we generate $k_n/k!$ possible allocations. ■

Proposition A.3 *Let us consider an instance of FLA with parameters k and D. Let q be the quotient and ρ be the remainder of the division of $\sum_{d \in D} p(d)$ with k. Then the maximum value of the fairness index that we can achieve is the following:*

$$\frac{[(k-\rho)q + \rho(q+1)]^2}{k[(k-\rho)q^2 + \rho(q+1)^2]}$$

Proof: The best we could do is to allocate load q to each of the $k-\rho$ peers, and allocate load $q+1$ to each of the remaining ρ peers. In this case, the allocation

vector is $\overline{x} = \{\underbrace{q+1, q+1, \ldots, q+1}_{\rho}, \underbrace{q, q, \ldots, q}_{k-\rho}\}$ and the fairness index value is

$$\frac{[\overbrace{q+q+\ldots+q}^{k-\rho}+\overbrace{(q+1)+\ldots+(q+1)}^{\rho}]^2}{k[\underbrace{q^2+q^2+\ldots+q^2}_{k-\rho}+\underbrace{(q+1)^2+\ldots+(q+1)^2}_{\rho}]} =$$

$$\frac{[(k-\rho)q+\rho(q+1)]^2}{k[(k-\rho)q^2+\rho(q+1)^2]}$$

■