Modeling Peer-Peer File Sharing Systems

Zihui Ge, Daniel R. Figueiredo, Sharad Jaiswal, Jim Kurose, Don Towsley
Department of Computer Science
University of Massachusetts, Amherst
{gezihui, ratton, sharad, kurose, towsley}@cs.umass.edu

Abstract— **Peer-peer networking has recently emerged as a new paradigm for building distributed networked applications. In this paper we develop simple mathematical models to explore and illustrate fundamental performance issues of peer-peer file sharing systems. The modeling framework introduced and the corresponding solution method are flexible enough to accommodate different characteristics of such systems. Through the specification of model parameters, we apply our framework to three different peer-peer architectures: centralized indexing, distributed indexing with flooded queries, and distributed indexing with hashing directed queries. Using our model, we investigate the effects of system scaling, freeloaders, file popularity and availability on system performance. In particular, we observe that a system with distributed indexing and flooded queries cannot exploit the full capacity of peer-peer systems. We further show that peer-peer file sharing systems can tolerate a significant number of freeloaders without suffering much performance degradation. In many cases, freeloaders can benefit from the available spare capacity of peer-peer systems and increase overall system throughput. Our work shows that simple models coupled with efficient solution methods can be used to understand and answer questions related to the performance of peer-peer file sharing systems.**

I. Introduction

Peer-peer networking has recently emerged as a new paradigm for building distributed networked applications. The peer-peer approach differs from the traditional client/server approach towards building networked applications in several crucial ways. Perhaps most importantly, a peer is both a producer and a consumer of the implemented service. In a peer-peer file-sharing application, for example, a peer both requests files from its peers, and stores and serves files to its peers. A peer thus generates workload for the peer-peer application, while also providing the capacity to process the workload requests of others. As a result, an increase in the number of peers results not just in an increase in workload, but also in a concomitant increase in the capacity to serve the workload. In the traditional client-server approach, a clear distinction exists between the consumers and producers of a service — clients generate workload and workload is processed by servers; an increase in the number of clients results simply in an increase in workload. A second important difference is that a peer's lifetime in the system is transitory — a peer may be active in the system for some time (both generating requests and serving the requests of others) and then go off-line, removing itself from the system.

This research has been supported in part by the NSF under grant awards ANI-9809332, ANI-9805185, and EIA-0080119, by DARPA under contract 9608-02-22ALAN010 and by CAPES (Brazil). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Considerable research has been devoted to developing a fundamental understanding of the performance of traditional client-server applications [1]. For example, various models exist for predicting the system throughput and average response time of a workload that is load-balanced among a set of servers. Perhaps because the field is so new, significantly less work has been devoted to developing a fundamental understanding of peer-peer applications. Much of the research to date in peer-peer networking has focused on developing point solutions to specific peer-peer problems, or on measuring client workloads and serving characteristics of currently deployed peer-peer systems.

To our knowledge, no study has yet evaluated fundamental performance issues of peer-peer file sharing systems. It is difficult to evaluate any conjecture in real deployed peer-peer systems, given the unregulated and transitory nature of such networks. Thus, a formal framework is clearly needed to provide for the systematic evaluation of such performance issues. For example, it is generally assumed that peer-peer systems can scale better than the traditional client/server approach. However, there has not been any quantitative evaluation of the scalability of peer-peer systems. Moreover, it is commonly assumed that freeloaders can degrade the performance of peer-peer systems since they do not contribute to the capacity of the system. Again, we are not aware of any study that examines the impact freeloaders have on the overall system performance and on other non-freeloading peers.

The goal of our work is to develop simple mathematical models that can be used to illustrate and illuminate fundamental performance issues of peer-peer file sharing systems. We introduce a flexible mathematical abstraction of a peer-peer file sharing system that is general enough to capture the essence of different architectures and different peer behaviors. We also present an approximate solution method for the system throughput which will be used to analyze the system performance. By defining specific model parameters we capture three different architectures (centralized indexing, distributed indexing with flooded queries, distributed indexing with routed queries) and two classes of peers (non-freeloaders and freeloaders).

A particular strength of our approach is the generality of the modeling framework which allows one to explore a range of issues and performance trade-offs of file sharing peer-peer systems. In particular, we address the following fundamental issues: **(i)** the scalability of different architectures with the number of peers; **(ii)** the impact on such systems due to the presence of different classes of peers (e.g., freeloaders); **(iii)** the impact of imbalance in service capacity and file request load on such systems.

We now summarize some of our results:

- The use of limited-scope flooding of queries has a seriously negative impact on the system performance. This approach has an inherent handicap in not being able to fully capitalize on the potential capacity of a peer-peer system.
- Contrary to what one might expect, freeloaders often do not have a significant negative impact on the performance of other peers. In fact, peer-peer systems are likely to have spare capacity which freeloaders can benefit from.
- It is known that well designed distributed indexing architectures like CAN and Chord [2], [3] can scale with population size. However, we show that a centralized indexing approach can scale gracefully until the system reaches its capacity to process queries, and can outperform the other architectures for small population sizes.
- The performance of a peer-peer system shows only a minor degradation when the most requested files are not the most widely available.

In the following section we provide an overview of current peer-peer file sharing architectures and discuss their differences. In Section III we introduce an abstract model for peer-peer file sharing systems. The model parameters corresponding to each particular architecture are defined in Section IV. In Section V, we derive an approximate solution technique for the system throughput. Numerical results and comparison between different architectures under various conditions are presented in Section VI. Finally, we conclude the paper in Section VII.

II. Background

It is possible to divide the currently proposed peer-peer file sharing systems into three different architectures. The earliest design uses a central server (or server cluster) to coordinate participating nodes and to maintain an index of all available files being shared. When a peer node joins the system, it contacts the central server and sends a list of the local files that are available for other peers to download (shared files). To locate a file, a peer sends a query to the central server, which performs a database lookup and responds with a list of peers that have the desired file. If a peer leaves the system, its list of shared files is removed from the central server. We will denote such an architecture as a CIA (Centralized Indexing Architecture). An example of such a system is the Napster network [4].

The two other architectures eliminate the central server and distribute the indices of available files among participating nodes; they differ from each other primarily in the manner in which they distribute the file indices. In one approach, each peer is responsible for maintaining the indices of only the files it stores. A limited-scope query message is flooded to the network when a peer wants to locate a file in the system. All nodes reached by the flooding that have a positive match with the query reply to the peer that originated the request. Note that since query messages have limited-scope, it is possible that peers cannot locate files even though the files are available in the system. This type of architecture will be denoted as DIFA (Distributed Indexing with Flooding Architecture). It is typified by the Gnutella network [5].

The third approach eliminates flooding by systematically distributing the file indices among participating nodes, with queries being routed directly to the node responsible for that subset of the file index. When a node joins the system, it is assigned a subset of the index space and it receives all keys ($\langle$ file name, peer address $\rangle$ pairs) for that subset. If a peer leaves the system, another peer (or peers) becomes responsible for its subset of the file index. We will denote such systems as DIHA (Distributed Indexing with Hashing Architecture) which are typified by Chord [2], CAN [3], Pastry [6] and Tapestry [7].

Despite their architectural differences, three common important functions are performed by all peers in all architectures: **(i)** *maintenance of the infrastructure of the peer-peer system.* This involves handling peer arrivals and departures and ensuring network connectivity among peers; **(ii)** *handling queries.* This involves query propagation and response mechanisms; **(iii)** *file transfers.* This last function involves reliable file download, and is performed similarly in all three architectures, since after locating the destination node a direct file transfer takes place, between the peer requesting the file and the one storing it. The manner in which the first two functions above are accomplished, however, differ significantly among the three architectures. In the central server architecture, the central server handles all infrastructure maintenance and query processing functions. In contrast, these functions are distributed among all the peers in the system in the two distributed architectures.

III. Model for Peer-Peer File Sharing Systems

In order to capture the distinguishing characteristics of peer-peer file sharing systems, we seek a simple, yet representative, mathematical abstraction. We model the peer-peer system as a multiple class closed queueing network where each class consists of a fixed population of peers, given by $\mathbf{N} = (N^{(1)}, \ldots, N^{(C)})$, where C is the number of classes. Peers can be either on-line, participating in the system, or off-line. While on-line, a peer generates workload by posing queries to locate files and performing file downloads. When a peer joins the system, moving from the off-line to the on-line state, it not only generates workload, but also brings service capacity to the system. In the distributed architectures, a peer participates in infrastructure maintenance and query processing (which we will term "common service"). In all architectures a newly arriving peer will also be available to serve files to its peers, which increases the service capacity of the system. After downloading a file, a peer either leaves the system (and returns to the off-line state) or remains in the system (to generate subsequent requests, and to provide file transfers to other

M	total number of distinct files in the system
C	number of classes of peers
$N^{(c)}$	population size of the c-th class of peers
$\mathbf{N}$	vector $(N^{(1)}, \ldots, N^{(C)})$ of population sizes
$N_a^{(c)}$	no. of peers on-line from the c-th class
$\mathbf{N_a}$	vector $(N_a^{(1)}, \ldots, N_a^{(C)})$ of peers on-line
$1/\lambda_{off}^{(c)}$	average off-line time of the c-th class
$1/\lambda_{idle}^{(c)}$	average idle time of the c-th class
$\mu_q(\mathbf{N_a})$	service rate of common service queue
$\mu_f(\mathbf{N_a}, i)$	service rate for downloading the i-th most replicated file
p_j	prob. that a request is associated with j-th most popular file
$p_{off}^{(c)}$	prob. that a peer from the c-th class goes off-line
$q_f(N_a, i)$	prob. that a query for the i-th most replicated file fails
$T^{(c)}$	throughput of c-th class observed at reference point **A** in Figure 1
T	system throughput, $\sum_{c=1}^{C} T^{(c)}$, observed at reference point **A** in Figure 1

TABLE I
NOTATION AND MODEL PARAMETERS.

peers). Figure 1 illustrates a model that captures this behavior, while the notation for the model is introduced in Table I. We next consider the individual components of this model in more detail.

The common services component of the system is abstracted by having a single server queue represent query processing. In the distributed architectures, peers cooperate to process queries, and hence the service rate $\mu_q(\mathbf{N_a})$ for this queue is an increasing function of the number of on-line peers. $\mathbf{N_a}$ is a vector containing the number of peers on-line from each class. As noted earlier, peer-peer architectures differ significantly in how they process queries. Thus, $\mu_q(\mathbf{N_a})$, appropriately chosen, can be used to model the distinctive characteristics of the search mechanism of each architecture.

An important abstraction of our model is that each distinct file being shared in the system will have a certain service capacity associated with it. That is, there will be some number of copies of each file in a peer-peer network, and the larger the number of copies, the larger the capacity of the peer-peer system to serve this file. This *abstraction* is modeled by associating a single server queue with each distinct file in the system (we assume there are M distinct files). All requests to download a particular file must enter its corresponding queue to be served. The service capacity to serve a file depends on the number of replicas of that file present in the system, which in turn generally depends on the popularity of that file. The number of replicas in the system also depends on the number of peers currently on-line. Thus, we denote the service capacity for the i-th most replicated file as $\mu_f(\mathbf{N_a}, i)$, where i varies from 1 to M.

Another important aspect of file sharing applications is that queries are not uniformly distributed among the M distinct files available in the system. Thus, let $j = 1, \ldots, M$ indicate the rank of the file based on the number of requests it receives. Let p_j denote the probability that a query is directed to the j-th most requested file. Note that the j-th most popular file might not be the same as the j-th most replicated file. If a file has recently become "hot" it might receive more requests than any other file in the system, however, it might not be the most replicated file of the system.

In all architectures, a query request can fail to locate the desired file. This can happen for one of several reasons: **(i)** the desired file is not present in the system; **(ii)** peers that hold the desired file are off-line; **(iii)** the query message did not reach the peer node that holds the index for the file being searched. Again, we want to model the fact that queries fail with unequal probability across the files available in the system. The probability that a query fails is a function of the number of replicas of the desired file. Moreover, the number of on-line peers also affects the probability that a query will fail. Thus, let $q_f(\mathbf{N_a}, i)$ denote the probability that a query for the i-th most replicated file fails given that there are $\mathbf{N_a}$ peers on-line. By appropriately choosing $q_f(\mathbf{N_a}, i)$ we can model different architectures and their behaviors with respect to query failure.

When peers are on-line, they are not always posing queries and downloading files. A peer might remain silent and neither generate queries nor download files for a period of time. This behavior is modeled by including a "think phase" that each peer must go through before posing a query. User behavior (while on-line) thus consists of alternating intervals of think time and file query/download activity. The think time period for the c-th class ($1 \leq c \leq C$) is modeled by an infinite server queue with mean service time $1/\lambda_{idle}^{(c)}$. The time spent querying/downloading will depend on system architecture, load, and other specific system parameters, as will be discussed later.

The final component of our model captures the behavior of peers going off-line and, subsequently, coming back on-line. After downloading a file a peer might decide to leave the peer-peer system (go off-line), which occurs with a class dependent probability $p_{off}^{(c)}$. Note that $p_{off}^{(c)}$ determines the expected number of downloads a peer from class c performs before going off-line, which is given by $1/p_{off}^{(c)}$. Upon leaving the system the peer will remain off-line for a period of time before going on-line again. This corresponds to users exiting the application and executing it again at a later point in time. Peer behavior thus consists of alternating intervals of being on-line and off-line. The off-line period for the c-th class is modeled by an infinite server queue with mean service time $1/\lambda_{off}^{(c)}$.

A. Multiple classes of peers

Recent measurement studies have observed a significant disparity in peer behavior. For example, in [8] the authors point out that nearly 70% of Gnutella users do not share any files; these peers are classified as "freeloaders". More generally, in [9] the authors identify two distinct classes of peers: **(i)** a set that behave more like servers, adding capacity to the system (by sharing many files) and requesting fewer downloads; **(ii)**

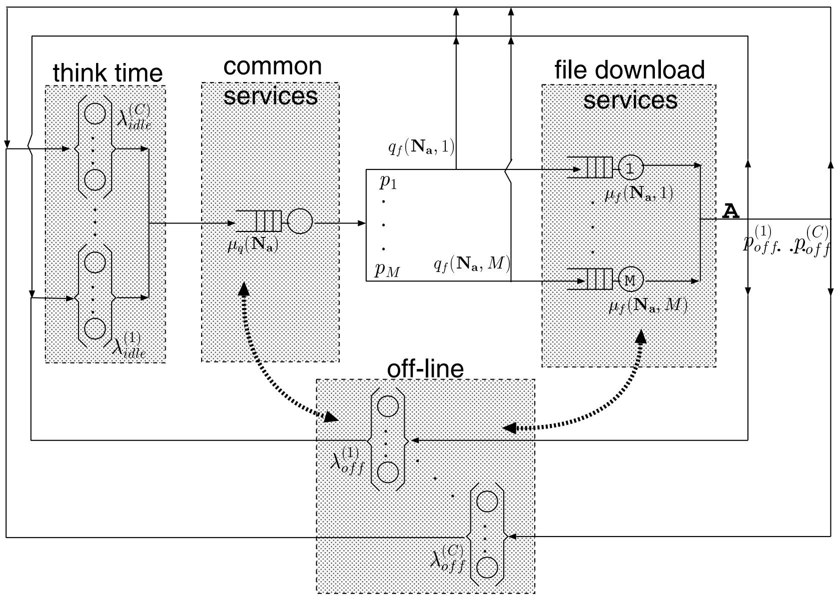


Fig. 1. Model for peer-peer file sharing systems

a set that behaves more like clients, adding little capacity to the system and requesting more downloads.

Our model captures this important aspect by having multiple classes of peers and by providing differentiated treatment according to their class. We distinguish among different classes by giving different values to model parameters that characterize the peer behavior, such as average think time, probability of going off-line, and average time off-line, on a per class basis.

Moreover, having the service capacity (query processing and file downloading) depend on $\mathbf{N_a}$ allows the model to differentiate the amount of capacity that peers in different classes bring into the system. Note that different classes of peers can be used to distinguish not only among user behavior, but also other characteristics, such as different peer bandwidth connectivity.

B. Model extensions

It is worth noting that the framework on which our model is constructed is easily extended to accommodate other characteristics of peer-peer file sharing systems. In particular, we have extended it to capture the impact that peers have on the system when making the transition from the off-line to the on-line stage. During this phase, the system adapts to the new peer and may perform local (or global) updates in the file index. This transient stage can be modeled by another infinite server queue located just after the off-line stage. While in this transient stage, peers effectively reduce the service capacity of the system. We believe that adding this subsystem to the model has little impact on the general trend of our results. Two reasons can help explain why: (i) the relatively short period of time peers stay in the transient stage (compared to on-line and off-line periods, which are several orders of magnitude larger); (ii) the relatively small overhead of adapting the system to the new peer (an average update should not involve significantly more work than the equivalent of a few queries). We have considered this issue in a more detailed version of this work [10].

Another possible extension is to allow for a variable number of peers in the entire system. This would eliminate our fixed population assumption and allow us to examine the system under a varying population. The model could be extended by introducing an exogenous arrival process of peers to the on-line stage and allowing peers to probabilistically depart the system. Although we have not pursued this modification, we believe such extensions are easy to incorporate in our current modeling framework.

IV. Model Parameters

In the previous section, we presented a peer-peer file sharing model in which model parameters are functions of $\mathbf{N_a}$, file popularity and file replication, which we now elaborate in more detail.

Only recently has research been conducted on characterizing the workload and the service capacity of existing peer-peer file sharing applications. To our knowledge, there exists only a few measurements studies performed on actual peer-peer systems that attempt to characterize the available content and peer behavior [8], [11], [12]. Thus, we will rely on our understanding of different architectures, coupled with results from these measurement studies, to guide the choice of model parameters for each type of architecture. Note that the model and the approximate solution method (to be described in Section V) are flexible with respect to the choice of parameters and functions used to represent the service capacity. Therefore, it is possible to modify or replace parameters and functions of the model and still solve it using the solution method proposed. We first present the parameters for the single class case, i.e.,

$C = 1$ and $\mathbf{N_a} = (N_a^{(1)})$. We will discuss the case where $C = 2$ later.

Results from a measurement study of the Gnutella network [11] have shown that most file queries are directed to a few, highly popular topics. Other studies [8] have shown that the number of replicas of a certain file in Napster and Gnutella is also heavily skewed. A promising candidate distribution that has been shown to capture such a popularity characteristic is the Zipf distribution. This distribution can be used to determine the probability that a query is associated with the j-th most requested file (p_j) and also to determine the service capacity of the i-th most replicated file. We will use i to denote rank according to the number of file replicas and j to denote the rank according to the number of requests a file receives. Using the Zipf distribution, we have that $p_j \propto 1/j^\alpha$, where α is the scaling parameter of the distribution. Since $\sum_{j=1}^{M} p_j = 1$, we have

$$p_j = K/j^\alpha \tag{1}$$

where $K = 1/\sum_{j=1}^{M} 1/j^\alpha$.

The service capacity for a given file in the system is directly proportional to the number of replicas of that file and to the number of peers that are currently on-line. Since the number of replicas of a file are described by Zipf's distribution, we model the service rate of the i-th most replicated file by

$$\mu_f(\mathbf{N_a}, i) = \frac{N_a^{(1)}\ H\ K}{i^\alpha} \tag{2}$$

where H represents the basic service rate associated with the contribution of a single peer to the file service capacity. Note that all M distinct files are assumed to be available whenever one or more peers are on-line.

We assume that the file download component is identical for all peer-peer architectures, since files are shared by peers and are directly downloaded from one another. However, the architectures differ significantly in the manner in which participating peers contribute to the common services. Hence, this component can be crucial in illuminating the performance differences among the three architectures. An important parameter is how balanced is the query workload distributed among peers on-line, which will be quantified by $\theta, 0 < \theta \leq 1$. To exemplify, consider a server cluster that is fully balanced, in which case we say $\theta = 1$. However, if the workload of the cluster is skewed towards a small subset of the servers, then not all available service capacity can be utilized. Thus, the effective service capacity of the system is less than the sum of the capacity of the individual servers. In this case we say $\theta < 1$. Peer-peer systems with distributed architecture are subject to this imbalance, as queries generated by peers tend to have a skewed distribution.

We now explore the distinctive characteristics of the three architectures and describe the rationale behind our choices of $\mu_q(\mathbf{N_a})$.

- **CIA** - Centralized Indexing Architecture
  In such an architecture, there exists a single server that maintains the index of all files and performs all query lookups. We thus model the service capacity of the common services as being independent of the number of users on-line. Thus,

  $$\mu_q(\mathbf{N_a}) = C_1$$

  where $C_1 > 0$ is determined by the capacity of the central server to perform query lookups.
- **DIFA** - Distributed Indexing with Flooding Architecture
  In such an architecture, each node maintains the indices of the files it owns and responds positively to matching queries. Thus, the system-wide query service capacity increases linearly with the number of peers on-line. However, due to the limited-scope flooding, a query reaches a bounded number of peers in the system. Thus, the system-wide query service capacity should be scaled down by this factor. We denote the number of peers a query reaches by $\mathcal{T}^\beta$, where $\mathcal{T}$ is the value of TTL, which indicates the maximum number of hops a query message traverses in limited-scope flooding, and $\beta > 1$ is a parameter related to the connectivity of the topology formed by the peers. Hence we define

  $$\mu_q(\mathbf{N_a}) = \frac{C_q}{\mathcal{T}^\beta}\ N_a^{(1)}\ \theta$$

  where $C_q > 0$ is determined by the capacity of a single peer to process a query. Note that in this architecture there is little imbalance in query processing, since queries are not directed at any particular set of peers, but rather flooded to locate files, thus, $\theta \approx 1$.
- **DIHA** - Distributed Indexing with Hashing Architecture
  In such an architecture, each peer is responsible for a subset of the index space, thus, as in DIFA, the system-wide query service capacity increases linearly with the number of peers on-line. However, the average number of peers traversed to answer a query, and hence involved in processing it, increases in proportion to $\log(N_a^{(1)})$ [2]. Thus, we scale down the system service capacity by this factor. Moreover, the DIHA architecture suffers from the imbalance phenomena, as specific peers are assigned fixed subsets of the index space, and queries will not be distributed uniformly across the index space. Hence, for this architecture, $\theta < 1$. Thus, we have

  $$\mu_q(\mathbf{N_a}) = \frac{C_q}{\log(N_a^{(1)})}\ N_a^{(1)}\ \theta$$

  Note that C_q is the same in both distributed architectures as we assume that peers have identical service capacity independent of the architecture.

Another important difference between the three architectures is how queries fail, which determines the probability of query failure. If we assume that all queries are for files that exist in the system, that a peer that holds the index for this file is always on-line, and that communication is reliable, then a query fails only if it is unable to reach the peer node that holds the index for the file being searched. In CIA and DIHA, queries are always routed to the peer node that contains the index for the file being searched, hence these architectures can always locate the requested file. For these systems

$$q_f(\mathbf{N_a}, i) = 0$$

However in DIFA, since a query is flooded only to a bounded number of peers, there is a chance that the query might not reach a peer that has the desired file. Thus, the probability that a query for a given file will succeed is proportional to the ratio of the TTL to the diameter of the peer-peer network (approximated as $(N_a^{(1)})^{1/\beta}$, with β previously defined). Another important factor is that queries directed at files with more replicas have a higher chance of success.

Based on the previous arguments, we define the probability of query failure for a particular file as

$$q_f(\mathbf{N_a}, i) = \begin{cases} 0 & N_a^{(1)} \leq \mathcal{T}^\beta \\ 1 - \frac{\mathcal{T}}{(N_a^{(1)})^{1/\beta}} \; \frac{M+1-i}{M+1} & \text{otherwise} \end{cases}$$

If we rank the files by the their number of replicas, then $\frac{M+1-i}{M+1}$ captures the notion that a file with a higher rank has a better chance to be located. It follows that the overall probability of query failure is defined as:

$$q(\mathbf{N_a}) = \sum_{i=1}^{M} p_i \; q_f(\mathbf{N_a}, i)$$
$$= \begin{cases} 0 & N_a^{(1)} \leq \mathcal{T}^\beta \\ 1 - \frac{\mathcal{T}}{(N_a^{(1)})^{1/\beta}}(1 - \frac{K}{M+1} \sum_{i=1}^{M} i^{1-\alpha}) & \text{otherwise} \end{cases}$$

Note that in the equation above we have assumed that the rank of a file according to its request popularity is identical to its rank determined by the number of replicas it has. Although this need not be true, it allows for the analytical tractability presented in the Section V. We will relax this assumption and explore this mismatch in Section VI-C.

A. *Modeling freeloaders*

As described earlier, it has been observed that peers divide into two classes, those that provide file service capacity (non-freeloaders) and those that do not (freeloaders). It has been argued that freeloaders can have a negative impact on overall system performance. Hence, a fundamental question concerning the viability of file sharing peer-peer networks is if it can remain scalable under such a dichotomy in peer behavior. Freeloaders bring in capacity only to the common service component of the system, and do not contribute to the capacity of serving files. This represents the fact that freeloaders do not share any files but can still provide capacity to the infrastructure of the architecture (e.g., routing queries). Moreover, freeloaders generally exhibit a more aggressive behavior than non-freeloaders, in the sense that they generate more queries and download more files than non-freeloaders. Their off-line period can also be different, staying off-line for longer periods of time, although we are not aware of any measurement studies that support this possibility. We will denote the non-freeloaders and freeloaders as classes 1 and 2, respectively.

Considering our above discussion and using the functions defined earlier for the model parameters, we can redefine these functions for the two class case as:

$$\mu'_q(\mathbf{N_a}) = \mu_q(N_a^{(1)} + N_a^{(2)})$$
$$\mu'_f(\mathbf{N_a}, i) = \mu_f(N_a^{(1)}, i)$$
$$q'_f(\mathbf{N_a}, i) =$$
$$\begin{cases} 0 & N_a^{(1)} + N_a^{(2)} \leq \mathcal{T}^\beta \\ 1 - \frac{\mathcal{T}}{(N_a^{(1)}+N_a^{(2)})^{1/\beta}} \; \frac{M+1-i}{M+1} \; \frac{N_a^{(1)}}{N_a^{(1)}+N_a^{(2)}} & \text{otherwise} \end{cases}$$

for DIFA and $q'_f(\mathbf{N_a}, i) = 0$ for CIA and DIHA, as before. The aggressiveness of freeloaders is modeled by properly choosing $p_{off}^{(2)}$, $1/\lambda_{idle}^{(2)}$ and $1/\lambda_{off}^{(2)}$.

V. Solving the model

In this section we describe an approximate solution technique that will be used to numerically solve the model described above. This technique is based on bottleneck analysis [13], with an extension to handle multiple classes of customers [14].

Consider a closed queueing network consisting of a set Q of single server queues and infinite server queues, and C classes of customers. The visit ratio $V_k^{(c)}, k \in Q, c = 1, \ldots, C$ is defined as the average number of visits of customers from class c to queue k for every visit of that customer to a reference point in the network where the throughput will be measured. The service demand $D_k^{(c)}$ is defined as the product of the respective visit ratio and the average service time for class c at queue k. In this system, the utilization of each queue is proportional to its service demands. The queue with the highest service demand has the highest utilization and is called the bottleneck queue of the system. Let $R^{(c)}$ represent the average delay of a class c user to traverse the system with respect to the reference point when the total population size is $N^{(c)}$. Since a customer must spent at least the respective service time in each in each device visited, we have that $R^{(c)} \geq \sum_{k \in Q} D_k^{(c)}$. Using *Little's Law* we obtain an upper bound on the throughput for class c: $T^{(c)} \leq N^{(c)}/R^{(c)}$. However, if there exists a bottleneck device that is queueing customers, we can include the queueing delay of this device to further improve the average delay bound. Hence, we have $T^{(c)} \leq N^{(c)}/(R^{(c)} + V_{\mathcal{B}}^{(c)} \; W_{\mathcal{B}})$, where $\mathcal{B} \in Q$ is the bottleneck device of the system, and $W_{\mathcal{B}}$ is the average queueing time (time waiting in queue before starting to receive service) at the bottleneck queue. Putting these two bounds together we have:

$$T^{(c)} \leq \begin{cases} \frac{N^{(c)}}{\sum_{k \in Q} D_k^{(c)}} & \text{w/o queueing} \\ \frac{N^{(c)}}{\sum_{k \in Q} D_k^{(c)} + V_{\mathcal{B}}^{(c)} \; W_{\mathcal{B}}} & \text{w/ queueing} \end{cases}$$

The system throughput, T, is simply given by the sum of the throughputs of each class: $T = \sum_{c=1}^{C} T^{(c)}$. Moreover, when there is queueing in the system, the bottleneck device becomes the key limiting factor in the system throughput. In particular, its service capacity bounds the system throughput, and the following equation holds:

$$\sum_{c=1}^{C} T^{(c)}\ V_{\mathcal{B}}^{(c)} \leq \mu_{\mathcal{B}} \tag{3}$$

where $\mu_{\mathcal{B}}$ is the service capacity of the bottleneck device. Notice that there is no queueing for service in the infinite server queues. Using the above framework we elaborate an approximate solution for the per class throughput, $T^{(c)}$.

Reverting back to our model, we define the system throughput as the number of successful file downloads per unit time as measured at the reference point **A** illustrated in Figure 1. The visit ratio for each class for each queue of the system with respect to reference point **A** is given by:

$$\begin{aligned} V_{off}^{(c)} &= p_{off}^{(c)} \\ V_{idle}^{(c)} &= \frac{1}{1-q(\overline{\mathbf{N_a}})} \\ V_{q}^{(c)} &= \frac{1}{1-q(\overline{\mathbf{N_a}})} \\ V_{f,i}^{(c)} &= \frac{p_i\ (1-q_f(\overline{\mathbf{N_a}},i))}{1-q(\overline{\mathbf{N_a}})} \end{aligned}$$

where $V_{off}^{(c)}$ is the probability of going off-line; $V_{idle}^{(c)}$ and $V_q^{(c)}$ reflects the number of times on average a peer revisits the idle component and the common service queue, respectively, due to query failures; $V_{f,i}^{(c)}$ is the frequency of visiting the file service component for file i; and $\overline{\mathbf{N_a}} = (\overline{N_a^{(1)}}, \ldots, \overline{N_a^{(C)}})$, is the expected number of peers of each class on-line in the system. Note that with the exception of $V_{off}^{(c)}$, all other visit ratios are the same among different classes of peers.

Thus the corresponding demands for each part of the system are:

$$\begin{aligned} D_{off}^{(c)} &= \frac{p_{off}^{(c)}}{\lambda_{off}^{(c)}} \\ D_{idle}^{(c)} &= \frac{1}{\lambda_{idle}^{(c)}\ (1-q(\overline{\mathbf{N_a}}))} \\ D_{q}^{(c)} &= \frac{1}{\mu_q(\overline{\mathbf{N_a}})\ (1-q(\overline{\mathbf{N_a}}))} \\ D_{f,i}^{(c)} &= \frac{p_i\ (1-q_f(\overline{\mathbf{N_a}},i))}{\mu_f(\overline{\mathbf{N_a}},i)\ (1-q(\overline{\mathbf{N_a}}))} \end{aligned}$$

and the system throughput is bounded by

$$T^{(c)} \leq \begin{cases} \dfrac{N^{(c)}}{D_{idle}+D_q+\sum_{i=1}^{M} D_{f,i}+D_{off}} & \text{w/o queueing} \\ \dfrac{N^{(c)}}{D_{idle}+D_q+\sum_{i=1}^{M} D_{f,i}+D_{off}+V_{\mathcal{B}}^{(c)}\ W_{\mathcal{B}}} & \text{w/ queueing} \end{cases} \tag{4}$$

By applying Little's Law at the infinite server queue that represents the off-line period of a peer in the system, we have

$$(N^{(c)} - \overline{N_a^{(c)}})\ \lambda_{off}^{(c)} = T^{(c)}\ p_{off}^{(c)}$$

where $N^{(c)} - \overline{N_a^{(c)}}$ is the average off-line population size of class c, $T^{(c)}\ p_{off}^{(c)}$ is the rate at which peers from class c go off-line, and $1/\lambda_{off}^{(c)}$ is the service rate of the off-line infinite server queue.

This leads to:

$$\overline{N_a^{(c)}} = N^{(c)} - \frac{T^{(c)}\ p_{off}^{(c)}}{\lambda_{off}^{(c)}} \tag{5}$$

We will assume that the throughput of the system actually equals the upper bound established by equations (3) and (4). This assumption combined with equation (5) provides us with $2C+1$ non-linear equations with $\overline{N_a^{(c)}}$, $T^{(c)}$ and $W_{\mathcal{B}}$ as unknowns. We solve the resulting fixed point problem numerically using an iterative method. We believe it can be shown that this fixed point problem always has a unique solution [10]. As we will see in the next section, the results obtained with this approximate numerical solution agrees very well with exact results obtained from simulating the model.

Another classical metric usually studied to quantify performance is the average response time of the system. In our model, the average response time of an on-line peer would be a combination of the average delay in processing the query (common services component) and the average delay in downloading the file (file service component). In our closed queuing network, we can express the average response time of a class c peer, $R_{time}^{(c)}$ as:

$$R_{time}^{(c)} = \frac{N^{(c)}}{T^{(c)}} - \frac{p_{off}^{(c)}}{\lambda_{off}^{(c)}} - \frac{1}{\lambda_{idle}^{(c)}\ (1-q(\overline{\mathbf{N_a}}))}$$

Here $N^{(c)}/T^{(c)}$ is the average delay experienced by a user to traverse a network. To obtain the average user response time, we subtract the average time spent off-line and the time spent in the idle component from this value.

From this expression, we note that for a system with a fixed population, $\mathbf{N}$, a higher throughput implies a lower average response time. Hence we present our results simply in terms of the system throughput, T.

VI. Model Results

The model coupled with an efficient solution method provides us with the ability to understand and explore how the system performs under various conditions. Different fundamental "what-if" questions can be answered by simply varying system parameters. We now address some of these questions, presenting results obtained with the approximate analytical method described in the previous section.

In order to obtain numerical solutions we must specify the value of all system parameters and this can directly influence the magnitude of the results. We explain the choice of numerical values for some of these parameters based on our

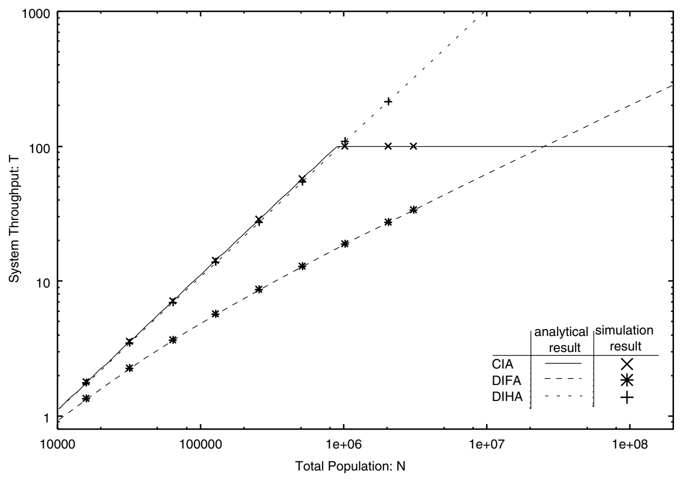


Fig. 2. System throughput for different architectures with varying population size

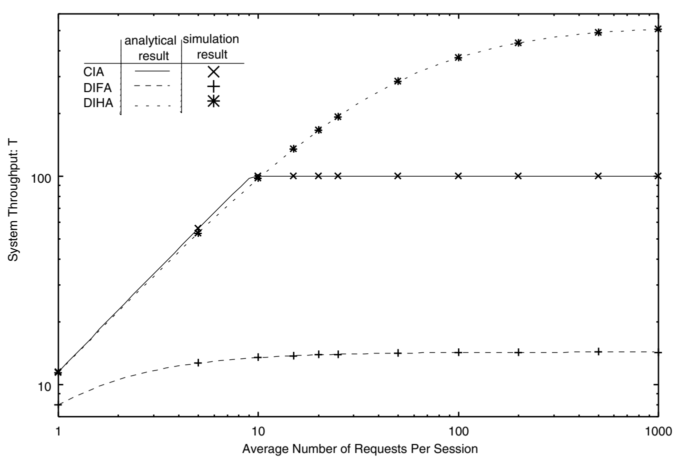


Fig. 3. System throughput for different architectures with varying average number of requests per session ($N = 500,000$)

understanding of realistic peer-peer networks. For example, the basic service rate associated with the contribution of a single peer to a file service capacity, H, is set to 1/20. This corresponds to a file download time of apporximately 150 seconds for the most popular file. This is reasonable, since the mean file size could be 3.5Mb and the access link of a peer could have 200Kbps. We choose the capacity of the central server to perform query lookups (C_1) to be 100 requests per second. This is two orders of magnitude larger than the capacity of a single peer to process a query, which we set to $C_q = 1$. We choose the TTL value for the DIFA system to be $\mathcal{T} = 7$, since this is the default value used in applications such as Gnutella [5]. Other parameteres are (time unit in seconds): $M = 1000$, $\theta = 0.1$ (DIHA), $\beta = 2$, $\alpha = 1$, $\lambda_{idle}^{(1)} = 1/300$, $\lambda_{idle}^{(2)} = 1/30$, $p_{off}^{(1)} = 0.2$, $p_{off}^{(2)} = 0.1$, $\lambda_{off}^{(1)} = \lambda_{off}^{(2)} = 1/43200$.

We start by comparing the scalability of the three different architectures under a single class of peers and determining the system throughput, under scenarios of varying load and system capacity. Figure 2 plots the system throughput T versus the total number of peers in the system N. Given the characteristics of a peer-peer networks, an increase in the population corresponds to an increase in both the system workload and capacity.

We observe that the CIA architecture outperforms DIFA and is slightly better than DIHA when the population size is small. This is primarily due to the higher capacity of the centralized query lookup. This trend persists until the central server in CIA becomes the bottleneck. At higher population sizes, DIFA and DIHA perform better than CIA, since the serving capacity in these systems *scales* with the increase in the population. DIFA however suffers from another drawback: the probability of query failure increases with the population size. This occurs since queries can only reach a bounded number of peers which is independent of the population size, which implies that queries only search over a bounded subset of the index space. Thus, DIFA cannot capitalize on the potential capacity of peer-peer systems, as queries that could have succeeded fail. This limits the effective throughput of a DIFA architecture, resulting in a lower performance than DIHA in terms of system throughput.

In Figure 3, we study the impact of increasing the workload of the system, by increasing the average number of file downloads a user performs during an on-line period. We fix the population size to 500,000 peers and examine the system throughput. Since the population is fixed, this has the effect of increasing the load in the system without a corresponding increase in service capacity. We observe that all architectures reach a bottleneck point at a certain level of activity. This bottleneck corresponds to the capacity of the system to download files. Once again, the DIHA architecture scales best, since it is not constrained by a central server and there are no query failures as in DIFA.

The approximate solution method has been experimentally validated by simulating the same analytical model. In the simulation we assume that service times of both single and infinite server queues are exponentially distributed. Our simulation results are in close agreement with the approximate solution method, as illustrated in Figures 2 and 3 (crosses indicate simulation results).

Further validation of the model results, however, is not an easy task. Existing measurement studies on realistic file sharing peer-peer applications have not focused on characterizing system wide performance (e.g. system throughtput). Also, even if such studies are conducted, the results would be for an instance of such system, and would not yield any insights into how the system performs upon scaling different parameters, such as population size, peer behavior, etc. Although it might be possible to validate our model through detailed simulations of realistic peer-peer applications, the programming and computational cost would be prohibitive.

A. *Impact of freeloaders*

We now consider a scenario where there are two classes of peers, freeloaders and non-freeloaders. Freeloaders are as-

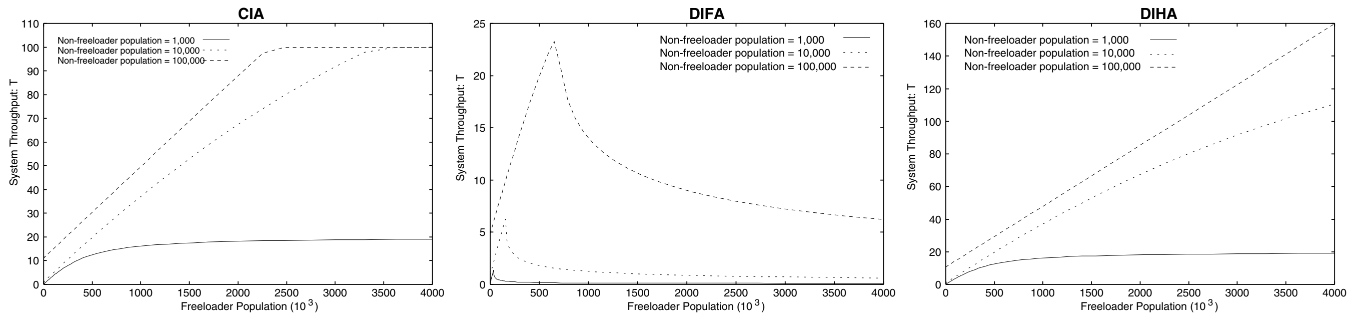


Fig. 4. Total system throughput with varying population of freeloaders and a fixed number of non-freeloaders for (a) CIA (b) DIFA and (c) DIHA

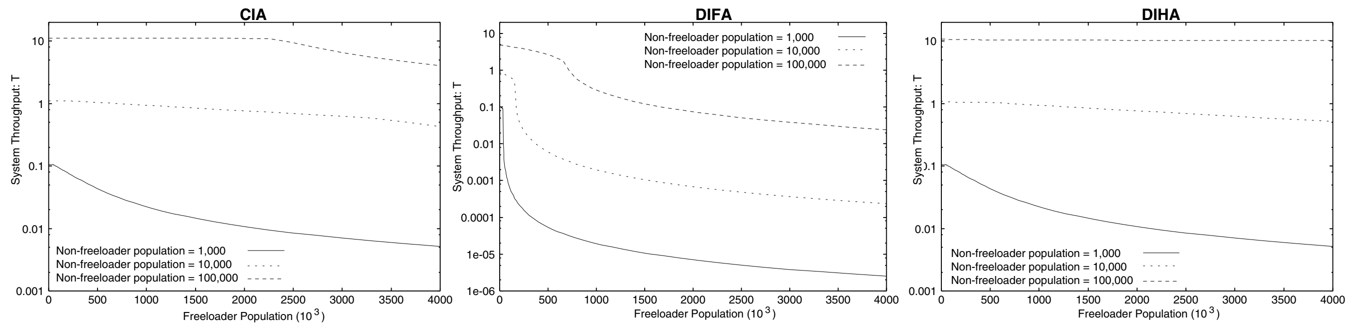


Fig. 5. Throughput of the non-freeloader class with varying population of freeloaders and a fixed number of non-freeloaders for (a) CIA (b) DIFA and (c) DIHA

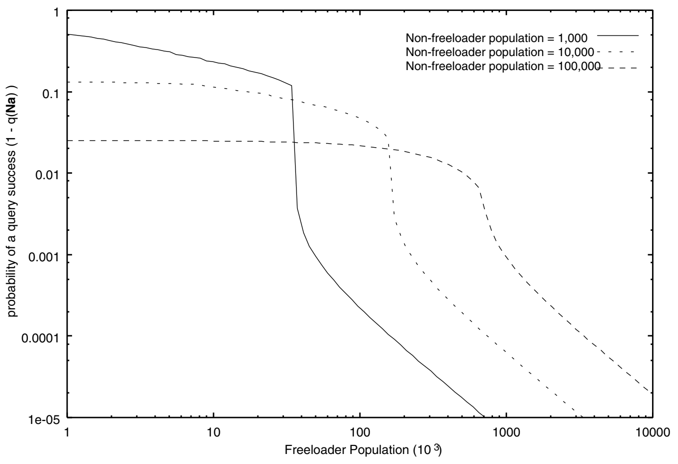


Fig. 6. Probability of a query succeeding in locating a file with vary population of freeloaders and a fixed number of non-freeloaders for DIFA

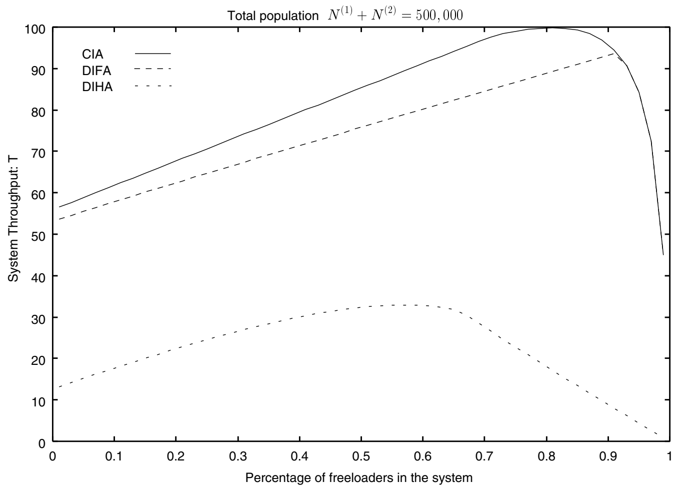


Fig. 7. Total system throughput with a varying fraction of freeloaders with a total population of 500,000 peers, for CIA, DIFA and DIHA

sumed to be more aggressive than non-freeloaders. To capture this behavior we let the think time of freeloaders be 10 times smaller (30 seconds as compared to 5 minutes) than that of the non-freeloaders, double the average number of downloads a freeloader performs before going off-line (from 5 to 10), and keep the average off-line time identical for both classes (12 hours).

Figure 4 plots the system throughput as a function of the number of freeloaders while maintaining a constant number of non-freeloaders. Our first observation is that given a reasonable number of serving nodes, both CIA and DIHA scale well. Thus, a small number of serving nodes can support a large peer-peer network. Note that from an overall system perspective, having freeloaders actually increases the throughput, since they bring more load to the system, which was initially under-utilized. However, this does not hold true for DIFA, since in this architecture, a query propagates only over a bounded number of peers. Thus, increasing the freeloader population while keeping the number of non-freeloaders fixed, makes it more difficult for a query to locate a file. This behavior is captured in the definition of $q'_f(\mathbf{N_a}, i)$. In any case, the overall system throughput increases (even in DIFA, for a small population of freeloaders) suggesting that peer-peer file sharing systems have spare capacity that freeloaders can benefit from.

One might think that even though overall system performance scales well with an increase in the number of freeloaders, the throughput of the non-freeloaders might decrease. Although this is true, the decrease in the throughput of non-freeloaders is very small for the CIA and DIHA architectures, even for a large number of freeloaders, as illustrated in Figure 5. This suggests that freeloaders can benefit from available capacity in the system without significantly degrading the performance of the non-freeloaders. However, the non-freeloaders in DIFA suffer more due to the increased query failure probability, as discussed below.

Figure 6 illustrates the probability of a query succeeding in locating a file $(1 - q'(\mathbf{N_a}))$ for varying freeloader population. We observe a sharp drop in the success probability for all curves plotted. An increase in the non-freeloader population causes an increase in the number of queries that fail. Failed queries generate reattempts which further add to the system workload. This creates a vicious cycle with consistently degrading system performance, which becomes markedly evident at the sharp drop point in the query success probability. This illustrates that DIFA can tolerate some number of freeloaders without much degradation in the query success probability, but that performance drops sharply if this number is too large.

We now examine the impact of freeloaders in a slightly different manner shown in Figure 7. We compute the system throughput with an increasing percentage of freeloaders in the system for a fixed overall population of 500,000 peers. Once again, we initially observe an increase in throughput with an increase in the ratio of freeloaders for all three architectures. However, above a certain ratio the system throughput starts to degrade. Note that CIA and DIHA can support a much larger ratio of freeloaders than DIFA. Again, this rapid degradation for DIFA is mainly due to the increase in the query failure probability. This result shows that freeloaders can take advantage of the spare capacity in the system, but that this spare capacity is limited and saturates at a given ratio.

B. Exploring TTL in DIFA

A crucial aspect of the DIFA architecture is the value used for $\mathcal{T}$ (TTL), which can impact system performance. Increasing $\mathcal{T}$ reduces the probability of query failure, at the cost of increasing the load on the system by having more peers process a given query. This trade-off is illustrated in Figure 8, which plots the system throughput as a function of $\mathcal{T}$ for different values of β. Note that β is a parameter related to network connectivity, and determines the number of peers reachable in a query for a given TTL ($\mathcal{T}^{\beta}$). We observe an optimal value for TTL that maximizes the system throughput in all curves. Initially, the system throughput is small since queries frequently fail due to small TTL values. Increasing TTL too much can reduce the system throughput as queries

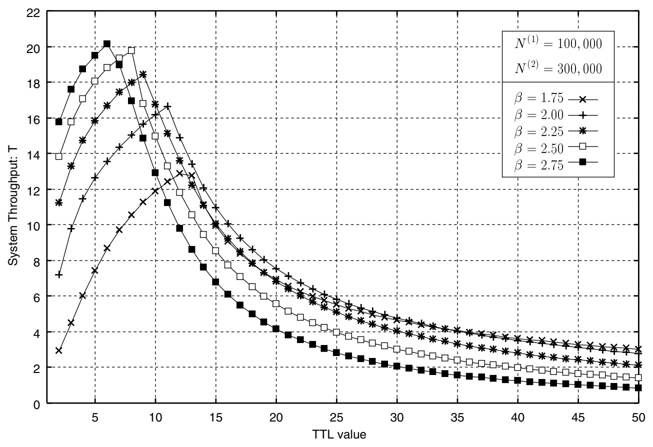


Fig. 8. System throughput for different values of β with varying TTL for DIFA architecture

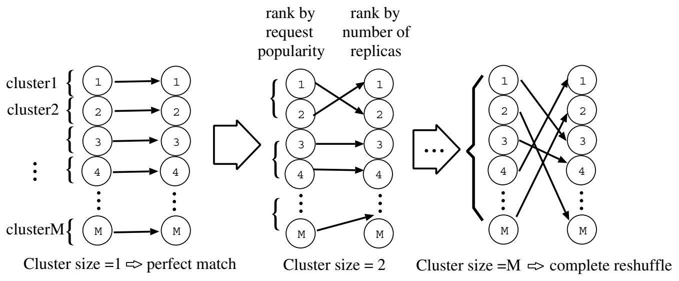
now pose a larger workload to the system. Note that increasing β reduces the optimal value for TTL and increases the system throughput. A higher value of β implies that the same number of peers can be reached with a smaller TTL. Note that for a fixed small value of TTL (< 6), the system throughput is larger for larger values of β. However, for fixed large values of TTL (> 35), the larger β pays higher cost for processing queries, delivering a lower throughput.

C. Mismatch in file popularity and replication

Until now, we have assumed that the distribution of file requests and that of file replication (or availability) are the same, i.e. the most popular file is also the most replicated. This intuition is justifiable, since the most popular content is also in all likelihood downloaded more often, and hence has a higher chance to be made available as a replica to other users.

However, this may not always be true. For instance, a newly released popular file may take some time to become well replicated. Similarly, a very popular and also well replicated file might slowly lose its popularity and be phased out of the system.

As a consequence, there can exist an imbalance between the capacity to serve a file and the number of requests this file receives. In order to study this behavior, we will introduce a mismatch in the file ranks, such that the i-th most requested file is **not** the same as the i-th most replicated file.



rank by request popularity
rank by number of replicas
cluster1
cluster2
clusterM
Cluster size =1 ⇨ perfect match
Cluster size = 2
Cluster size =M ⇨ complete reshuffle

Fig. 9. Clustered shuffling to create mismatch between file ranks

Figure 9 illustrates the procedure to create a mismatch in the file ranks. As described, each file is assigned two ranks; one according to its request popularity and the other based on the number of replicas. We start by assuming that the ranks of the files have a perfect matching. Then, we group files with successive rank into clusters of fixed size. The size of the cluster is a parameter and determines the degree of mismatch between the ranks. Inside a cluster, the ranks are shuffled. Note that a cluster size of one is identical to having a perfect match in the ranks. Increasing the size of the cluster increases the imbalance in the mapping. With a cluster size of M, the two file ranks become independent of each other.

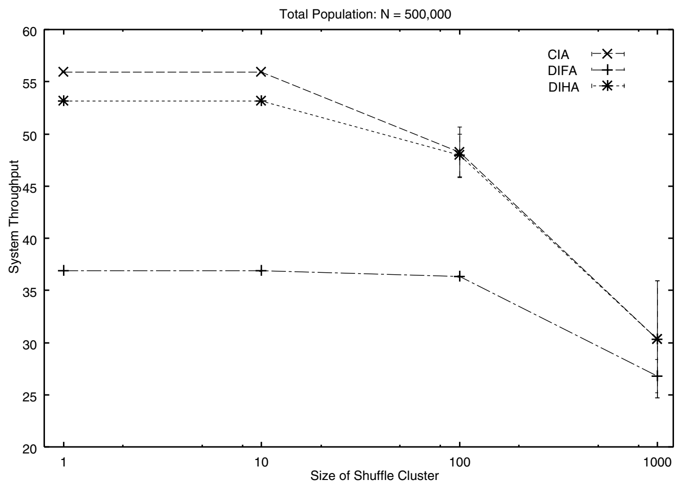


Fig. 10. Throughput with varying levels of mismatch between file popularity and replication

Model simulation is used to investigate the performance of the unbalanced system. In Figure 10 we observe, as expected, that the system throughput drops if the most popular files are not sufficiently replicated. Interestingly, this degradation in the system performance is not noticeable until the cluster size significantly becomes large, indicating a large mismatch.

VII. Conclusion

Peer-peer systems present an interesting and inherently different resource allocation problem than traditional client/server models. The work in this paper is motivated by the belief that simple models can provide important insights into understanding the behavior of such systems. In this paper, we have presented a mathematical model that captures the main characteristics of peer-peer file sharing systems. A strength of our approach is the generality and the flexibility of the framework used, which allow us to both capture the essence of different architectures and peer behavior, and explore a range of performance trade-off issues. Moreover, as discussed in Section III, the model can be easily extended to address other system characteristics not discussed in this paper.

Using an approximate solution method we have numerically analyzed a few fundamental performance issues of different architectures under different scenarios. Our results highlight interesting issues, such as the inherent limitation of DIFA due to its query mechanism, and the available spare capacity that freeloaders can benefit from without significantly degrading performance of non-freeloaders.

References

[1] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall, 2001.

[2] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of ACM SIGCOMM'01*, Aug 2001.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. of ACM SIGCOMM'01*, Aug 2001.

[4] "Napster protocol specification," March 2001, http://opennap.sourceforge.net/napster.txt.

[5] Clip2, "The gnutella protocol specification v0.4," 2000, http://www.clip2.com/GnutellaProtocol04.pdf.

[6] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware*, 2001, pp. 329–350.

[7] Y. Zhao, J. D. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," U. C. Berkeley," USB//CSD-01-1141, April 2000.

[8] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5, no. 10, October 2000.

[9] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[10] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, "Modeling peer-peer file sharing systems," University of Massachusetts, Dept. of Computer Science, Tech. Rep. CMPSCI 02-27, 2002.

[11] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, "An analysis of internet content delivery systems," in *Proc. of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[12] J. Chu, K. Labonte, and B. N. Levine, "Availability and locality measurements of peer-to-peer file sharing systems," in *Proc. of SPIE ITCom: Scalability and Traffic Control in IP Networks*, vol. 4868, July 2002.

[13] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[14] D. L. Eager and K. C. Sevcik, "Bound hierarchies for multiple-class queuing networks," *Journal of the ACM*, vol. 33, no. 1, pp. 179 – 206, 1986.