Towards High Performance Peer-to-Peer Content and Resource Sharing Systems

Peter Triantafillou
Dept. of Computer Engineering and Informatics, University of Patras, Greece
peter@ceid.upatras.gr

Chryssani Xiruhaki
Dept. of Electronic and Computer Engineering, Technical University of Crete, Chania Crete, Greece
xiruhaki@intelligence.tuc.gr

Manolis Koubarakis
Dept. of Electronic and Computer Engineering, Technical University of Crete, Chania Crete, Greece
manolis@intelligence.tuc.gr

Nikolaos Ntarmos
Dept. of Computer Engineering and Informatics, University of Patras, Greece
darmnik@softnet.tuc.gr

Abstract

Peer-to-peer sharing systems are becoming increasingly popular and an exciting new class of innovative, internet-based data management systems. In these systems, users contribute their own resources (processing units and storage devices) and content (i.e., documents) to the P2P community. We focus on the management of content and resources in such systems. Our goal is to harness all available resources in the P2P network so that the users can access all available content efficiently. Efficiency is taken both from (i) the point of view of the system, in that we strive to ensure fair load distribution among all peer nodes, and (ii) from the point of view of the users, in that we strive to ensure low user-request response times.

We propose a novel architecture for this new class of applications, which differs drastically from what is either found currently in existing products or proposed in academia. We contribute and study novel solutions that achieve our goals, while at the same time addressing the formidable challenges due to the autonomy of peers, their heterogeneous processing and storage capacities, their different content contributions, the huge system scale, and the highly dynamic system environment.

1. Introduction

The client-server model has been the dominant model for constructing distributed systems and services and the simplicity of its concept has played a key role in the successful commercial deployment of distributed computing for more than a decade. The emergence of internet computing and applications, however, have made prominent some of the model's inherent weaknesses: the requirement of central control of information and processing at specialized computing nodes (i.e., the servers) is too stringent and limiting for a variety of rapidly emerging internet computing application classes. Internet content sharing systems are an example of systems supporting such an application class, which has become very popular with systems like Napster [20], Gnutella [13], KaZaa [18], Audio Galaxy [4], etc.

In general, content sharing systems consist of a (potentially very large) number of computing nodes, offering (computing and storage) resources and content ("documents") to the community. Thus, nodes belonging to users may contribute content to the rest of the community and, in addition, may permit the use of their own resources to store content contributed by others and allow access to it from other community members. Given these characteristics, the central control of information at special nodes is undesirable in order to avoid central points of failure and performance bottlenecks, to preserve the anonymity of users accessing content and services, and to fully utilize the available resources contributed by *all* member nodes. As a result, the *peer-to-peer* (P2P) paradigm for architecting distributed systems is recently becoming increasingly popular. P2P systems consist of a set of peers, which are nodes of equal stature, which have autonomy, and which can collaborate with each other, pulling together their resources, in order to either obtain services or jointly tackle large computing jobs.

1.1 Problem Definition

Our Goals

Our goal with this research is to ensure the high performance of the system through the proper exploitation of all available resources and the efficient management of content. Efficiency is interpreted both (i) from the system's point of view and will be ensured through the

fair load distribution across all peers in the system, and (ii) from a user's point of view, by facilitating short response times to user requests. Fair load distribution in our setting implies that we ensure load balancing, while taking into account the heterogeneity of the contributed resources by the different peers.

In addition to the obvious high usefulness of achieving these goals, it should be stressed that achieving fair load distribution is of fundamental importance in a P2P system, given the equal stature of the nodes, and that low response times is one of the main promises inherent in the peer-to-peer paradigm.

The Challenges

Despite the appropriateness of the P2P paradigm, ensuring the efficiency in P2P content sharing systems is a formidable task. The associated challenges stem from the desirable use of the P2P paradigm and from the application's features. Having autonomous nodes of equal stature introduces the need of complex distributed coordination algorithms. Furthermore, the system should be expected to scale to hundreds of thousands of nodes and millions of documents. In addition, different nodes make different content contributions (e.g., from no contribution at all, to contributing several documents) and offer/possess heterogeneous processing and storage capacities. Finally, the system may operate in a highly dynamic environment in which the popularity of the stored content varies with time, nodes enter and leave the system at their free will, and content can be added and deleted at any time. These facts significantly raise the level of complexity involved in the goal of managing content and all available resources so as to provide efficient access to the stored content.

1.2 An Overview of our Approach

We assume that the content in the system has (an initially static and) known popularity[1], with document popularities following the Zipf distribution, as is the case for Web objects [19, 31] and existing popular P2P systems [17]. The key elements of the proposed architecture are that:

1. We form groups of documents. The document groups can be defined for example using hashing functions on document ids, or they can be defined using more elaborate approaches: for example, assuming that the contributed documents are accompanied by keywords characterizing their semantic content, we can utilize tools (e.g., [27, 5, 32]) that can classify the documents, given their keywords, into semantic categories. (Hereafter, for simplicity we will describe our architecture using document semantic categories. However, it should be clear that any other document grouping is equally acceptable).
2. We form clusters of peers (nodes), based on the semantic categories of the documents contributed by the peers.
3. Given this, our load balancing task is then viewed as a two level problem: First, we ensure load balancing across the peer clusters (inter-cluster load balancing) through the appropriate assignment of the document categories to the clusters; second, we ensure the load balancing among the peers that belong in the same cluster (intra-cluster load balancing), for all clusters.
4. Finally, our intra-cluster load balancing and response time goals are achieved through the maintenance and exploitation of metadata describing the association between peer clusters with document categories, and/or the use of routing indices, as proposed in the literature.

The maximum number of clusters in the system is tunable and predefined during the bootstrap of the system, while the number of document categories may expand at will. This means that a document category exists only when there are published documents associated with it. On the other hand, since the maximum number of clusters is constant during the lifetime of the system, we can assume that all clusters are created empty during the bootstrap phase and are dynamically populated with document categories over time. We should note here that peer clusters partition the domain of document categories, so that every category belongs to only one cluster. However, it is possible for a cluster to exist, independently of its association to document categories (this could happen, for example, when the number of published categories is smaller than the maximum number of clusters).

The Contributions

To our knowledge this is the first work that addresses the problem of harnessing all available resources contributed by the peers so to ensure efficiency of operation in this exciting new application area. Our main contributions are:

1. A radically different proposal for architecting P2P systems. Our proposed architecture imposes a logical system structure based on the concepts of document categories, node clustering, and their associations.
2. A formal description of the problem of inter-cluster load balancing and its solution. We utilize the *fairness index* of [25] as a novel metric for load balancing in our context. We show that a greedy algorithm achieves very good to excellent inter-cluster load balancing across a wide range of scenarios. We also present additional mechanisms that can facilitate short response times and intra-cluster load balancing.
3. A study of the robustness of our solution to the dynamics of peers and content, and

[1] Initial popularities of documents can fairly easily be estimated in most applications. For example, music file popularities follow their position in popular charts; the popularities of book files in library applications can be estimated using check-out information at conventional libraries; popularities of video files can be estimated by information at video rental stores, etc.

4. The mechanism consisting of the additional architecture, algorithms, and protocols, which accommodate the dynamics of peers and content, maintaining the system's efficiency on the fly.

We offer evidence that show that our solution achieves our performance goals and is robust with respect to different distributions of the popularity of document categories, varying skew of access to documents, different scales (with respect to the number of documents, the number of nodes, the number of categories, and to the number of clusters), differing content contributions by nodes, differing node processing and storage capacities, and with respect to the dynamics of the environment.

We hold the view that internet sharing systems are an exciting application class, indicative of the next wave of applications and the associated difficulties, which the data management community will be called to face. Also, the P2P paradigm is rapidly emerging as the paradigm of choice for such applications. And, based on the difficulties mentioned above, P2P content sharing systems pose a number of difficult challenges to our community, especially if high performance is sought.

The organization of this paper is as follows. Section 2 presents some related work. In Section 3 we introduce the architecture of our P2P system and explain how queries are processed. In Section 4 we introduce the problem of inter-cluster load balancing in a P2P system and propose a solution for it. In Section 5 we examine the robustness of our solution to the load balancing problem under varying changes to the system. In Section 6 we present the additional mechanisms necessary for our system so that it adapts to a dynamically changing environment. Finally, in Section 7 we conclude our paper.

2. Related Work

The problem of fair load distribution for P2P sharing systems has not been addressed extensively by related research and is an open problem.

Recent work in P2P systems, like the overlay networks that have been designed to facilitate the development of P2P applications, (e.g., Tapestry [8], Pastry [3], CAN [28], and Chord [15]), focus only on the problems of query routing and object location to guarantee that if there exist results relevant to a query, these results will be returned. In these systems load balancing is addressed in a rather naive way simply by resorting to the uniformity of the hash function utilized.

The work in [29] is an exception to the above rule, addressing the problem of load balancing in order to face "flash crowds" in a static system architecture and assumes global knowledge. Our work aims to solve a much more general problem, as stated above.

With respect to response times, typical systems such as Freenet [14] and Gnutella [13] might face serious difficulties when it comes to ensuring low response times, since requests are passed from peer to peer, until either one is found that stores the desired document(s), or a user-determined "number-of-hops" count is reached and the system gives up. Our architecture will not burden the user with such difficult decisions and will ensure a response time within only a few hops for the common case and an upper bound on the number of hops for the worst case.

The problem of the efficient search in a P2P network is also addressed in [1] by introducing the concept of routing indices, which allow nodes to forward queries to their neighbors that are more likely to have answers. Also [7] studies the same problem and demonstrates that some simple search algorithms from AI can offer big benefits when compared with the strategies of Gnutella and Freenet. The search protocols of [1,7] can be applied to our architecture, as well. In [6] an analysis of "hybrid" P2P systems (i.e., P2P systems where some sort of centralized control still exists) is presented. [6] develops and validates an analytical model and uses it to compare various hybrid P2P architectures.

Other related work in P2P systems is the system Gridella [17] and the project Piazza [30]. Gridella is a system that goes beyond the well-known Gnutella system and provides efficient and robust search by relying on a sophisticated data structure called P-Grid [17]. Piazza is an ambitious project at the University of Washington with the goal of using successful ideas from database technology in the field of P2P computing [30].

Another strand of project DIET, under which this work is carried out, studies P2P systems from the point of view of autonomous agents built using a bottom-up and ecosystem-inspired approach [23, 9]. SWAN is a recent P2P lookup system implemented using the agent platform developed in DIET [12]. The system Anthill, presented in [22], also attempts to build P2P systems using ideas from nature-inspired computing (in particular, ant colonies [11]). So far a file-sharing and a load balancing application have been built using Anthill [22, 2].

3. System Architecture

The basic (technological and "philosophical") principle of our approach that makes it radically different compared to other approaches towards a P2P system architecture is that, in order to achieve high performance in such a system, we need to impose a *logical system structure*, which can facilitate the achievement of our performance goals. This structure we propose consists of the peer clusters, the document categories, and their associations. The challenge is to ensure that this structure respects the autonomy of peers, the heterogeneity of their characteristics, and that the overall solution successfully addresses the difficulties presented earlier and facilitates the achievement of our performance goals.

A fact that complicates our task is that there is no consensus in our community to resolve the debate for the most appropriate overall architecture for P2P systems. Some favor *pure peer-to-peer*, as opposed to *hybrid peer-*

to-peer architectures in which typically there exists a set of *super peers,* which are organized in their own P2P network and are burdened with additional chores, such as maintaining metadata and key knowledge for the proper system functioning [6]. Where this dilemma arises in the presentation of our architecture, we will be discussing solutions that fit both environments.

We first present the architectural organization of our system, assuming a static system, for reasons of simplicity. In Section 6 we deal with the dynamic behavior of the system.

3.1 Peer Clustering and Document Categories

The nodes of the system will be *logically* organized into a set *C* of clusters. All nodes belonging to the same cluster will be able to either serve all the retrieval requests for documents contributed by all the nodes of that cluster (for example, in the case the nodes can store all documents), or find another node that can. The latter can be achieved by having each node or a distinct set of super peer nodes, storing cluster metadata, describing which documents are stored by which cluster nodes. In the following we will assume this latter design choice and discuss the type of metadata needed and its use. Alternatively, if pure P2P solutions are favored, the same goal can be achieved using routing indices at the cluster's nodes, routing requests for documents/categories to the proper cluster node(s) – for routing indices and their use for a similar goal refer to [1].

In the proposed architecture, 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 node may belong to *more than one* cluster if it contributes documents associated with more than one category.

3.2 Metadata description

Each node maintains data structures in order to be able to communicate with other nodes and exchange information or find other nodes that hold desired information. Each node keeps the following metadata structures, shown in Figure 1:

- The Document Table (DT), mapping ids of documents stored locally to document categories.
- The Document Category Routing Table (DCRT), mapping each document category to a cluster-id, where cluster-ids are pseudorandom numbers computed during the bootstrapping of the system.
- The Node Routing Table (NRT), mapping each cluster-id to a list of node-ids (pseudorandom numbers computed during the bootstrap of each node) of nodes belonging to the cluster. (The construction of the NRT will be described later in Section 6.3).

DT

DocumentID	Semantic Category
21524	"Heavy Metal"
21635	"Hard Rock"
61376	"Hard Rock"
52447	"Heavy Metal"
62473	"Pop"
27447	"Hard Rock"
32563	"Pop"
…	…

DCRT

Semantic Category	ClusterID
"Pop"	000000
"Classic Rock"	000001
"Hard Rock"	000002
"Heavy Metal"	000003
"Folk"	000000
"Ambient"	000004
"Electronica"	000002
…	…

NRT

ClusterID	NodeIDs				
000000	1992945	9274957	2876299	9344899	8119877
000001	6236324	7834927	9817987	6965193	
000003	2623456	5696234	3297134		
000005	2384729	3164698	6498987	3321747	

Figure 1: The data structures of each node

For the moment, we assume that each document belongs to a single category. We will eliminate this assumption later.

3.3 Query Processing

Let us now give an example of how queries are processed in our system.

User queries Q are submitted to the system through the users' nodes and are of the form $[(k_1, k_2 \ldots k_n), m, id_Q]$, where k_i are user-supplied keywords, m is the number of desired results, defaulting to and bounded by a system-wide value (e.g., 50 results), and id_Q is a pseudorandom number, uniquely identifying each query (e.g., a secure hash of k_i, s_i, and m). Query processing is a two step procedure:

1. The requesting node does the following:
   a. It maps the keywords $(k_1, k_2 \ldots k_n)$ to one or more semantic categories, s_i, using appropriate categorization tools (e.g, [27, 5, 32]) (see section 1.2, item (1)).
   b. Through its DCRT it finds the clusters of nodes with the semantic categories, s_i.
   c. Chooses a random node n_i from each associated cluster, using its NRT, and sends to it the query. If no live node exists, the query will fail. The random selection of nodes can ensure that cluster nodes get an equal share of the workload targeting their cluster.
2. The target node n_i does the following:
   a. It matches the categories of the query against the semantic categories of its documents and finds a number a of resulting documents matching the query.
   b. If the number a of resulting documents is less than m, n_i forwards the query to all of its known neighbors in the cluster, decreasing m by a. This will be recursively repeated until the desired number of document(s) is found or all reachable nodes of the cluster have been queried. Loops in

query forwarding can be easily detected and broken using $id_{Q'}$.

c. The final result set R is returned to the requesting node by node n_i.

We assume that the metadata data structures are up to date. How this is accomplished is shown later.

Note that, as a result of our architecture, in the worst case, the response time will be bounded from above by the number of nodes in the larger cluster to participate. Thus, with this approach we can see that both (i) the load is balanced within a cluster and (ii) guarantees can be given to users with respect to worst case response times.

4. Fair Load Distribution

In order to ensure the high performance of the system, we have to avoid bottlenecks and balance the load across *all* system nodes. *Load* in our case is the number of requests *served* by a data store node of the system. This goal can be partly achieved by associating the document categories with clusters of nodes, in a manner that ensures a fair distribution of the document-category popularities to the clusters of nodes. We refer to this property as *inter-cluster load balancing*.

With the term *intra-cluster load balancing* we mean that, for each cluster, all the nodes that belong to it receive on average (approximately) the same number of requests from the total requests that target this cluster. With the term *global load balancing* we mean that the average load of all the nodes that belong to the system must be as uniform as possible. Global load balancing is achieved by independently achieving inter-cluster and intra-cluster load balancing.

In the following we assume that all the nodes of a cluster have enough storage capacity in order to store all the documents of the cluster. In Section 4.3 we eliminate this assumption. Thus, when a query reaches any node of the cluster, it will be answered locally from the queried node. So, if the query is initially forwarded to a randomly chosen node of the cluster, then the load will be balanced among the nodes of the cluster as described in Section 3.3. So, intra-cluster load balancing will be achieved and henceforth we concentrate on the problem of inter-cluster load balancing.

4.1 Formal Problem Formulation

We now formally define the problem of inter-cluster load balancing. In order to gain insight on the complexity of the problem and leverage known solutions, we first make a few simplifying assumptions which lead us to the NP-Completeness result and to known heuristics for the problem. Later, we form our solution by extending known heuristics and drop the simplifying assumptions.

As we already said above, we initially assume that peers have the same processing and storage capacities, and enough storage space to store all documents of the categories to which they contribute. We will do away with all these assumptions in Section 4.3.

We denote with $|A|$, the number of elements of the set A. Our system will store a set D of *sharable documents*. The $|D|$ documents are contributed by a set N of *peers/nodes*. Each node in the system is a user's computer. Nodes can contribute more than one document to the system. The $|D|$ sharable documents of the system belong to a set S of categories. This is captured by a function $f: D \rightarrow S$ that maps each document to one or more document categories. Each document d is associated with a *popularity*, $p(d) \in [0,1]$, which gives us the probability that a user will want to retrieve d. We will denote with $p(s)$ the total popularity of semantic category s. This popularity is equal to the sum of the popularities of the documents it consists of. In other words, $p(s) = \sum_{\{d:\, f(d)=s\}} p(d)$. If a document belongs to more than one semantic category, its popularity is evenly distributed among them. We denote with $D(n)$ and $S(n)$ the set of all the documents and categories that are stored by node n. We will denote with $D_i(n)$ the set of documents whose categories are assigned in cluster c_i and are stored in node n and $S_i(n)$ denotes the subset of $S(n)$ that is assigned to cluster c_i. Each node n has a popularity $p(n) \in [0,1]$ that is equal to the sum of the popularities of its documents. Formally, $p(n) = \sum_{d \in D(n)} p(d)$. Finally, let S_i be the set of all semantic categories belonging to cluster I, and $p(S_i) = \sum_{s \in S_i} p(s)$.

4.2 Complexity Results

We now define the *inter-cluster load balancing problem* (ICLB) formally.

The Decision Problem ICLB

Instance: A set N of nodes and a set D of documents contributed by the nodes in N. Each document $d \in D$ has an associated category $f(d)$ from a set of categories S and a popularity $p(d) \in [0,1]$. For simplicity reasons we assume that each node contributes documents belonging to a single semantic category, that all nodes have the same processing and storage capacity, and that each document may belong to a single semantic category.

Question: Is there a partition of N into k clusters $N_1, N_2, \ldots, N_k$ such that the following two constraints are satisfied?

1. If two documents belong to the same category, then the nodes that contributed/store these documents belong to the same cluster.
2. Clusters have equal normalized popularities. Formally, $\frac{p(S_i)}{|N_i|} = \frac{p(S_j)}{|N_j|}$ for all $1 \le I, j \le k$.

The load-balancing objective formalized by ICLB is to create k clusters of peers/nodes that will have equal total popularity and consequently equal load. But not all clusters are of equal size. So the popularity should be normalized with respect to the number of nodes in the clusters so that the average load faced by each peer/node in the system will be equal. Thus, we want the clusters that will be created to have equal normalized cluster popularities and this is captured by the second constraint above.

Proposition. ICLB is NP-complete.

Sketch of Proof. Membership in NP is straightforward. To prove NP-hardness we use a transformation from the BALANCED PARTITION problem[2]. This problem is a generalization of the PARTITION problem given in [21]. ■

In the inter-cluster load-balancing problem as defined above, the normalized cluster popularities are constrained to be equal to each other. An alternative way of looking at this issue, would be not to equate but rather to *balance* the normalized cluster popularities, $\frac{p(S_i)}{|N_i|}$, among the clusters. This balancing can be performed according to any metric of "fairness" such as the ones compared in [24]. In this paper, "fairness" is measured according to the *fairness index* of [25] which is defined as follows. Let r be a resource to be shared among n individuals and x be a random variable giving the amount allocated to each individual (where x_i gives the amount allocated to individual i). The fairness index of x is then given by the following formula:

$$fairness(x) = \frac{(E(x))^2}{E(x^2)} = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\left(\sum_{i=1}^{n} {x_i}^2\right)}$$

In our problem setting, r stands for the load, n stands for the number of clusters, and x_i for the normalized popularity of cluster c_i.

The value of the fairness index is always between 0 and 1. The closer the value is to 1, the fairer the load distribution becomes (with 1 giving total fairness). Values closer to 0 are less fair. As an example, if the fairness index is 0.20 it means that the load distribution is fair (unfair) for the 20% (80%) of the nodes in the system. The fairness index represents a *global* property of the system and naturally captures the essence of load balancing. In our current work we revisit the issue of fairness using majorization that has been shown to be stricter than other fairness metrics such as the fairness index [24].

[2] The proof is due to Apostolis Dimitromanolakis.

4.3 Accounting for Different Peer Processing and Storage Capacities and Content Contributions

So far, we have assumed that all the peers in the system are equivalent, in that they all have the same processing and storage capacity, that each node contributes documents of a single category, and has enough storage space to store all documents of this category. In the following sections we will do away with these simplifying assumptions one by one and generalize the presented solution.

4.3.1 Peers with Different Processing Power

In the general case the nodes of the system will not have the same processing power. We accommodate this, by modeling each node as having a number of processing capacity units, measured in relation to some reference point (e.g., clock speed, cpu benchmark performance, etc.). So, the computational capacity of each cluster now depends on the number of processing units it contains, which is the sum of the processing units of all its nodes, while still assuming that a node contributes documents belonging to a single semantic category. To calculate the normalized popularity of a cluster, c_i, instead of dividing the total cluster popularity by the number of nodes in c_i, we divide it by the total number of computational units of the cluster, $U_i = \sum_{k \in N_i} u_k$, where u_k is the number of computational units of the node k, and N_i is the set of nodes belonging to cluster i. So, the normalized cluster popularity of c_i is equal to $\frac{p(S_i)}{U_i}$. Obviously, u_k can easily become a more sophisticated formula that would take other resources into account (e.g., bandwidth, etc.).

4.3.2 Peers with Different Content Contributions

In general, a node can contribute a different number of documents, and these documents can belong to more than one category. If the documents contributed by a node belong to different categories, but they are assigned to the same cluster, then there is no other change concerning the normalized popularity of the cluster except of the one presented in Section 4.3.1. On the other hand, when a node contributes documents of categories, which are assigned to different clusters, the node is modeled as belonging simultaneously to all these clusters, contributing different parts of its computational units to the clusters. The computational units of the node that are contributed to a specific cluster are analogous to the popularities of the categories that the node stores. Consider, for example, a node n having u_n processing capacity units and contributing documents belonging to the categories s_1 and s_2, which are assigned to clusters c_1 and c_2 respectively. Remember that so far we are assuming that nodes store *all* documents of the categories assigned to their cluster. Thus, because n belongs to c_1 and

c_2 it will store all documents in the categories assigned in c_1 (in our notation, S_1) and all documents in the categories assigned in c_2 (in our notation, S_2). Thus, this node will contribute $\frac{u_n \cdot p(S_1)}{p(S_1)+p(S_2)}$ of its computational units to the cluster c_1, and $\frac{u_n \cdot p(S_2)}{p(S_1)+p(S_2)}$ of its computational units to the cluster c_2. Thus, the new formula computing the normalized popularity of each cluster is:

$$normalized\ popularity\ of\ cluster\ c_i = \frac{p(S_i)}{\sum_{k \in N_i} \frac{u_k \cdot p(S_i)}{p(S(k))}}$$

where, $S(k)$ denotes the set of all categories which node k stores and N_i are the nodes belonging to cluster i.

4.3.3 Peers with Different Storage Capacities

In general, the system will consist of nodes that will have different storage capacities and thus they may not be able to store all documents of the categories to which they contribute. Our approach to deal with this is as follows.

In this scenario, each node n stores a set of documents, $D(n)$, whose categories may be assigned to different clusters. If, say, some documents of n are assigned to cluster c_1 and some to cluster c_2 then $D_1(n)$ and $D_2(n)$ represent the documents stored by n which are assigned to c_1 and c_2 respectively. Thus, in this case, the normalized cluster popularities are given by:

$$normalized\ popularity\ of\ cluster\ c_i = \frac{p(S_i)}{\sum_{k \in N_i} \frac{u_k \cdot p(D_i(k))}{p(D(k))}}$$

where, $p(D(k))$ and $p(D_i(k))$ represent the popularity of the document sets $D(k)$ and $D_i(k)$.

We naturally expect that each node will be able to store locally at least the documents it contributes. The additional storage capacity that exists in some nodes will be used to store replicas of some of the most popular documents within the cluster.

The astute reader will notice that the achievement of intra-cluster load balancing in this case can no longer be guaranteed by the mechanism presented in Section 3.3, which randomly selects a node, between those storing a replica of the desired document within the cluster, and forwards the request to it. The reason for this is that the different nodes of the cluster can store content whose total popularity differs.

Our solution for the intra-cluster load balancing problem here is based on storing popular document replicas in such a way that ensures that the total content popularities stored by any two nodes of a cluster are (almost) equal.

In this way, the mechanism presented in Section 3.3 continues to suffice to ensure intra-cluster load balancing. In essence this document replica placement strategy determines the values for the sets $D(n)$, and $D_i(n)$ for all nodes n and all clusters c_i in the above formula. Furthermore, this solution must also ensure that the storage space requirements imposed by it onto the nodes must be very small.

The details of the proposed solution is as follows. We expect that there is a number reflecting the desirable number of replicas, *n_reps*, for each document (assume for simplicity that this number is the same for all documents). Then, for each cluster c_i and for every category s, which is stored in this cluster and which contains *n_docs* documents, each with size *size_of_doc*, the total storage space required to store s is equal to:

$$size(s) = n\_docs \times n\_reps \times size\_of\_doc$$

We divide this storage size into $|N_i|$ pieces, one per each node of this cluster. Thus, the popularity of a node k $p(k)$ depends on the documents k stores. If the popularity distribution of the documents in s is uniform, then for any two nodes k_1, k_2, we have $p(k_1) = p(k_2)$ and we are done.

If the document popularity distribution in s is skewed, we select the m most popular documents in the category, whose total popularity covers a significant percentage of the document probability mass (e.g., 35%). Subsequently, we place one replica of these m documents in all nodes of the cluster. In our experiments we have found that less than 10% of all documents typically total more than 35% of the document probability mass for practically all realistic different Zipf distributions. Thus, the end result of this policy is that the expected load to be received by the nodes of this cluster is effectively balanced between the cluster nodes.

The following example illustrates the storage requirements this approach puts on the nodes.

Example. Consider a system with 2,000,000 documents, 200,000 nodes, 2,000 categories and 500 clusters, *n_docs* = 1,000, *n_reps* = 5, and which stores 3-minute MP3 documents with *size_of_doc* = 4MB. Then for every category s, *size(s)* = 20GB. Assuming each cluster consists of 200 nodes, i.e., $|N_i|$=200, each node in the cluster receives originally 100 MB of data. The requirement to store the m = 100 (10%) most popular documents costs an additional 400 MB for each node, bringing the total storage requirement to 500 MB per node, per category stored in the cluster. Since on average 4 categories are placed in a cluster, the total required storage space amounts to 2GB, a very small percentage of current disk space. ■

4.4 The MaxFair Algorithm for Inter-cluster Load Balancing and its Performance

We have developed a greedy algorithm, called MaxFair, for inter-cluster load balancing based on maximizing the fairness index as defined in section 4.2. MaxFair works as follows. It considers each category in turn and assigns it to a cluster of nodes (O(|S|) time). The criterion for this assignment is the "maximum fairness" among the normalized cluster popularities, as they emerge after this

assignment. When a new category must be assigned to one of the clusters, all the possible assignments are tested (O(|C|) time) and finally it is assigned to the cluster yielding the maximum fairness for the normalized cluster popularities (O(|C|) time), between all possible assignments. The MaxFair algorithm is not executed in a distributed manner. MaxFair is incomplete but its worst-case time complexity is O(|S|$\times |C|^2$) where $|S|$ is the number of categories and $|C|$ is the number of clusters. Additionally, it performs very well for a variety of scenarios as we show below.

Performance Study Setup

In all the experiments we performed (which refer to the most general scenario of Section 4.3.3) the popularity distribution of documents follows the Zipf distribution with the Zipf parameter value equal to 0.8. (We note that the Zipf parameter values, which have been found to adequately capture the distributions of accesses for web objects range between 0.6 and 0.8) [19, 31]. Since we cannot know in advance how the documents are mapped to categories, we also cannot know the popularity distribution among the document categories. Thus, we test our algorithm using two scenarios.

In the first scenario, we initially use a Zipf distribution for the category popularities with a value for the Zipf parameter θ equal to 0.7. This is a worse case scenario since it is highly likely that the categories will contain a mixture of popular and unpopular documents, resulting in a category popularity distribution that is much less skewed than the one with θ=0.7.

In this first scenario, each document is associated with a category using a random number generator and taking into account the popularity of the categories as this is given from the initial Zipf distribution of the category popularities. It should be noted that the final category popularity distribution created in this way is Zipf-like, with several "spikes" in the popularities of categories, which give a significantly higher popularity for a large number of categories than what was predicted by the original Zipf category popularity distribution. The results are shown in Figure 2.

In the second scenario, we have a random assignment of documents to categories, which results in a near-uniform distribution of documents into categories. The results are shown in Figure 3.

We have tested a number of different configurations. The system configuration we present here consists of $|D|$=200,000 documents, $|N|$=20,000 nodes/peers, $|C|$=100 clusters, and $|S|$=500 semantic categories. Each one of the nodes has a relative processing capacity that is randomly chosen in the range [1...5]. Each node contributes a random number of documents, which span various categories (between 1 and 20). Note that this configuration corresponds to a system that is of much larger scale in terms of the number of nodes in it. This is so, since these 20,000 nodes are the "altruistic" nodes contributing documents, disk storage space, and processing capacity to our system. It is well known that the greatest percentage of nodes in systems like Napster and Gnutella, for example, were "free riders" [10] – these free riders are not included in our algorithms managing content and computational/storage resources. Please also note that, the larger scale of the system, the easier the load balancing problem becomes.

Discussion of the Results

In general, for all the tested cases the fairness achieved by MaxFair is greater then 95%.

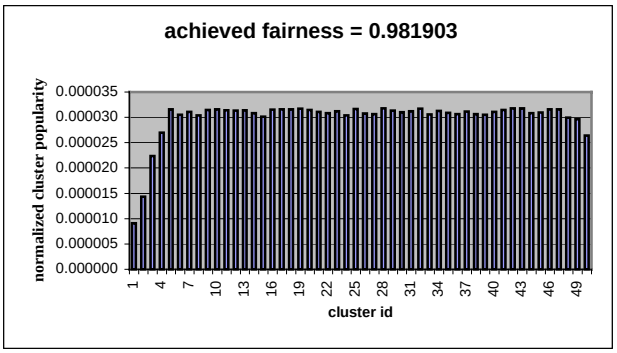

Figure 2: Normalized cluster popularities for a Zipf-like (θ=0.7) popularity distribution for the category popularities

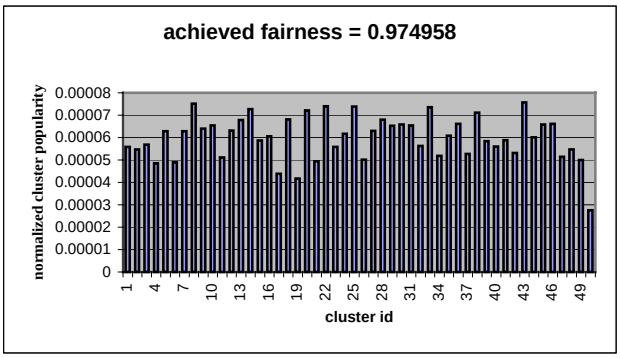

Figure 3: Normalized cluster popularities for a random popularity distribution for the category popularities

Also, as the number of categories and the number of clusters increases, the achievable fairness increases, as well; this is as expected, since with greater numbers of categories and clusters the balancing problem becomes inherently easier (there are more and smaller popularity values to be assigned to more clusters). However, even for small values of these parameters (50 clusters, 200 categories), the achievable fairness was above 90%.

In Figures 2 and 3 we show the results for a skewed assignment of documents to categories and for a Zipf

distribution of category popularities. The results testify to the very good performance of MaxFair.

As mentioned, we have also evaluated MaxFair under different access-distribution skews for documents and categories, cluster population configurations, scaling factors (on the number of documents, the number of nodes, the number of categories, and the number of clusters). We omit the results for space reasons. The general conclusion is that the MaxFair algorithm can (i) achieve very good to outstanding performance, and that (ii) its performance is robust across a wide range of system and problem parameter values and configurations.

5. Robustness under a Highly-Dynamic Environment

In the developed algorithms up to now we assumed to know the *static* popularities of the documents and the resulting popularities of the document categories. We also assumed to know the *static* population of peers and content in the system. One of the essential characteristics of a P2P content sharing system is its dynamic behavior, with the key properties of the system changing "on the fly" during the system's operation. Henceforth, we will study the impact of these dynamics on the developed solution so far and later, in Section 6, we will derive the additional infrastructure necessary to adapt to the highly dynamic environment.

Robustness under varying content population and popularities

We present here the results of our robustness experiments when the content population and content popularities vary while the system is in operation. Such a situation can occur when new users join the system or existing users quit the system, bringing/deleting new/existing documents, or when documents are added and deleted from the system. In that case the nodes of the system will vary and, consequently, so will the computational units that are contributed to the system. Hence, when such changes occur, the load allocation of the system will be affected.

In order to study this case we performed the following experiment. The popularity distribution of the document categories is Zipf-like with θ=0.7. The popularity distribution of the documents is Zipf with θ=0.8. Initially the MaxFair algorithm runs, placing the document categories into clusters of nodes. Then we add 5% new documents into the system, *which become the new most popular documents in the system*. The new documents that are added correspond to 30% of the total probability mass of the documents. The new documents are assigned randomly to some semantic categories. After the assignment we calculate the resulting fairness according to the initial placement of the semantic categories into clusters (i.e., we do *not* run again the MaxFair algorithm).

Note that this is indeed a severe stress test for the system. The results of these experiments are presented in Figure 4. As we can notice from the figure, fairness decreases. Note, however, that this test is a massive upset of the conditions with which the original load distribution occurred. Despite this, still the decrease in fairness is tolerable (in the worst-case the fairness drops from 1 to 0.78).

This experiment leads us to conclude that the problem's characteristics (especially the massive scale of documents and nodes) and our solution's characteristics (i.e., the clustering of peers, the grouping of documents, and their association) "help" the original solution computed by the MaxFair algorithm to be able to withstand such perturbations without significantly affecting the quality of the original solution, testifying to the robustness of our approach.

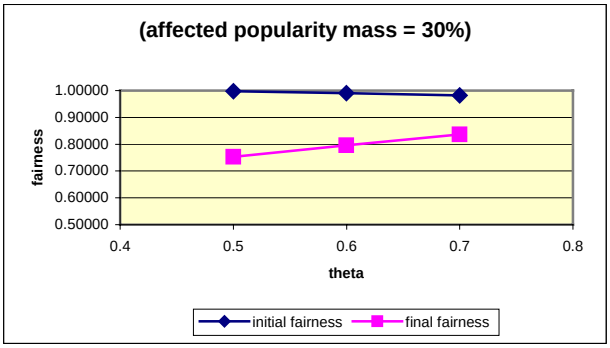

Figure 4: Fairness variations for varying content population and popularities.

6. Accommodating the Dynamics of Peers and Content

We now develop the additional mechanism, consisting of a cluster architecture, algorithms, and protocols necessary to dynamically adapt to the changing environment so that inter-cluster load balancing is ensured continuously. In particular, we will consider the following three cases:

1. The content popularity varies.
2. The content population varies (documents can be added or deleted).
3. The peer population varies (nodes can be added or deleted).

The same approach will be used for all these cases.

6.1 Content Popularity Variations

In order to handle the complexity and the associated overheads of content popularity variations, we adopt a hierarchical organization for the nodes in the clusters: at the top of the hierarchy, each cluster has a node, called the *leader* of the cluster.

6.1.1 Cluster Leader Election

The selection of the cluster leader is done as follows: Leaders are elected periodically (e.g., every day). Before the end of a period, nodes inform their cluster neighbors (the nodes in the cluster appearing in the node's NRT) of their computing, storage, and bandwidth capabilities, while also forwarding relevant information received by other nodes. Thus, over time, all nodes of the cluster have a quite clear picture of the status of all nodes in the cluster, as far as processing, storage, and bandwidth capabilities are concerned. When the time has come for the system to enter the adaptation stage, the most powerful node in each cluster is chosen to be the leader of the cluster (note that a node can be the leader in more than one cluster). Note, that this process may result in more than one peer believing to be the cluster leader (due to network partitionings, or when peers decide with incomplete information, for example). However, this poses no problem.

Of course, during the adaptation stage, nodes probe their cluster leaders to assure they are alive. In the case of a leader failure, another node is selected to be the new leader. This can be the next more capable node, or a node close to the leader in the tree hierarchy (i.e., one that has most of the information needed for the adaptation and needs to send out only a small number of messages to acquire all remaining information). Furthermore, this guarantees that all nodes of a cluster know which node is currently their leader.

Note that the selection of the cluster leader is performed on-the-fly when needed and assumes no previous explicit structure or knowledge. Thus this is indeed a pure P2P solution, since all nodes can become their clusters' leaders at some time and the parent-children relations are dynamic.

6.1.2 Dynamic Adaptation

Our approach is structured into four phases: a per-cluster monitoring phase, a leaders' communication phase, a fairness evaluation phase, and a load-rebalancing phase. In the following we present step by step the four phases of the adaptation mechanism. We assume that nodes keep statistics of the retrieval requests they served, broken down into per-category hits (implemented as per-category hit counters).

Phase 1. Per Cluster Monitoring

The leaders send to their cluster nodes a request for their hit counters. The request message is recursively sent from each node to its cluster neighbors, while loops in the graph are detected and broken by dropping request messages that have been seen again at a node. This results in the dynamic, on-the-fly creation of a tree structure from the cluster graph, where nodes consider the source of the request query to be their parent and the target of the forwarded queries to be their children.

Since the creation of the tree structure is orthogonal to its actual functionality, we can alternatively assume a preexisting tree structure, such as the one suggested in [26].

Eventually, the leader node ends up with the set of total per-category hits for the whole cluster in the last period. Naturally, these hit counter values reflect the current popularity of the categories that have been stored in the cluster.

The above description covered the no-failure case. As alluded to earlier, failures and faults may result in the physical partitioning of clusters, resulting in turn in the creation of multiple trees (sub-clusters) per cluster, which will participate independently in the adaptation process. Ensuring fault tolerance operation in P2P systems efficiently, is largely an open problem and certainly outside the scope of this paper. Here we only mention that there are mainly two alternatives: (i) allow sub-clusters, in different physical partitions, to form their own trees and proceed in the next phases of the adaptation, described below, and when the partitioning is resolved reconcile their actions, merging their trees; and (ii) impose limits (e.g., on the number of peer participants) so to reduce the possibility that more than one tree per cluster participates in the adaptation phases. In the following, we refer to a "cluster", and its "leader" but readers should keep in mind that it could be only a "sub-cluster" in the presence of failures.

Phase 2. Leader Communication Protocol

The leader nodes of clusters communicate with each other in order to share their clusters' load figures. In system configurations with fewer but larger clusters this can be done using traditional multi-cast protocols involving all leaders. For systems with very large numbers of clusters, the load information exchange can be done through "epidemic-style" protocols, enabling leaders of neighboring clusters to know about each other's load. Thus, at the end of this phase all communicating leaders know the current load distribution among their clusters and the load distribution among the document categories. Note that, since nodes in a cluster know their leader during adaptation, a cluster leader needs only contact one random node in every cluster to discover the cluster's leader.

Phase 3. Evaluation of fairness

A chosen leader (e.g., the leader of the cluster with the highest normalized popularity either among all clusters or among the clusters in a "neighborhood") measures the fairness index value. If it is above a low-threshold value (e.g., 90%) nothing is done. Else, the rebalancing phase is entered).

Phase 4. Load rebalancing among clusters

In this phase, the leader with the highest normalized popularity runs a variation of the MaxFair algorithm, called the MaxFair_Reassign algorithm, with which categories are reassigned to new clusters so that the load is balanced and fairness increases.

Algorithm MaxFair_Reassign
While (*fairness* < *threshold* **AND** *moves* < *max_moves* **)**

1. Among all clusters, find the cluster c_i with the highest normalized cluster popularity.
2. For every semantic category s of cluster c_i:
   a. For every cluster c_j different than c_i:
      i. Do dummy reassigns of s to c_j, recompute the resulting fairness values and keep the id of the cluster, m, during the move that gave the better result.
3. Actually reassign s to c_m (the procedure of moving around categories will be discussed later).
4. Update the normalized popularity of c_i and c_m and recalculate the *fairness* value.
5. Increment *moves*.

End while

The algorithm is greedy in the sense that it tries at each step to achieve the maximum gain in balance by making the best reassignment in terms of which category to reassign and to which cluster to reassign it and that it selects, at each iteration, a category from the cluster with the highest normalized cluster popularity.

Our primary concern was to ensure fast rebalancing, which implies that only a very small number of categories need be moved, since this will be the source of the associated cost.

Algorithm MaxFair_Reassign updates only the metadata kept by the nodes in the affected clusters. In this way, we avoid having to perform very large volume transfers, all in once, as a single transaction. Instead, we adopt a *lazy rebalancing protocol*, as will be explained below.

Lazy Rebalancing Protocol

After running MaxFair_Reassign:

1. Within the "source" cluster, all the nodes will be notified and update their metadata to reflect that the moved category is no longer to be served by this cluster's nodes. "Trace data" will be included pointing to the "destination" cluster to which the category has moved. Within the "destination" cluster all nodes' metadata will be also updated to reflect the reassignment.

2. Recall from Section 4.3.3 that the set of documents of each category have been partitioned into groups and different groups have been stored in the nodes of the "source" cluster. The transfer of these groups to the nodes of the "destination" cluster will occur by pairing nodes between the two clusters with each "source" node delivering its group to the "destination" node. These transfers can be taking place in parallel and can be scheduled for the first opportune time. The goal of this step is to break down a very large transfer to a number of much smaller transfer tasks. The transfer of higher popularity groups can be scheduled first, so to ensure faster rebalancing. The pairing between nodes in the source and destination clusters can be achieved in several ways, either using cluster node metadata found at super peers, or searching in pure peer-to-peer form by each node based on information exchanged by the leaders in phase 2 of the adaptation process.

3. Future requests for this category's documents will be initially coming into nodes of the "source" cluster from nodes of other clusters that were not notified of the reassignment. These requests will be forwarded to the "destination" cluster as follows. A node, say *n,* of the "destination" cluster will be randomly selected (among those that have been chosen to store the requested document) and the request and the id of the requesting node will be forwarded to *n*.

4. When a node from the "destination" cluster, *n,* receives a request, if *n* actually stores the requested document(s) (i.e., step 2 has reached *n* for the requested documents) it will send the reply to the requesting node. Else, *n* will explicitly request the document(s) from its coupling node in the "source" cluster, store, and then return the document(s) to the requesting node. It will also piggyback onto the reply the update in the metadata information reflecting the reassignment.

5. Periodically, all the nodes in the cluster send to their neighboring nodes updates to their metadata information that they have collected from the piggybacked responses to their requests. The nodes merge the information (resolving conflicts, as is explained below) and propagate it. This epidemic-style protocol eventually guarantees that all nodes of the cluster become aware of all metadata information updates.

Conflicts may arise in the metadata updates received from different neighbors of a node of a cluster during step 6 above (e.g., when a category has moved twice and one descendant node in the cluster is informed of the first move while another descendant node is informed of the second move). In order to facilitate conflict resolution, we extend the DCRT of each node to keep a per-category move_counter. When a category is moved to a new cluster the new cluster's nodes store as part of the metadata for this category a *move_counter* (incremented by one in step 1 above, for every move MaxFair_Reassign decides). Thus, to resolve conflicts, the metadata information with the highest move counter value is kept at the ancestor node.

6.1.3 Discussion

Having "traditional" distributed systems in mind, one might view the costs associated with rebalancing (i.e., the transfer of potentially thousands of documents) as prohibitively costly. However, we stress that P2P content sharing systems are definitively characterized by very large document transfers. For example, users routinely "download" up to hundreds of megabytes. Our effort, as exemplified in step 2 above, is intended to break down a huge "rebalancing" data transfer to a number of smaller ones, which mimic routine transfers to satisfy user requests.

Example. To illustrate this further, consider an example system with 200,000 nodes grouped in 400 clusters, each with 500 nodes. The documents have a size of 4MB (e.g., 3-minute MP3 clips). Let us suppose further that MaxFair_Reassign selects to reassign 10 categories, with each containing 1000 documents and that all documents are desired to have a minimum number of two replicas. This creates a data transfer for each reassigned category of 8GB (1000 * 4MB * 2). This large data transfer is broken down to 500 transfers (to the 500 nodes of the destination cluster) of 16MB each. Since 10 categories are reassigned, up to 5,000 pairs of nodes may be engaged in this transfer. Thus with our approach, in the example system of 200,000 active nodes engaged in content sharing, the major rebalancing cost, owing to the data transfers, "masquerades" as an increase of 2.5% on the active users, engaged in small-to-medium-size data transfers of 16MB each. ■

6.2 Content Population Variation

The new content, say a document, will either belong to an existing category, or not. In the former case the document will be stored in the cluster to which the document's category is assigned. The impact on the normalized cluster popularity will normally not be great. If the normalized cluster popularity is changed significantly after the addition of many documents, the system will face the change in the same way that was described above in section 6.1, using MaxFair_Reassign.

Publish Protocol

For every document d_i a node n wants to publish, it performs the following:

1. Extract the set of categories S associated with d_i.
2. For each s_i belonging to S, the node n checks whether an entry for a document associated with s_i already exists in DT. The existence of such an entry will mean that the current node has already published its contribution to the category s_i and needs not notify its peers again.
3. For each s_i for which no such entry exists, the node n uses its DCRT to extract the cluster-id associated with it. By default, all zero-document categories are initially mapped to the first cluster of the system (i.e. the one with cluster-id 0). This assures us that if two nodes concurrently publish documents in the same, previously empty, category, they will eventually end up publishing them to the same cluster, as will be shown.
4. Using its NRT, the node n sends a "publish" message to all the nodes belonging to the target cluster.
5. The nodes receiving the "publish" message reply with their DCRT and NRT metadata. Thus, if the category was once served by the target cluster but has since moved, the publishing node will be informed. It will then have to go back to step 4 and contact the nodes of the new target cluster. This procedure will be repeated until the correct target cluster has been found.

If the nodes receiving the "publish" message actually belong to the cluster serving the published category, they update their NRT about the addition of the new node. The node n will also update its metadata structures to reflect its addition to the target cluster. Since this way NRTs can grow very fast, an LRU replacement algorithm can be adopted in the maintenance of NRT entries.

6.3 Peer Population Variation

When nodes are added in the system or deleted from the system, the normalized popularities of the clusters will change because both content may be added/deleted and the number of computational units of the nodes that belong to the clusters changes. Normally, the impact on the normalized cluster popularity from additions and deletions of nodes will not be high and these additions/deletions of nodes will probably be affecting the different clusters in a near-uniform manner. At any rate, the rebalancing in this case will be performed according to the procedure described in Section 6.1.

The main additional problem that we have to face in the case of deletions of nodes is the (un)availability of documents. When a node leaves the system, the nodes of its cluster must be informed about the data that will no longer be available. So, one solution to the problem is to force the nodes to send "leaving" messages to the other nodes of their cluster. In this way, the metadata of each cluster will be reorganized progressively and additional steps will be taken, if needed (e.g., to create an additional copy of documents whose desirable replication degree is to be violated).

The main additional problems we have to face during the addition of nodes are: (i) selecting which documents the new nodes will store, and (ii) the cost in order for these nodes to become full members of the system, which primarily consists of the transfer (of the selected documents and the cluster's metadata) to the nodes.

Note that this data transfer will be needed *only* the first time the user node joins our system; subsequent

reconnections will assign the user to the cluster that contains the categories of the stored documents in the user's node.

Similarly, selecting which documents the new node will store is required only when the user joins the system for the first time; subsequent reconnections will assign the user node to the proper cluster. Our approach is first to store to new users' nodes the content of those nodes in the same cluster that store the most popular documents. Given this decision, we use the MaxFair algorithm to decide to which cluster to assign the new node.

Finally, for both additions and deletions of nodes, the leader of the cluster is notified so that the metadata about which nodes store which documents and about the new tree structure can be updated and propagated down the tree at an opportune time to all cluster nodes.

Join Protocol

Assume we have a node *n* that wishes to join the system. The join protocol proceeds as follows:

1. The node *n* finds a node *n'* already participating in the system. This can be done, for example, by means of out-of-band communication, execution of specialized node discovery algorithms (e.g. expanding-ring search), contacting a stable/known/super-peer node of the system etc.
2. The node *n* contacts node *n'* and retrieves its metadata information (namely, its DCRT and NRT).

The node *n* will then execute the publish protocol for every document *d* it wishes to contribute to the community. If no such documents exist (i.e. the node is a free-rider), it will perform a dummy publish, so that it will be added to a cluster and receive further updates of NRTs and DCRTs.

6.4 The Performance of Rebalancing

In this subsection we present some results regarding the performance of our load rebalancing approach. We focus on the internals of the MaxFair_Reassign algorithm, and in particular, on the number of reassignments required to ensure that the fairness of the inter-cluster load balancing is within acceptable levels (i.e., within the upper and lower thresholds).

In Figure 5, we present five experiments with each experiment producing an initial configuration and inter-cluster load balancing performed by MaxFair for the "challenging" case defined by an initial skewed Zipf popularity distribution among the documents and among the categories ($\theta = 0.8$). In all experiments the reason for requiring load rebalancing was the addition of new documents, the total popularity of which amounts to 30% of the total document popularity and which are distributed randomly to the categories.

The upper and lower threshold values for the fairness are 92% and 83% respectively. The system consists of the same number of documents, categories, clusters, and nodes as for the test cases in Sections 4 and 5.

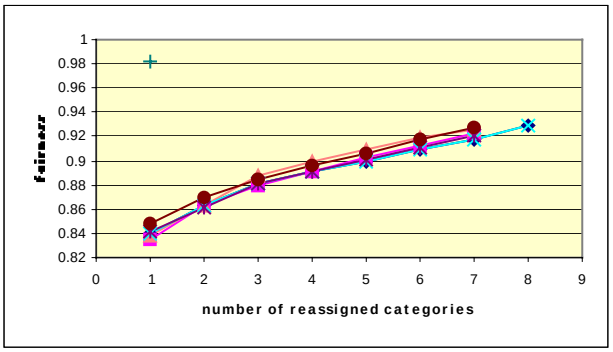

Figure 5: The MaxFair_Reassign algorithm, when the 30% of the popularity mass changed, for five different experiments.

It should be apparent that the above scenario represents a very challenging test case for our approach. The expected changes in a real system will bring about a significantly lower change than the one captured by the above test case.

The results show clearly that the number of reassignments required in order to maintain very high load distribution fairness among the clusters is small (7 or 8). Finally, because:

1. The fairness of the initial inter-cluster load balancing achieved through algorithm MaxFair is very high and,
2. With a small number of reassigned categories the algorithm MaxFair_Reassign can improve fairness significantly, and
3. the cost for each category reassignment is small, as argued above,

we can conclude that the overall cost of our solution for rebalancing the load is small.

7. Conclusions

The P2P paradigm is becoming increasingly popular for developing internet-scale applications. P2P content sharing systems, as popularized by the initial endeavors of Napster, Gnutella, etc., are receiving increasing attention from academics and industry as an important class of internet data management applications.

In this paper we have presented the problem of managing content and resources in P2P sharing systems so to ensure the efficiency of operation. Efficiency is viewed both from the point of view of the system, in the sense of ensuring globally fair load distribution among all peers, and from the point of view of the users, in the sense of facilitating low user-request response times.

We have presented (i) an overall system architecture, (ii) a formal problem formulation, and, (iii) algorithms, protocols and mechanisms that achieve our performance

goals. To our knowledge this is the first paper that tackles the load distribution problem head on and at the same time facilitating fast response and giving worst-case guarantees for the response times user queries suffer. Achieving these goals is a formidable challenge given the need to respect the autonomy of peers, their heterogeneity (in terms of storage and processing capacity), the peer population dynamics (e.g., peers enter and leave the system at their free will), and the content population dynamics (e.g. documents being added/deleted at any time and document populations are varying with time).

The proposed architecture is based on (i) the grouping of documents, (ii) the clustering of peer nodes, and (iii) the association of document categories to peer clusters. We studied the problem of inter-cluster load balancing, presenting complexity results and proposing a novel algorithm. Our results show that the proposed algorithm, which utilizes a novel metric of fairness for our context, can achieve very good to excellent load balancing results in the sense that only a small percentage of clusters receives unfair loads. Our solution takes into account the different content contributions of peers and the different processing and storage capacities of peers.

We have tested the robustness of the proposed solution under varying content population and content popularities. Our results here showed that the fundamentals of our approach lead to an inherently robust solution, even under drastic changes in the conditions, which existed when the original solution was computed.

Finally, we have also presented the additional architecture, algorithms, and protocols necessary to accommodate the dynamics of the environment (peer population changes, content population changes and content popularities changes) and we presented arguments regarding the efficiency of our approach.

We believe the above constitutes the first effort towards an architecture that addresses comprehensively the problem of fair load distribution and facilitates low response times in P2P sharing systems. With this paper we have presented evidence that testifies to the benefits and the validity of our approach, which is a radically different proposal for architecting P2P sharing systems. However, to complete our approach, a large number of issues, of both theoretical and "systems" flavor, remain open. These include, (i) the development of optimal algorithms for inter-cluster load balancing and heuristics achieving near-optimal performance; (ii) optimal system configurations, in terms of the number of clusters versus the number of nodes per cluster; (iii) epidemic-style algorithms (a la Freenet and OceanStore[16]) for leader collaboration in systems with very large numbers of clusters, (iv) alternative architectures for each cluster, (v) alternative definitions/metrics for fairness and related algorithms, (vi) the optimal "granularity" (i.e., whether nodes, documents, or whole categories should be moved) when correcting imbalances between clusters, (vii) alternative, more space-efficient document placement policies and related algorithms that guarantee intra-cluster load balancing, and (viii) cache placement and replacement algorithms that can complement our architecture. Future efforts should also involve the implementation of "champion applications" and the empirical evaluation of the system's performance. We call on the community to join us in tackling these research problems.

Acknowledgements

This work was supported in part by project DIET (IST-1999-10088) funded by the IST Programme of the European Commission, under the FET Proactive Initiative on "Universal Information Ecosystems". We would like to acknowledge the contributions of all partners of DIET (Btexact Technologies, Universidad Carlos III de Madrid, DFKI) to this work.

8. References

[1] A. Crespo, H. G. Molina. *Routing Indices for Peer-to-Peer Systems*. In Proceedings of ICDCS, 2002.

[2] A. Montresor, O. Babaogly and H. Meiling. *Load-Balancing through a Swarm of Autonomous Agents*. Proceedings of the Workshop on Agents and P2P Computing, Bologna, Italy, July 2002. LNCS, Springer, forthcoming.

[3] A. Rowstron and P. Druschel. *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems*. Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001), November 2001.

[4] Audio Galaxy web site: http://www.audiogalaxy.com

[5] Autonomy web site: http://autonomy.com

[6] B. Yang and H. G. Molina. *Comparing Hybrid P2P Systems*. Proceedings of the 27th VLDB Conference, Roma, Italy 2001.

[7] B. Yang and H. G. Molina. *Improving Search in P2P Networks*. In Proceedings of ICDCS 2002, July 2002.

[8] B. Zhao, J. Kubiatowicz, and A. Joseph. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.

[9] C. Hoile, F. Wang, E. Bonsma, P. Marrow. *Core Specification and Experiments in DIET: A Decentralised Ecosystem-inspired Mobile Agent System*. In Proceedings of AAMAS 2002.

[10] E. Adar and B. A. Huberman. *Free riding on Gnutella*. Technical report, Xerox PARC, 10 Aug. 2000.

[11] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial*

Intelligence. Oxford University Press, 1999.

[12] E. Bonsma and C. Hoile. *A distributed implementation of the SWAN peer-to-peer lookup-system using mobile agents*. Proceedings of the Workshop on Agents and P2P Computing, Bologna, Italy, July 2002. LNCS, Springer, forthcoming.

[13] Gnutella web site: http://gnutella.wego.com

[14] Ian Clarke, O. Sandberg, B. Wiley and T. W. Hong. *Freenet: A Distributed, Anonymous Information Storage and Retrieval System.* ICSI, Berkley, California. Workshop on Design Issues in Anonymity and Unobservability, 2000.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. *Chord: A scalable peer-to-peer lookup service for Internet applications*. Proceedings of the 2001 ACM SIGCOMM Conference.

[16] John Kubiatowicz, et al. *OceanStore: An Architecture for Global-Scale Persistent Storage*. Proceedings of ASPLOS 2000.

[17] K. Aberer, M. Punceva, M. Hauswirth and R. Schmidt. *Improving Data Access in P2P Systems.* IEEE Internet Computing, January-February 2002.

[18] Kazaa web site: http://www.kazaa.com

[19] Lee Breslau, Pei Cao, Li Fan, Graham Philips, Scott Shenker. *Web Caching and Zipf-like Distributions: Evidence and Implications*. IEEE INFOCOM, pp. 126 –134, 1999.

[20] Napster web site: http://napster.com

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability- A Guide to the Theory of NP-Completeness* w. h. Freeman and Co. 1979.

[22] O. Babaoglu, H. Meling, and A. Montersor. *Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems*. In Proceedings of ICDCS 2002, July 2002.

[23] P. Marrow, M. Koubarakis, R.H. van Lengen, F. Valverde-Albacete, E. Bonsma, J. Cid-Suerio, A.R. Figueiras-Vidal, A. Gallardo-Antolin, C. Hoile, T. Koutris, H. Molina-Bulla, A. Navia-Vazquez, P. Raftopoulou, N. Skarmeas, C. Tryfonopoulos, F. Wang, C. Xiruhaki. *Agents in Decentralised Information Ecosystems: The DIET Approach*. Symposium on Information Agents for E-Commerce, AISB'01 Convention, 2001.

[24] R. Bhargava, A. Goel and A. Meyerson. *Using approximate majorization to characterize protocol fairness*. SIGMETRICS/Performance, pp. 330-331, 2001.

[25] R. Jain D-M. Chiu, W.R. Hawe. *A Quantitative Measure of Fairnsess and Discrimination for Resource Allocation in Shared Computer Systems.* DEC-TR-301, 1984.

[26] R. Renesse, K. Birman, D. Dumitriu and W. Vogels. *Scalable Management and Data Mining using Astrolabe*. Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002

[27] Semio web site: http://www.semio.com

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A scalable content-addressable network*. In Proceedings of ACM SIGCOMM 2001, August 2001.

[29] T. Stading, P. Maniatis, M. Baker. *Peer-to Peer Cashing Schemes to Address Flash Crowds*. Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2002), March 2002.

[30] Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suciu. *What Can Peer-to-Peer Do for Databases, and Vice Versa?* Proceedings of the Fourth International Workshop on the Web and Databases (WebDB '2001).

[31] V. Almeida, A. Bestavros, M. Crovella and A. de Oliveira, *Characterizing reference locality in the WWW*. Proceedings of 1996 International Conference on parallel and Distributed Information Systems (PDIS '96), 1996.

[32] T. Joachims. *Text categorization with Support Vector Machines: Learning with many relevant features*. In Machine Learning: ECML-98, Tenth European Conference on Machine Learning.