THE UNIVERSITY OF CHICAGO

RESOURCE DISCOVERY IN LARGE RESOURCE-SHARING ENVIRONMENTS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY

ADRIANA IOANA IAMNITCHI

CHICAGO, ILLINOIS

DECEMBER 2003

ABSTRACT

Opportunistic sharing of Internet-connected resources is a low cost method for obtaining access to unprecedented-scale collections of resources. An essential service in any resource-sharing environment is resource discovery: given a description of the resources desired, a resource discovery mechanism returns locations of resources that match the description.

Two resource-sharing environments are particularly well defined by applications, user communities, and deployments: Grid and peer-to-peer systems. Grids are sharing environments that rely on persistent, standards-based service infrastructures that allow well-established, mainly professional communities to share computers, storage space, sensors, software applications, and data across organizational boundaries. Peer-to-peer systems are Internet applications that harness resources from millions of autonomous participants. Thus, Grids provide infrastructure to support a variety of applications on resources shared by relatively small communities; at the scale of the peer-to-peer communities, remarkable sharing patterns are exhibited, such as free riding and intermittent resource participation.

The focus of this dissertation is on solution design for resource discovery in Grids of the scale and lack of reliability of today's peer-to-peer networks. This hybrid target environment requires fully decentralized solutions that scale with the number of users and resources and tolerate intermittent resource participation.

To explore the solution space, we propose a taxonomy for resource discovery solutions. This taxonomy proves to be a useful tool for discussing and comparing existing solutions.

Using this taxonomy, we delimit and explore a portion of the solution space. We build a scalable Grid emulator to evaluate mechanism performance in this subspace. Large-scale experiments reveal that the performance of mechanisms in this subspace is strongly dependent on sharing characteristics.

For inspiration, we turned to studying user behavior in various communities. We uncovered a significant usage pattern in file-sharing communities: users naturally form interest-based groups. This pattern can be exploited for system design in a variety of problems: we designed a file-location mechanism, FLASK, that exploits and benefits from this naturally emerging pattern. Trace-driven evaluations show FLASK leads to lower response latency, good scalability, support for intermittent participation, and satisfies requirements typical of scientific usage of data.

ACKNOWLEDGEMENTS

I am indebted to my advisor, Ian Foster, for 6 years of peer-to-peer collaboration: Ian always encouraged me to follow my own interests, under the motto "What's the point in doing it if it's not fun?" He often reminded me to look at the big picture. The big picture now shows that an advisor has many students during his career but is condemned to remain unique for each of his students.

With Janos Simon I had uncountably many conversations during these years: our random walks between graph theory and the addressing forms in the Brazilian-Portuguese provided many ideas incorporated in this thesis. Bruno Codenotti's insights helped shape this thesis and still provide good material for future work.

I am indebted for the access to the D0 data to a team of scientists from Fermi National Accelerator Laboratory: Lee Lueking first suggested this opportunity at SuperComputing 2001; Ruth Pordes, Gabriele Garzoglio, and John Weigand helped materializing it. I owe the access to Kazaa data to Nathaniel Leibowitz.

Parts of this thesis were presented in various meetings and conferences and improved significantly thanks to feedback from the audience: I thank now to the many people I cannot name. Discussions "on the road" were loaded with ideas and suggestions from László Barabási, Stephen Eubank, Christos Papadimitriou, Karl Czaikovski, Miron Livny, Zoltan Toroczkai, and Gabriel Istrate.

To Leo Irakliotis I owe the opportunity to teach in the Masters Program he leads. His humor and philosophy of life made me feel at home. Not less important, his talents in setting up the projector for my dissertation defense deserves a warm $\epsilon\upsilon\chi\alpha\rho\iota\sigma\tau\omega$!

To Irina Athanasiu and Valentin Cristea, my professors from Politehnica University of Bucharest, Romania, I owe my interest for distributed systems. Gail Pieper edited many of my writings during the years. She always took the time to write detailed explanations of the writing rules that I repeatedly got wrong.

I often ran to my friends Radu, Andrei and Livache for help with statistics and math questions. To have them, Raluca, and Lei around was one of the joys of living in Chicago. Kavitha proved a constant friend and good colleague during the years. Always online for my silly questions, I gradually came to rely on Stephen for everything not explained to my understanding in the Encyclopaedia Britannica, the Oxford Dictionary, or by Google. He also proofread many of my papers and a significant part of this thesis. To Anca and Andi I owe the wonderful time we spent in Southern France and so much more.

Matei asked to be acknowledged for his perseverance and dedication at washing dishes during his five years in Chicago. Without minimizing the importance and the influence of his dish-washing skills on my research, his help and support manifested in many other ways. Matei acted as the most constant (and the toughest) critique of my work; he gave feedback to all my papers; he was always careful to make me break my “suicidal regime”, as he called the periods of intense work. We worked together on many projects and papers, of which many ideas are found in this thesis.

I owe the peace of mind that made me finish this thesis to some wonderful people who helped my father recover after a stroke: Dr. Tatiana Luchian made miracles. Our wonderful friend Nora Mihoc did her share of impossible. My mother managed the situation like only mothers can and provided invaluable support. With my brother Marius I shared the joys and the worries during all these years. And all these would have had no effect without my father’s optimism and perseverance.

For all I am, I am indebted to my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

An important consequence of the ubiquitous Internet is the opportunity to share with total strangers not only communication infrastructure but also other resources, such as computers and data. The classic example is Seti@home [8]: a project from Berkeley that benefits from computational power of computers on the Island of Man, Tonga, and Saint Pierre and Miquelon. Another example is the peer-to-peer, or P2P, music-sharing networks such as Napster [103], Kazaa [71], or Gnutella [98]: a file provided by a peer in Guam is downloaded by a user in Sicily, this operation being made possible by many other peers from, for instance, Hawaii, Mexico and Portugal.

Sharing resources over the Internet raises many technical problems, amplified by the potential scale of the resource pool, the heterogeneity of platforms, the diversity in user behavior, and the inherent lack of reliability. Sharing resources over the Internet also has many names and objectives: the Web is a form of resource sharing, but so are Grid, global computing, P2P, or utility computing, to name just a few. In all these systems, one important functionality is resource discovery: given a resource description, the resource discovery component returns the location of one or more resources that fit the description. However, the resource discovery problem has slightly different meanings and requirements depending on the particular resource-sharing environment and, therefore, its objectives and constraints.

Two resource-sharing environments have particularly well defined philosophies, communities, and incarnations: Grids and P2P systems. Grids [42, 43, 41] provide the software infrastructure to federate resources from multiple administrative domains in support of a variety of data- and compute-intensive applications. They have grown incrementally to provide quite sophisticated services to a mainly scientific community. Their objective is to support computations on an unprecedented scale using collections of resources aggregated dynamically, while providing security guarantees and accommodating resource usage policies dictated by owners.

P2P systems are Internet applications that harness the resources of a large number of autonomous participants. They have emerged as vertically integrated popular applications such as music-sharing and grew tremendously in user-base over a short time. The scale of current P2P systems makes them an interesting case study: a community with millions of users gives significant insight into user behavior, heterogeneity, reliability, design bottlenecks, and so on.

This thesis focuses on resource discovery in Grid environments of the scale and lack of reliability of today's P2P networks. The characteristics of the target environment requires solutions that are fully decentralized, scalable with the number of users and resources, and tolerant to intermittent resource participations (whether voluntary or due to failures). To define the solution space for resource discovery mechanisms, we propose a taxonomy. This taxonomy is a useful tool for discussing previous work but does not reduce the size of the (large) solution space. We get guidance for mechanism design by examining user behavior in different file-sharing communities. We discover an important pattern in file usage: users naturally cluster in interest-based groups. This observation guides the solution design for an instance of the resource discovery problem, namely file location. Important lessons for resource discovery are revealed by the design and evaluation of this file-location mechanism.

In the remainder of this chapter we introduce the resource discovery problem, discuss the characteristics of the target resource-sharing environment, identify a set of requirements for resource discovery solutions, and present the roadmap of this thesis.

1.1 The Resource Discovery Problem

When dealing with large sets of entities, a basic problem is locating one or more entities that match a given description. These descriptions can have various forms, from globally unique identifiers (such as a person's name and address) to enumerations of desired attributes, such as the required skills of a qualified candidate presented in job advertisements.

As already suggested, everyday life is awash with instances of resource discovery: a lookup in the Yellow Pages for a cab phone number; a Google search; posting an advertisement in a newspaper; browsing the Web by specifying the desired URL. Given the characteristics of each instance, different solutions are employed: centralized indexes (as in telephone books or Google), targeted broadcast (as in newspaper advertisements), or mechanisms that associate human-readable addresses with machine-understandable locations (as in DNS).

We study the resource discovery problem in a resource-sharing environment that preserves the essence of Grids but has the scale and dynamism of peer-to-peer communities. While the two environments have the same final objective—to pool large sets of resources—they emerged from different communities, and hence their current designs highlight different requirements. We believe that the design objectives of the two environments will eventually converge. Consequently, it is important to analyze, compare, and contrast their current requirements and characteristics.

To this end, we discuss the resource location problem in the context of the two resource-sharing environments (Section 1.2) and we identify four critical design objectives for resource discovery (Section 1.3).

1.2 Grid and Peer-to-Peer Environments

In recent years significant interest has focused on two resource-sharing environments: Grids and peer-to-peer (P2P) systems. The two systems have followed different evolutionary paths. Grids have incrementally scaled the deployment of relatively sophisticated services, connecting small numbers of sites into collaborations engaged in complex scientific applications. P2P communities, on the other hand, have developed rapidly around unsophisticated but popular services such as file sharing, focusing on scalability and support for intermittent user and resource participation. As a result of these different evolutionary paths, the two systems differ in three dimensions: target communities, resources, and applications. We discuss each of these dimensions below.

Despite these differences, however, we maintain that the two environments are in fact concerned with the same general problem, namely, resource sharing within *virtual organizations* that link resources and people spanning multiple physical organizations. Moreover, the two environments seem likely to converge [40] in terms of their concerns, as Grids scale and P2P systems address more sophisticated application requirements.

1.2.1 Target Communities and Incentives

The development and deployment of Grid technologies were motivated initially by the requirements of professional communities to access remote resources, federate datasets, and/or to pool computers for large-scale simulations [7] and data analyses [52, 39]. Participants in contemporary Grids are members of established communities that are prepared to devote effort to the creation and operation of required infrastructure and within which exist some degree of trust, accountability, and opportunities for sanctions in response to inappropriate behavior.

In contrast, P2P has been popularized by grass-roots, mass culture (music) file-sharing and highly parallel computing applications [8, 9] that scale in some instances to hundreds of thousands of nodes. The “communities” that underlie these applications comprise diverse and anonymous individuals with little incentive to act cooperatively and honestly. Thus, for example, we find that in file-sharing applications, there are few providers and many consumers [4]; the operators of SETI@home [105] devote significant effort to detecting deliberately submitted incorrect results; and people tend to intentionally misreport their resources [103].

The two target communities differ in scale, homogeneity, and the intrinsic degree of trust. The natural tendency of Grids to grow, however, will inevitably lead to less homogeneous communities and, consequently, to smaller degrees of trust. Participation patterns will also change with scale: intermittent participation is likely to become the norm. All these characteristics have a strong impact on defining the assumptions one can make (or, rather, cannot) about the environment. We need support for volatile user communities; and we need to deal with the lack of incentives for and interest in centralized, global administrative control.

1.2.2 Resources

In general, Grid systems integrate resources that are more powerful, more diverse, and better connected than the "desktop at the edge of the Internet" [106] that constitutes a typical P2P resource. A Grid resource might be a cluster, storage system, database, or scientific instrument of considerable value that is administered in an organized fashion according to some well-defined policy. This explicit administration enhances the resource's ability to deliver desired qualities of service and can facilitate improvements to services (e.g., through software upgrades), but it can also increase the cost of integrating the resource into a Grid. Explicit administration, higher cost of membership, and stronger community links within scientific virtual organizations mean that resource availability tends to be high and uniform.

In contrast, P2P systems deal with intermittent participation and highly variable behavior. For example, one study of Mojo Nation [121] showed that a node remained connected on average for about 28% of time. Moreover, the connection time distribution was highly skewed, with one sixth of the nodes remaining always connected.

Grids integrate not only "high-end" resources: desktop systems with variable availability [80] form a major component of many contemporary Grids [12]. However, the ensemble of all such resources within a Grid are not treated as an undifferentiated swarm of global scope. Rather, they are aggregated within administrative domains via technologies such as Condor [74, 75] to create local resource pools that are integrated into larger Grids via the same Grid protocols as other computational resources.

Resources in Grids, traditionally from research and educational organizations, tend to be more powerful than home computers that arguably represent the majority of P2P resources (e.g., 71% of SETI@home systems are home computers [105]). The difference in capabilities between home and work computers is illustrated by the average CPU time per work unit in SETI@home: home computers are 30% slower than work computers (13:45 vs. 10:16 hours per work unit).

As their size increases, Grids will increasingly adopt some of the characteristics of today's P2P systems in resource participation: unreliable resources and intermittent participation will constitute a significant share. At the same time, Grid resources will preserve or increase their diversity. Consequently, services—and resource discovery in particular—will have to tolerate failures and adapt to dynamic resource participation.

1.2.3 Applications

The range and scope of scientific Grid applications vary considerably. Three examples that show the variety of deployed Grid applications are the HotPage portal, providing remote access to supercomputer hardware and software [113]; the numerical solution of the long-open `nug30` quadratic optimization problem using hundreds of computers at many sites [11]; and the NEESgrid system that integrates earthquake engineering facilities into a national laboratory [90]. Grid applications typically share a common infrastructure (such as the Globus Toolkit [48]) that provides basic services such as security and resource management.

In contrast, P2P systems tend to be vertically integrated solutions to specialized problems: currently deployed systems share either compute cycles or files. Diversification comes from differing design goals, such as scalability [95, 112, 101], anonymity [28], or availability [28, 66].

Grid applications also tend to be far more data intensive than P2P applications. For example, a recent analysis of Sloan Digital Sky Survey data [10] involved, on average, 660 MB input data per CPU hour; and the Compact Muon Solenoid [81] data analysis pipeline involves from 60 MB to 72 GB input data per CPU hour. In contrast, SETI@home moves at least four orders of magnitude less data: a mere 21 KB data per CPU hour. The reason is presumably, in part at least, better network connectivity, which also allows for more flexibility in Grid application design: in addition to loosely coupled applications [2, 10], Grids have been used for numerical simulation [7, 99, 100] and branch-and-bound-based optimization problems [11].

The variety of Grid applications requires significant support from services. Applications may use not only data and computational power, but also storage, network bandwidth, and Internet-connected instruments at the same time. Unlike in file-sharing systems such as Gnutella, this variety of resources requires attribute-based identification (such as "Linux machine with more than 1 GB memory"), since globally unique names (such as "ficus.cs.uchicago.edu") are of no significant use. Also, Grid services must provide stronger quality-of-service guarantees: a scientist that runs data-analysis applications in a Grid may be less willing to wait until data is retrieved than is a typical P2P user in search for music files.

1.3 Requirements for Resource Discovery

As we have noted, we expect Grid and P2P systems to converge in a unified resource-sharing environment. This environment is likely to scale to millions of resources shared by hundreds of thousands of participants (institutions and individuals): No central, global authority will have the means, the incentive, and the participants' trust to administer such a large collection of distributed resources. Participation patterns will be highly variable: there will be perhaps a larger number of stable nodes than in today's P2P systems, but many resources will join and leave the network frequently. Resources will have highly diverse types (computers, data, services, instruments, storage space) and characteristics (e.g., operating systems, number of CPUs and speed, data of various sizes, services). Some resources will be shared following well-defined public policies, such as "available to all from 6 pm to 6 am." Other resources will participate rather chaotically, for example, when idle. Technical support will be variable: some participants will benefit from technical support, whereas others will rely on basic tools provided by the community (e.g., today's Gnutella nodes run various implementations of the protocol, each with its own peculiarities).

In addition to the requirements imposed by the resource-sharing environment, various challenges are raised by usage scenarios. Resource discovery may be expected to provide browsing capabilities: users may want to learn what resources are shared in a virtual organization, maybe to decide whether to join it with their own resources. Also, resource discovery may be employed to support other mechanisms, such as scheduling or monitoring services: when jobs are submitted, a scheduler may call resource discovery to get up-to-date information about available, appropriate resources. A monitoring service may want to detect all existing replicas of a file in order to decide whether more replicas are needed for improving performance or availability. Another challenge comes from the diversity of requests, from simple requests that specify one resource, such as a file named "foo," to aggregated requests, such as 100 Linux machines separated by at least 20ms. latency pairwise.

However, the design of a mechanism that meets the desired usage scenarios must obey the rules imposed by the characteristics of the resource-sharing environment. These characteristics require a resource discovery mechanism with the following features:

- **Independence from central, global control.** This is a departure from previous Grid solutions and a step toward the fully decentralized solutions typical of P2P approaches.
- **Support for attribute-based search**, a feature not found in current P2P solutions. That is, solutions must support requests that specify a set of desired attributes ("Linux machine with more than 128 MB of available memory") rather than be designed around globally unique identifiers (such as "ficus.cs.uchicago.edu"). Attribute-based search is even more challenging when resource attributes can vary over time (e.g., CPU load, available bandwidth, even software versions) and when the number of resources is large and/or dynamic (as resources join, leave, or fail).
- **Scalability**, which becomes more important with increased scale and participation.
- **Support for intermittent resource participation**, a characteristic frequent in today's P2P systems but rare in current Grid solutions.

1.4 Thesis Roadmap

The central focus of this thesis is the design of fully decentralized resource discovery solutions for large-scale Grid environments in which resource and user participation is highly variable. Along the way, we define the solution space by proposing a taxonomy for resource discovery mechanisms; we design and implement a scalable Grid emulator; we evaluate a set of resource discovery strategies in emulated Grids; we identify usage patterns in file-sharing communities; we design a mechanism for an instance of resource discovery, namely file location; and we apply and adapt the lessons learned from the solution to the instance to the general problem. Accordingly, the remainder of this thesis is organized as follows:

Chapter 2: Resource Discovery in Distributed Systems

Chapter 2 proposes a general framework for any resource discovery solution. This framework, based on four components, defines the design space and provides the basis for comparing existing solutions. We prove the generality of this framework by mapping previous resource discovery solutions onto these four components. We also investigate experimentally a number of design solutions for one of these components in a large-scale emulated Grid.

Chapter 3: Small-World File-Sharing Communities

The four-component framework provides a common language for describing and comparing various design alternatives, but does not reduce the size of the (large) solution space. One approach to designing efficient solutions is to analyze and possibly exploit user behavior. Although Grids are the most mature deployment of computer-sharing environments, their deployment is currently limited in scale and usage: many Grids are still in experimental phases. Thus, a serious problem in Grid research is the lack of a large and diverse user community on non-experimental Grids: this fact makes the availability of user traces an intractable problem.

However, there is one instance of resource sharing, namely file sharing, which is widely deployed under various incarnations, from the Web to P2P systems. File-sharing is also an important aspect of Grid communities, as many collaborations are formed around sharing of scientific data. We hence focused on an instance of resource discovery—file location—in order to get guidance from existing user traces.

In Chapter 3 we analyze three file-sharing communities in an attempt to identify possible patterns in data interests. We propose a new structure called the *data-sharing graph* that captures users' common interests in files. Based on this structure and real traces from the three communities, we show that users naturally cluster in interest-based groups.

Chapter 4: FLASK: A File-Location Algorithm for Small-World Communities

Chapter 3 shows that using the data-sharing graph for system characterization has potential for basic science, because it reveals new structures emerging in real, dynamic networks. Chapter 4 shows that the data-sharing graph is also useful for system design, because we can exploit these structures when designing file-location mechanisms.

Despite the large number of existing file-location solutions, new solutions are necessary to satisfy the requirements raised by typical Grid communities in a P2P environment. We identify these requirements and propose a file-location mechanism, called FLASK, that satisfies them by exploiting the emergent patterns in data-sharing graphs. The basic idea in FLASK is to identify groups of users with common interests in data and to disseminate relevant file location information to these groups. Chapter 4 presents in detail the design of FLASK.

Chapter 5: FLASK: Experimental Evaluation

This chapter evaluates the performance of the main components of FLASK in trace-driven simulations. The performance results validate the basic algorithmic idea—that of exploiting the naturally emerging interest-based clusters—and prove that FLASK does satisfy the file-location requirements stated in Chapter 4.

Moreover, some of the FLASK components can be used to improve existing file-location solutions. For example, parts of FLASK can improve the performance and scalability of a centralized file-location mechanism by significantly reducing the load on the central index. This chapter shows that up to 65% of this load can be reduced by information dissemination in interest-based clusters.

Chapter 6: Lessons for Resource Discovery

File location is a simplified instance of resource discovery, since files are resources that can be uniquely identified with a simple attribute: their name. However, FLASK does not build search-optimized structures based on these unique identifiers: this design approach was chosen in order to allow FLASK ideas to remain applicable to the resource discovery problem. Nevertheless, the translation of FLASK into a resource discovery solution is not obvious.

This chapter presents the lessons learned from FLASK and their applicability to the general resource discovery problem. FLASK's strength comes from disseminating file location information in interest-based clusters. Disseminating resource information can significantly improve the resource discovery performance, but clusters of interest are not easily identifiable in the context of general resources. Various approaches to determining where to disseminate resource information are discussed. This chapter also discusses FLASK's place in the resource discovery taxonomy.

Chapter 7: Discussion

The main contributions of this work are summarized in Chapter 7. We conclude with a discussion on future research directions.

CHAPTER 2
RESOURCE DISCOVERY IN DISTRIBUTED SYSTEMS

The resource discovery literature is rich and mature but lacks a common language for describing various solutions. In this chapter we propose such a common language: a taxonomy along four axes that is general enough to comprise the existing solutions, centralized and decentralized alike. We support this taxonomy by discussing previous work along its four axes.

We also propose an architecture for resource discovery. We test the strength of one component under the proposed architecture and draw important lessons for better solution design. Basically, we show evidence that relying on request propagation alone in a large, unstructured network of resource providers gives limited performance. Moreover, based on experimental studies, this performance is highly sensitive to sharing characteristics.

We shall use the terms "resource discovery" and "resource location" interchangeably.

2.1 Resource Discovery: Components and Solutions

Let us assume that every participant in the virtual organization—institution or individual—publishes information about local resources on one or more local servers. Let us call these servers nodes, or peers. Nodes hence provide information about resources: some advertise locally stored files or the node's computing power, as in a traditional P2P scenario; others advertise all the resources shared by an institution, as in a typical Grid scenario.

From the perspective of resource discovery, a Grid is thus a collection of geographically distributed nodes that may join and leave at any time and without notice (for example, as a result of system or communication failure). Users send their requests to some known (typically local) node. Typically, the node responds with the matching resource descriptions if it has them locally; otherwise it processes the request, possibly forwarding it to one or more nodes.

2.1.1 Four Axes of the Solution Space

We partition a general resource discovery solution into four architectural components: membership protocol, overlay construction, preprocessing, and query processing. This partitioning helps us recognize the unexplored regions in the solution space. It also provides the basis for comparing previous solutions from both the P2P area and the traditional distributed computing domain, solutions that we present in Section 2.1.2.

Membership Protocol

The membership protocol specifies how new nodes join the network and how nodes learn about each others (we refer to the latter part of the membership problem as *peer discovery*, although it has multiple names in the literature [68]).

Imagine a graph whose vertices are peers and whose edges indicate whether vertices know of each other. Ideally, despite frequent vertex departures and joins, this graph is a clique; that is, every member of the network has accurate information about all participants. In practice, however, this situation is impossible [26], and different protocols have been suggested, each involving different tradeoffs. For example, Gnutella uses an aggressive membership protocol that maintains the highly dynamic nodes in the membership graph connected, but at a significant communication cost [98]. More scalable with the number of nodes are membership protocols based on epidemic communication mechanisms [49].

Overlay Construction

The overlay construction function selects the set of collaborators from the local membership list. In practice, this set may be limited by such factors as available bandwidth, message-processing load, security or administrative policies, and topology specifications. Hence, the overlay network often contains only a subset of the edges of the membership graph. For example, a Gnutella node maintains a relatively small number of open connections (the average is less than 10, with 3.4 measured as of May 2001 [98]) but knows of many more peers (hundreds) at any given time.

The characteristics of the overlay topology have a significant effect on performance. For example, Barabási and Albert [15] show a strong correlation between robustness and the power-law topology; Adamic et al. [3] give a search algorithm that exploits the power-law topology in a cost-efficient way; and Kleinberg [65] presents an optimal algorithm for search in small-world graphs with a particular topology (two-dimensional lattice) and knowledge about global properties, such as distance between any two nodes. On the other hand, a large number of dynamic, real networks, ranging from the Internet to social and biological networks, all exhibit the same power-law and small-world patterns (as surveyed in [6], [34], and [85] and discussed in detail in [14] and [120]).

Preprocessing

Preprocessing refers to off-line processing used to enhance search performance prior to executing requests. For example, prefetching is a preprocessing technique, but caching is not. Another example of a preprocessing technique is dissemination of resource descriptions, that is, advertising descriptions of the local resources to other areas of the network for better search performance and reliability. A third example of preprocessing is rewiring the overlay network to adapt to changes in usage characteristics.

It is not obvious, however, that such preprocessing strategies work in the dynamic environments that we consider, in which resources may leave the pool and resource characteristics and user behavior may change suddenly. A recent result [30] shows that, in a static environment, the optimum replication of an item for search in unstructured networks is proportional to the square root of the popularity of that item.

Request Processing

The request-processing function has a local and a remote component. The local component looks up a request in the local information, processes aggregated requests (e.g., a request for A and B could be broken into two distinct requests to be treated separately), and/or applies local policies, such as dropping requests unacceptable for the local administration.

The remote component implements the request propagation rule. Request propagation is currently an active research topic in the P2P area [95, 112, 101, 77, 122, 58, 111]. In some cases, request propagation rules are dictated by other components of the resource discovery mechanism, as with distributed hash tables [95, 112, 101, 122], where the overlay and the propagation rules are strongly correlated. In an unstructured network, however, there are many degrees of freedom in choosing the propagation rule. Various strategies can be employed, characterized by the number of neighbors to which a request is sent and the way in which these neighbors are selected.

2.1.2 Previous Solutions

To provide a basis for our proposed characterization scheme, we discuss here existing solutions and related work from the perspective of the four architectural components presented above.

Many solutions to resource discovery presume the existence of globally unique names. In some cases, this naming scheme is natural (for example, filenames used as global identifiers in P2P file-sharing systems); in others, it is created to support discovery. In the context of Grid computing it is difficult (if even possible) to define a global naming scheme capable of supporting attribute-based resource identification. We first discuss resource location solutions that exploit natural naming schemes.

Domain Name Service [79] is perhaps the largest such system that provides name-based location information. Its hierarchical topology dictates the design of all four components: nodes (domains) join at a specified address in the hierarchy, the overlay function maintains the domain-based tree structure, requests are propagated upward in the hierarchy.

Recent contributions to name-based resource location solutions have been proposed in the context of P2P file-sharing systems, such as Gnutella and Napster. The basic mechanism used in Gnutella is flooding. Its flooding-based *membership component* manages a highly dynamic set of members (with median lifetime per node of about 60 minutes [103]) by sending periodic messages. Its *overlay function* selects a fixed number of nodes from those alive (in most instances, the first nodes of the membership list). Flooding is also the core of the *request-processing component*: requests are propagated in the overlay until their time-to-live expires. No *preprocessing* component is active in Gnutella. Answers are returned along the same trajectory, from node to node, to the node that initiated the request. Gnutella's relatively good search performance (as measured in number of hops) is achieved at the cost of intensive network use [98].

Napster uses a centralized approach: a file index is maintained at a central location, while real data (files) are widely distributed on nodes. The *membership* component is centralized: nodes register with (and report their locally stored files to) the central index. Hence, the *request-processing* component is a simple lookup in the central index. Napster does not use a distinct *overlay* function or a *preprocessing* component.

Distributed hash table structures such as CAN [95], Chord [112], Tapestry [122], and Pastry [101] build search-efficient overlays. All have similar *membership* and *request processing* components, based on information propagation in a structured overlay. Differentiating these four solutions is the definition of the node space, and consequently, the *overlay function* that preserves that definition despite the nodes' volatility: ring in Chord, d-coordinate space on a torus in CAN, Plaxton mesh [89] in Pastry and Tapestry. The maintenance of the overlay via "I'm alive" messages can be considered *preprocessing*.

The file location mechanism in Freenet [28] uses a *request-propagation* component based on dynamic routing tables. Freenet includes both file management and file location mechanisms: popular files are replicated closer to users, while the least popular files eventually disappear.

The solutions discussed above are concerned with locating resources that inherently can be named. Solutions that create an artificial name have been proposed for attribute-based service location (Ninja) and as location-independent identifiers (Globe).

In Ninja's service location service [50, 51], services are named based on a most relevant subset of their attributes. Its *preprocessing* component disseminates lossy aggregations (summaries) of these names up a hierarchy. Requests are then *guided* by these summaries up or down the hierarchy, in a B-tree search fashion. The fix *overlay function* (hence, the construction of the hierarchy) is specified at deployment.

The location mechanism in Globe [117] is based on a search-tree-like structure where the search keys are globally unique names. Its naming service [13] transforms an URL into a location-independent unique identifier. Consequently, the *overlay* *function* and the *membership* and the *request-processing* components are designed conform to the search-tree.

Among the few attribute-based resource location services is Condor's Matchmaker [92]. Resource descriptions and requests are sent to a central authority that performs the matching. Hence, the equivalent of the *preprocessing* component is registering resources with the central server; the *overlay* function always returns the address of the central server; *request processing* means sending the request to the central server; no *membership* component is necessary, other than, perhaps, making the address of the central server known to the new comers.

Lee and Benford [69] propose a resource discovery mechanism based on request propagation: nodes (called *traders*) forward unsolved requests to other nodes in an unstructured overlay. The *overlay function* takes into account neighbors' expertise and preference: a node connects to a node that has useful services and/or good recommendations. This evaluation uses information collected by the *preprocessing* component: traders explore the network off-demand, whenever necessary, and disseminate state changes via flooding.

Another solution is provided by the Globus Toolkit MDS [31]. Initially centralized, this service moved to a decentralized structure as its pool of resources and users grew. In MDS-2, a Grid consists of multiple information sources that can register with index servers ("nodes" in our terminology) via a registration protocol. Nodes and users can use an enquiry protocol to query other nodes to discover entities and to obtain more detailed descriptions of resources from their information sources. Left unspecified is the *overlay construction function*, the techniques used to associate information sources to nodes and to construct an efficient, scalable network of index servers.

2.2 Experimental Studies

Our objective is to observe and quantify the synergies emerging from the interaction of the four components of resource discovery in flat, unstructured networks. To understand how much to request from the other components, we ask the following question:

Q1 How powerful is the request propagation alone?

Therefore, we need experimental evaluations of the performance of this component in order to understand how much support is needed from the other components. In the absence of a large-scale, deployed Grid available to test design ideas, we modeled an environment in which we experimented with a set of resource discovery mechanisms. This emulated Grid, while specifically designed to test resource location ideas, can be easily expanded to evaluate other services on large-scale testbeds, such as resource selection and scheduling. More important, it provides a framework for evaluating aggregations of cooperative services, such as resource location, resource selection, and scheduling.

2.2.1 Emulated Grid

Existing Grid simulators are specialized for certain services, such as scheduling [70] or data replication [93]. Others, such as the MicroGrid [109], run Grid software and applications on virtual resources. No current simulator is appropriate for or easily extensible to evaluating generic Grid services. We built an emulated Grid that is scalable and is suitable for resource discovery but also is easily extensible to other purposes.

In our framework, nodes form an overlay network. Each node is implemented as a process that communicates with other nodes via TCP. Each node maintains two types of information: (1) information about a set of resources and (2) information about other nodes in the overlay network (including membership information).

The large number of processes needed by our large-scale emulation raises multiple problems, ranging from resource starvation to library limitations. For the preliminary experiments (of up to 32,768 virtual nodes) presented in Section 2.3, we used 128 systems (256 processors) communicating over fast Ethernet of the Chiba City cluster of Argonne National Laboratory. With minimal modifications, the framework could be used in real deployments.

2.2.2 Modeling the Grid Environment

Four environment parameters influence the performance and the design of a resource discovery mechanism:

1. *Resource information distribution and density*: Some nodes provide information on a large number of resources, whereas others just on a few (for example, home computers): resource information distribution models thus behaviors such as free riding. Also, some resources are common (e.g., PCs running Linux), while others are rare or even unique (e.g., specific services or data): resource density parameter distinguishes thus the “needle” from the “hay”.

2. *Resource information dynamism*: Some resource attributes are highly variable (e.g., CPU load or availably bandwidth between two nodes), while others vary so slowly that they can be considered static for many purposes (e.g., operating system version, number and type of CPUs in a computer, etc.). Dynamic information can hinder the performance of caching techniques, for example.

3. *Request popularity distribution*: The popularity of users’ requests for resources varies. For example, studies have shown that HTTP requests follow Zipf distributions [22]. Our analysis [57] of a scientific collaboration, on the other hand, reveals different request popularity patterns, closer to a uniform distribution.

4. *Peer participation*: The participation of peers, or nodes, varies more significantly in P2P systems than in current Grids. Influenced by incentives, some nodes activate in the network for longer than others.

The failure rate in a large-scale system is inevitably high and hence necessary to model. This factor can easily be captured by two of the parameters just listed, namely, resource information dynamism and peer participation. The effects of failure are visible at two levels: the resource level and the node level. When resources fail, the nodes that publish their descriptions may need to update their local information to reflect the change. Resource failure can therefore be seen as yet another example of resource attribute variation and can be treated as part of resource information dynamism. When nodes fail, not only do their resources disappear, but they cease to participate in maintaining the overlay and processing remote requests. Node failures can therefore be captured in the peer participation parameter as departures. We note, however, that node failures are ungraceful (unannounced) departures. Moreover, such failures may not be perceived in the same way by all peers; for example, in the case of network partitioning, a node may seem failed to some peers and alive to others.

To isolate some of the correlations between the many parameters of our study, we used a simplified, optimistic Grid model characterized by static resource attributes, constant peer participation, and no failures. Thus, we model only the resource and request distributions.

Resource Distributions

In Grids and peer-to-peer environments, the total number of resources increases with the number of nodes, so we model this as well. We assume that the average number of resources per node remains constant with the increase in the network size: in our experiments, we (arbitrarily) chose this constant equal to 5.

New nodes often bring new types of resources, however, such as unique on-line instruments, new data, and new, possibly locally developed, applications. To account for these, we allowed the set of resource types to increase slowly (5%) with the number of nodes in the system.

In this context, we experimented with two resource distributions of different degrees of fairness, as presented in Figure 2.2.2: a balanced distribution, with all nodes providing the same number of resources, and a highly unbalanced one, generated as a geometric distribution in which most resources are provided by a small number of nodes.

Request Distributions

Although usage patterns can be decisive in making design decisions, we faced the problem of not having real user request logs, a problem inherent in systems during

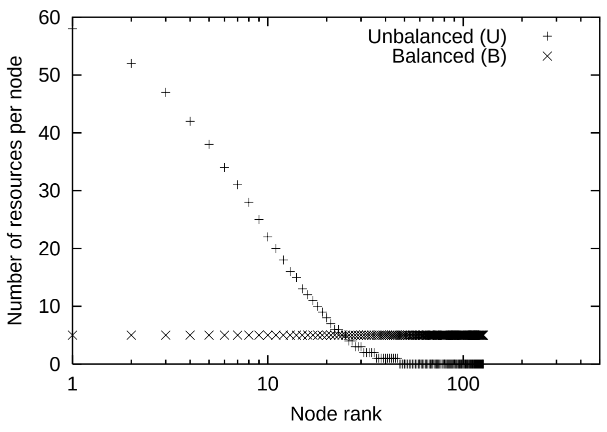


Figure 2.1: Distribution of resources on nodes: balanced (all nodes have equal number of resources) and unbalanced (a significant part of nodes have no resources).

the design phase. We therefore logged, processed, and used one week's requests for computers submitted to the Condor [74] pool at the University of Wisconsin. This pool consists mainly of Linux workstations and hence is a rather homogeneous set of resources. On the other hand, since it is intensively used for various types of computations, the requests specify various attribute values (e.g., minimum amount of available memory or required disk space). We processed these requests to capture their variety. We acknowledge, however, that despite their authenticity, these traces may not accurately represent the request patterns in a sharing environment that usually comprises data and services in addition to computers.

We also experimented with a synthetic request popularity distribution modeled as a uniform distribution. Figure 2.2.2 highlights the differences between the two request distributions. The Condor traces exhibit a Zipf-like distribution, where a small number of distinct requests appear frequently in the set of 2000 requests considered. In the pseudo-uniform distribution, on the other hand, requests are repeated about the same number of times. We evaluated various resource location strategies in overlay networks ranging in size from 128 to 32768 (2^7 to 2^{15}) nodes. In our experiments we randomly chose a fixed percentage of nodes to which we sent independently generated sets of 200 requests. The same sets of requests, sent to the same nodes, respectively, were repeated to compare various resource discovery strategies.

2.2.3 Resource Discovery Mechanisms

Our objective is to evaluate the performance of the request processing component alone. To this end, we considered a set of simple resource discovery mechanisms constructed by fixing three of the four components presented in Section 2.1.1 and varying the fourth: the request-processing component.

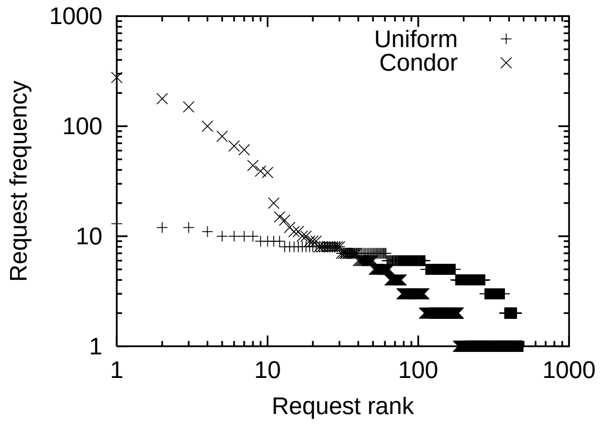


Figure 2.2: Distribution of user requests.

For the *membership protocol* we use a join mechanism that is commonly used in P2P systems: a node joins by contacting a member node. Contact addresses of member nodes are learned out-of-band: typical sources are web pages (as in Gnutella) or word-of-mouth. A node contacted by joining members responds with its membership information. Membership information is passively enriched over time: upon the receipt of a message from a previously unknown node, a node adds the new address to its membership list.

In our design, the *overlay function* accepts an unlimited number of neighbors: hence, we allowed the overlay connectivity to grow as much as the membership information. In this way, we neutralized one more component, aiming to understand the correlations between graph topology and discovery performance. We generated the starting overlay by using a hierarchy-based Internet graph generator [33].

We assumed no *preprocessing*.

Our design of the *request-processing component* is based on forwarding. We assumed simple requests, satisfiable only by perfect matches. Hence, local processing is minimized: a node that has a matching resource responds to the requester; otherwise, it decrements TTL and forwards it (if TTL>0) to some other node. Requests are dropped when received by a node with no other neighbors or when TTL=0.

We evaluated four request propagation strategies:

1. *Random walk*: the node to which a request is forwarded is chosen randomly from the local membership view. No extra information is stored on nodes.

2. *Learning-based*: nodes learn from experience by recording the requests answered by other nodes. A request is forwarded to the peer that answered similar requests previously. If no relevant experience exists, the request is forwarded to a randomly chosen node.

3. *Best-neighbor*: the number of answers received from each peer is recorded (without recording the type of request answered). A request is forwarded to the peer who answered the largest number of requests. First requests are always sent to random nodes.

4. *Learning-based + best-neighbor*: this strategy is identical with the learning-based strategy except that, when no relevant experience exists, the request is forwarded to the best neighbor instead of to a random node.

2.3 Experimental Results

This section presents preliminary results in two areas: (1) quantification of the costs of simple resource discovery techniques based on request-forwarding (no preprocessing), and (2) effects of resource and request distributions on resource discovery performance.

2.3.1 Quantitative Estimation of Resource Discovery Costs

Question *Q1*, reformulated, is:

> *What are the search costs in an unstructured, static network in the absence of preprocessing?*

To this end, we considered time-to-live infinite. The answer is presented in Figures 2.3.1, 2.3.1, and 2.3.1: the learning-based strategy is the best regardless of resource-sharing characteristics, with fewer than 200 hops response time per request for the largest network in our experiment. For a network of thousands of nodes (hence, possibly thousands of institutions and individuals) the average response time is around 20 hops. Assuming 20 ms to travel between consecutive nodes on a path (10 ms. latency in a metropolitan area network and 10 ms. necessary for request processing), then a path of 20 hops takes less than half a second.

Key to the performance of the learning-based strategy is the fact that it takes advantage of similarity in requests by using a possibly large cache. It starts with low performance until it builds its cache.

The random-forwarding algorithm has the advantage that no additional storage space is required on nodes to record history. We also expect it to be the least efficient, however, an expectation confirmed by the results shown in Figure 2.3.1 (Condor-based user requests, unbalanced resource distribution). For all network sizes in our experiments, the learning-based algorithm consistently performs well, while its more expensive version (learning-based + best neighbor) proves to be rather unpredictable in terms of performance (see, for example, the large standard error deviation for 1024 and 2048 simulated nodes in 2.3.1).

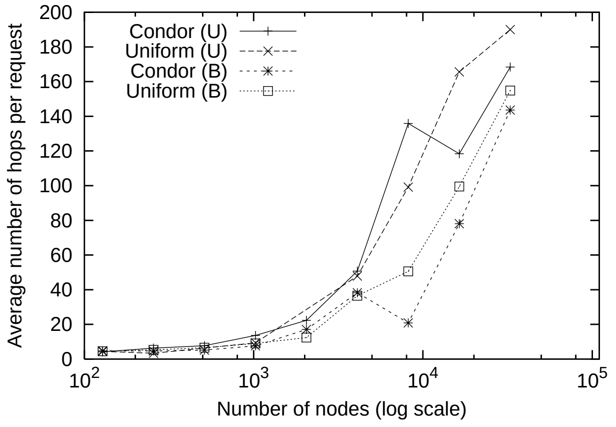


Figure 2.3: Performance (in average number of hops) of learning-based forwarding strategy for the two request distributions (Condor and uniform), in two environments with different resource-sharing characteristics (balanced B and unbalanced U).

We emphasize that these results do not advocate one strategy over another but give a numerical estimate of the costs (in response time) involved. These estimates are useful in two ways. First, they give a lower bound for the performance of resource location mechanisms based on request propagation. They show that more sophisticated strategies (potentially including preprocessing techniques) are needed for efficient resource location in large-scale (tens of thousands institutions) Grids. Second, they can be used in estimating the performance of more sophisticated mechanisms that have a request-propagation component (as is, for example, the solution in [58]).

2.3.2 Effects of the Environment

From the results obtained, a new question emerges:

Q2 *How sensitive is the search performance to sharing characteristics?*

Figure 2.3.1 highlights the influence of user request popularity distribution on the performance of the learning-based request forwarding strategy. (Of the strategies we considered, this is the most sensitive to user request patterns.) The slightly better performance in the fair-sharing environment is due to the random component of this strategy, employed when no relevant previous information on a specific request exists: random forwarding has a better chance of reaching a useful node when information is distributed fairly on nodes. The learning-based strategy takes most advantage of the Condor request distribution, where a significant part of the requests are repeated (and hence can benefit from previous experience).

The best-neighbor strategy is influenced more strongly by sharing patterns: compared with a balanced environment, in a highly unbalanced environment a node that

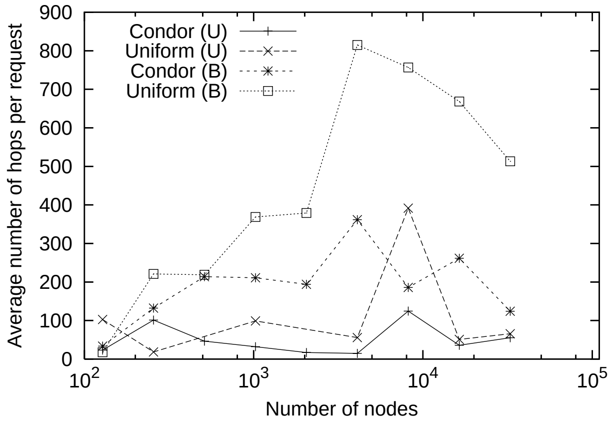


Figure 2.4: Performance (in average number of hops) of the best neighbor request forwarding strategy under different user request and sharing characteristics.

had already answered a request is more likely to have answers to other requests as well. Figure 2.3.1 shows the response latency in unbalanced and balanced environments for the two request patterns we considered: the response latency almost doubles in the balanced sharing environment as compared with the unbalanced one. We note that the performance of the best-neighbor strategy is influenced by past requests: the algorithm records the number of requests answered regardless of their type, hence it does not distinguish between nodes that answered same request n times and nodes that answered n distinct requests. This fact explains why the algorithm performs better under a uniform user distribution load than under the Condor traces: since the number of distinct requests in a uniform distribution is larger, the best neighbor identified by this strategy has indeed a larger number of distinct resources.

2.4 Summary

In Chapter 1 we argue that the characteristics and the design objectives of Grid and P2P environments will converge, even if they continue to serve different communities. Grids will increase in scale and inherently will need to address intermittent resource participation, while P2P systems will start to provide more complex functionalities, integrating data and computation sharing with various quality of service requirements. We are therefore studying the resource discovery problem in a resource-sharing environment that combines the characteristics of the two environments: the complexity of the Grids (that share a large diversity of resources, including data, applications, computers, online instruments, and storage) with the scale, dynamism, and heterogeneity of today's P2P systems.

We thus propose a decentralized architecture for resource discovery in Grids, in which the participating entities maintain and publish information about a possibly

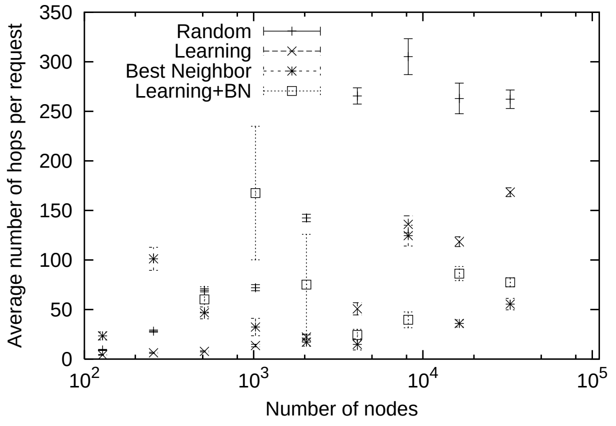


Figure 2.5: Performance of all four request-forwarding strategies for a Condor request load in the unbalanced resource-sharing environments.

large set of resources. In this very general framework, we have identified four components that can describe any resource discovery design: membership protocol, overlay function, preprocessing, and request processing.

Of the four components, we have focused in this chapter on request processing: we evaluate four request propagation strategies under various environmental assumptions. The main results of our study are:

- The request propagation component alone may not perform satisfactorily in a large network of nodes. A more efficient resource discovery mechanism may require contribution from the preprocessing component, as well.
- The request propagation component and, implicitly, any resource discovery mechanisms that relies on it, is highly dependent on sharing characteristics.

We have also proposed a Grid emulator for evaluating resource discovery techniques based on request propagation.

CHAPTER 3
SMALL-WORLD FILE-SHARING COMMUNITIES

The identification of a general framework for any resource discovery mechanism (as presented in the previous chapter) significantly simplifies the design of a resource discovery solution. However, the design space remains vast and the search for an efficient solution is still challenging.

As shown before, various solutions were chosen as to fit specific requirements. However, none looked at users behavior in an attempt to adapt a resource location mechanism to naturally occurring patterns. We are trying to get inspiration by looking at user behavior: what resources do users ask for? What are the relevant patterns in their requests for resources?

This chapter answers the question:

> *What patterns in user behavior are relevant and can be useful for designing a resource location solution?*

Given the lack of user traces from deployed, non-experimental Grids that could give insights into what resources users request, we shall profit of the many file-sharing deployments and thus focus our study on a particular type of resources: files.

3.1 Intuition

It is not news that understanding the system properties can help guide efficient solution design. A well known example is the relationship between file popularity in the Web and cache size. The popularity of web pages has been shown to follow a Zipf distribution [17, 22]: few pages are highly popular and many pages are requested few times. As a result, the efficiency of increasing cache size is not linear: caching is useful for the popular items, but there is little gain from increasing the cache to provision for unpopular items.

As a second example, many real networks are power law. That is, their node degrees are distributed according to a power law, such that a small number of nodes have large degrees, while most nodes have small degrees. Adamic et al. [3] propose a mechanism for probabilistic search in power-law networks that exploits exactly this characteristic: the search is guided first to nodes with high degree and their many neighbors. This way, a large percentage of the network is covered fast.

This type of observations inspired us to look for patterns in user requests for resources. But what patterns?

3.1.1 Patterns, Patterns Everywhere

It is believed that the study of networks started with Euler's solution of the Königsberg bridge problem in 1735. The field has since extended from theoretical results to the analysis of patterns in real networks. Social sciences have apparently the longest history in the study of real networks [85], with significant quantitative results dating from the 1920s [44].

The development of the Internet added significant momentum to the study of networks: by both facilitating access to collections of data and by introducing new networks to study, such as the Web graph, whose nodes are web pages and edges are hyperlinks [23], the Internet at the router and the AS level [35] and the email graph [86].

The study of large real networks led to fascinating results: recurring patterns emerge in real networks (see [6, 14, 34, 85] for good surveys). For example, a frequent pattern is the power-law distribution of node degree, that is, a small number of nodes act as hubs (having a large degree), while most nodes have a small degree. Examples of power-law networks are numerous and from many domains: the phone-call network (long distance phone calls made during a single day) [1, 5], the citation network [96], and the linguistics network [25] (pairs of words in English texts that appear at most one word apart). In computer science, perhaps the first and most surprising result at its time was the proof that the random graph-based models of the Internet (with their Poisson degree distribution) were inaccurate: the Internet topology had a power-law degree distribution [35]. Other results followed: the web graph [16, 23] and the Gnutella overlay (as of year 2000) [98] are also power-law networks.

Another class of networks are the "small worlds." Two characteristics distinguish small-world networks: first, a small average path length, typical of random graphs (here "path" means shortest node-to-node path); second, a large clustering coefficient that is independent of network size. The clustering coefficient captures how many of a node's neighbors are connected to each other. This set of characteristics is identified in systems as diverse as social networks, in which nodes are people and edges are relationships; the power grid system of western USA, in which nodes are generators, transformers, substations, etc. and edges are transmission lines; and neural networks, in which nodes are neurons and edges are synapses or gap junctions [120].

3.1.2 Research Questions

Newman shows that scientific collaboration networks in different domains (physics, biomedical research, neuroscience, and computer science) have the characteristics of small worlds [82, 83, 84]. Collaboration networks connect scientists who have written articles together.

Moreover, Girvan and Newman [47] show that well-defined groups (such as a research group in a specific field) can be identified in (small-world) scientific collaboration networks. In parallel, a theoretical model for small-world networks by Watts and Strogatz [119] pictures a small world as a loosely connected set of highly connected subgraphs.

From here, the step is natural: since scientists tend to collaborate on publications, they most likely use the same resources (*share* them) during their collaboration: for example, they might use the same instruments to observe physics phenomena, or they might analyze the same data, using perhaps the same software tools or even a common set of computers. This means that if we connect scientists who use the same files, we might get a small world. Even more, we might be able to identify groups that share the same resources. Notice that the notion of "collaboration" transformed into "resource sharing": the social relationships do not matter anymore, scientists who use the same resources within some time interval may never hear of each other.

Resource sharing in a (predominantly) scientific community is the driving force of computational Grids. If we indeed see these naturally occurring sharing patterns and we find ways to exploit them (e.g., by identifying users grouped around common sets of resources), then we can build mechanisms that can tame the challenges typical of large-scale, dynamic, heterogeneous, latency-affected distributed systems.

The research question now become clear:

Q3 Are there any patterns in the way scientists share resources that could be exploited for designing mechanisms?

But resource sharing also exists outside scientific communities: peer-to-peer systems or even the Web facilitate the sharing of data. Another question arises:

Q4 Are these characteristics typical of scientific communities or are they more general?

This chapter answers these two questions: it shows that small-world patterns exist in diverse file-sharing communities.

3.2 The Data-Sharing Graph

To answer question *Q3*, we define a new structure that captures the virtual relationship between users who request the same data at about the same time. We call this structure *the data-sharing graph.*

Definition: The data-sharing graph is a graph in which nodes are users and an edge connects two users with similar interests in data.

We consider one similarity criterion in this thesis: the number of shared requests within a specified time interval.

To answer question *Q4*, we analyze the data-sharing graphs of three different file-sharing communities. Section 3.3 presents briefly these systems and the traces we used. We discover that in all cases, for different similarity criteria, these data-sharing graphs are small worlds. The next sections show that using the data-sharing graph for system characterization has potential both for basic science, because we can identify new structures emerging in real, dynamic networks (Section 3.4); and for system design, because we can exploit these structures when designing data location and delivery mechanisms (Section 3.6). In Chapters 4 and 5 we shall present in details such a data-location mechanism.

3.3 Three Data-Sharing Communities

We study the characteristics of the data-sharing graph corresponding to three file-sharing communities: a high-energy physics collaboration (Section 3.3.1), the Web as seen from the Boeing traces (Section 3.3.2), and the Kazaa peer-to-peer file-sharing system seen from a large ISP in Israel (Section 3.3.3).

This section gives a brief description of each community and its traces. In addition, we present the file popularity and user activity distributions for each of these traces as these have a high impact on the characteristics of the data-sharing graph: intuitively, a user with high activity is likely to map onto a highly connected node in the data sharing graph. Similarly, highly popular files are likely to produce dense clusters.

Table 3.1: Characteristics of traces analyzed.

System	Users	Requests		Duration
		All	Distinct	Traces
D0	317	2,757,015	193,662	180 days
Web	60,826	16,527,194	4,794,439	10 hours
Kazaa	14,404	976,184	116,509	5 days

3.3.1 The D0 Experiment: a High-Energy Physics Collaboration

The D0 experiment [32] is a virtual organization comprising hundreds of physicists from more than 70 institutions in 18 countries. Its purpose is to provide a worldwide system of shareable computing and storage resources that can together solve the common problem of extracting physics results from about a Petabyte (c.2003) of measured and simulated data. In this system, data files are read-only and typical jobs analyze and produce new, processed data files. The tracing of system utilization is possible via a software layer (SAM [76]) that provides centralized file-based data management.

We analyzed logs over the first six months of 2002, amounting to about 23,000 jobs submitted by more than 300 users and involving more than 2.5 million requests for about 200,000 distinct files. A data analysis job typically runs on multiple files (117 on average). Figure 3.1 shows the distribution of the number of files per job.

In addition to studying the properties of the data-sharing graph, we evaluated the applicability of traditional workload models (such as file size distribution, file popularity distribution, and job interarrival time) in this environment. We learnt that some of the traditional workload models are appropriate, such as job inter-arrival time and the existence of diurnal usage patterns, while others (e.g., file size distribution and file popularity distribution) are inaccurate. These results are presented in detail in [57].

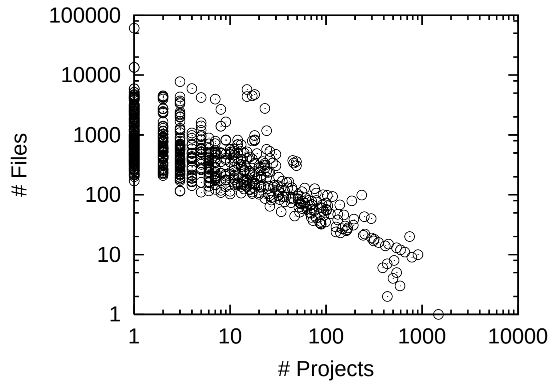


Figure 3.1: Number of file requests per project in D0.

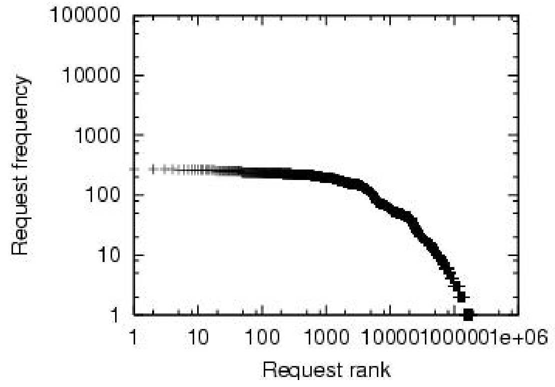


Figure 3.2: File popularity distribution in D0.

The daily activity is relatively constant (Figure 3.3), with a few significant peaks—corresponding perhaps to approaching paper submission deadlines in high-energy physics. User activity (Figure 3.4) is highly variable, with scientists who scan from tens of thousands of distinct data files to just a couple.

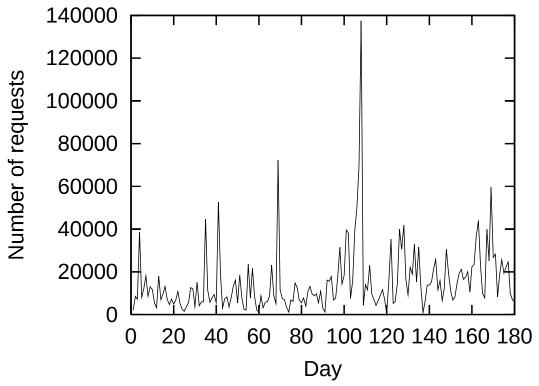


Figure 3.3: Number of file requests per day in D0.

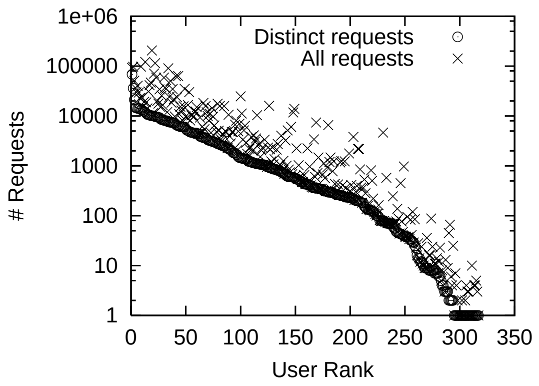


Figure 3.4: Number of files (total and distinct) asked by each user during the 6-month interval.

In D0 file popularity does not follow the Zipf's law typical of Web requests. (Figure 3.2). The reason, we believe, is that data in this scientific application is more uniformly interesting: a typical job swipes a significant part of the data space (and hence file set) in search of particular physics events.

3.3.2 The Web

We use the Boeing proxy traces [20] as a representative sample for Web data access patterns. These traces represent a five-day record from May 1999 of all HTTP requests (more than 20M requests per day) from a large organization (Boeing) to the Web. Because traces are anonymized and IDs are not preserved from day to day, our study was limited to one-day intervals. However, given the intense activity recorded (Figure 3.5 shows the number of requests per second), this limitation does not affect the accuracy of our results. Here we study a representative 10-hour interval.

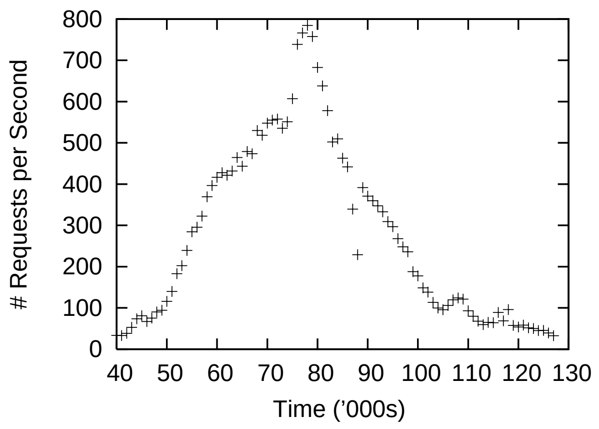


Figure 3.5: Requests per second (averaged over 15 min. intervals). In our experiments we used the traces from the interval 40,000 and 80,000 seconds (until right after the peak).

For the study of Web traces, we consider a user as an IP address. During the 10-hour interval, 60,826 users sent 16.5 million web requests, of which 4.7 million requests were distinct. It is possible that the same IP address corresponded in fact to multiple users (for example, for DHCP addresses or shared workstations). We do not have any additional information to help us identify these cases or evaluate their impact. However, given the relatively short intervals we consider in our studies—from 2 minutes to a couple of hours—the chances of multiple users using the same IP are small.

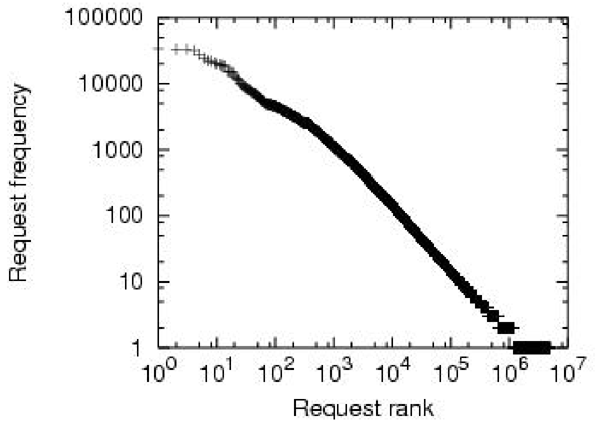


Figure 3.6: The file popularity distribution in Web follows Zipf's law.

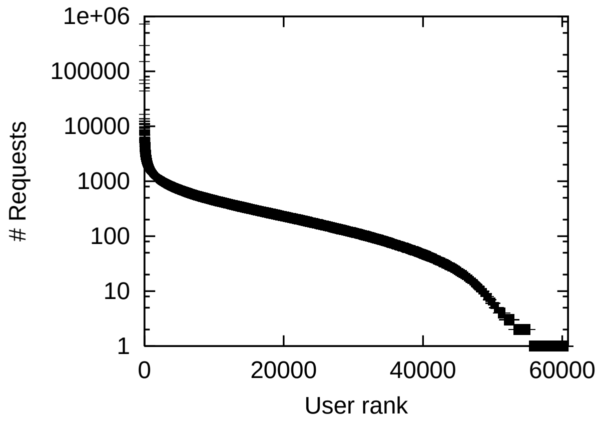


Figure 3.7: Number of requests per Web user.

3.3.3 The KaZaA Peer-to-Peer Network

Kazaa is a popular peer-to-peer file-sharing system with an estimated number of more than 4 million concurrent users as of June 2003 [108].

Few details are publicly available about the Kazaa protocol. Apparently, Kazaa nodes dynamically elect "supernodes" that form an unstructured overlay network and use query flooding to locate content. Regular nodes connect to one or more super-nodes and act as querying clients to super-nodes. Control information, such as queries, membership, and software version, is encrypted. Once content has been located, data is transfered (unencrypted) directly from provider to requester using the HTTP protocol. In order to improve transfer speed, multiple file fragments are downloaded in parallel from multiple providers.

Since control information is encrypted, the only accessible traffic information can be obtained from the download channel. As a result, we can only gather information about the files requested for download and not about files searched for (therefore, typos are naturally filtered). Details on how Kazaa traces were recorded as well as a thorough analysis of the Kazaa traffic are presented in [71].

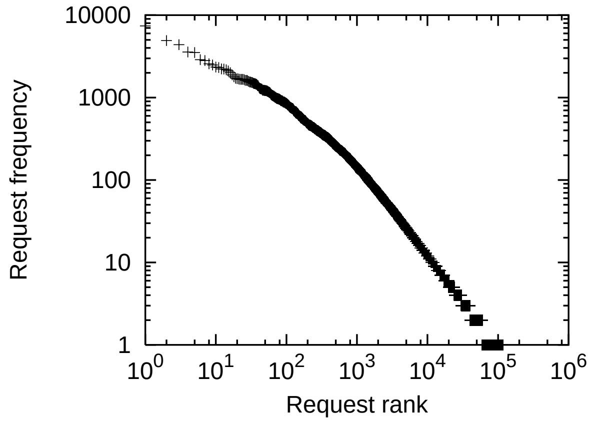


Figure 3.8: The file popularity distribution in Kazaa follows Zipf's law.

We had access to five days of Kazaa traffic from January 2003, during which 14,404 users downloaded 976,184 files, of which 116,509 were distinct. Users are identified based on their (anonymized) user ID that appears in the HTTP download request. The user population is formed of Kazaa users who are clients of the ISP: similar to the Boeing traces, these traces give information about only a limited set of users in the system.

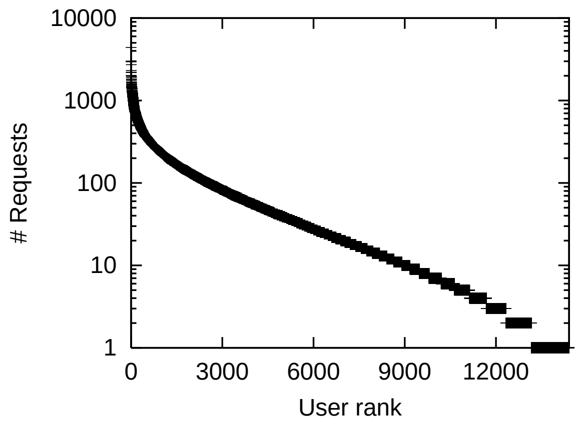


Figure 3.9: Number of requests per user in KaZaa.

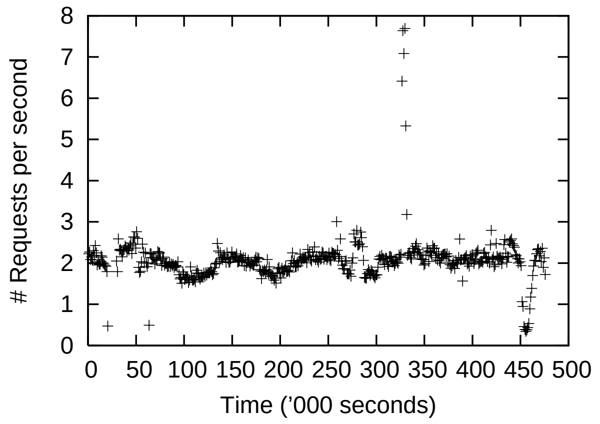


Figure 3.10: Requests per second in KaZaa (averaged over 100s)

3.4 Small-World Data-Sharing Graphs

Data-sharing graphs are built using the definition in Section 3.2: users are nodes in the graph and two users are connected if they have similar interests in data during some interval. For the rest of this paper we consider one class of similarity criteria: we say that two users have similar data interests if the size of the intersection of their request sets is larger than some threshold. This section presents the properties of data-sharing graphs for the three communities introduced previously.

The similarity criterion has two degrees of freedom: the length of the time interval τ and the threshold on the number of common requests μ. Section 3.4.1 studies the dependence between these parameters for each of the three data-sharing communities.

Sections 3.4.2 and 3.4.3 present the properties of the data-sharing graphs. We shall see that not all data-sharing graphs are power law. However, they all exhibit small-world characteristics, a result that we support with more rigorous analysis in Section 3.5.1.

3.4.1 Distribution of Weights

We can think of the family $G(\tau, 1)$ of data-sharing graphs as weighted graphs: two users are connected by an edge labeled with the number of shared requests during a specified time period. The distribution of weights (Figures 3.11, 3.12 and 3.13) highlights differences among the sharing communities: the sharing in D0 is significantly more pronounced than in Kazaa, with weights in the order of hundreds or thousands in D0 compared to 5 in Kazaa.

3.4.2 Degree Distribution

The node degree distribution of the data-sharing graph is particularly interesting for designing or evaluating distributed applications: for example, if the graph is an overlay, it gives insights on the number of neighbors a node needs to maintain. Figures 3.14, 3.15, and 3.16 present the degree distributions for the three systems: note that the Kazaa data-sharing graph is the closest to a power-law, while D0 graphs clearly are not power-law.

3.4.3 Small-World Characteristics: Clustering Coefficient and Average Path Length

We wanted to test our intuition that, similar to scientific collaboration networks, we find small-world patterns at the resource sharing level. We consider the Watts-Strogatz definition [119]: a graph $G(V, E)$ is a small world if it has small average path length and large clustering coefficient, much larger than that of a random graph with the same number of nodes and edges.

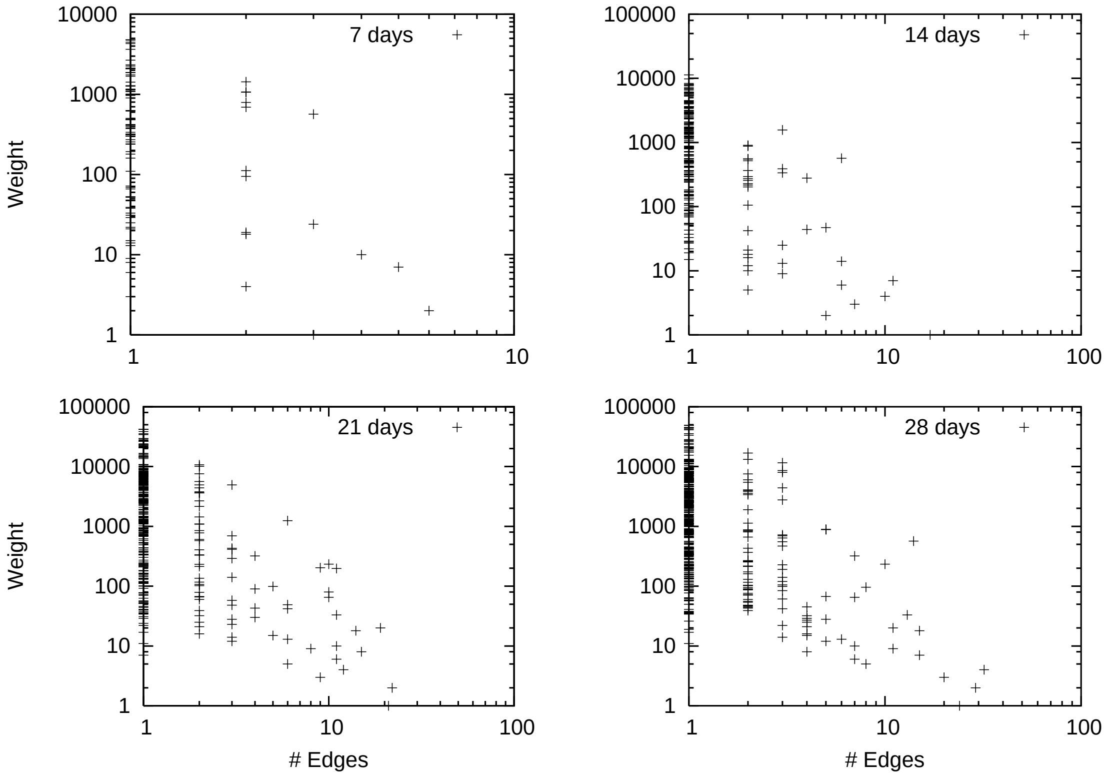


Figure 3.11: The distribution of weights in D0 data-sharing graphs for different intervals during the same period.

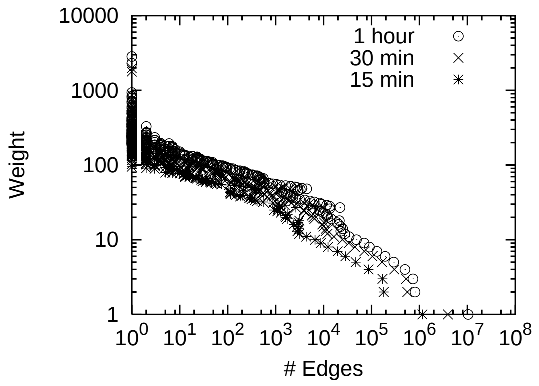


Figure 3.12: The distribution of weights in Web for data-sharing graphs for different time intervals.

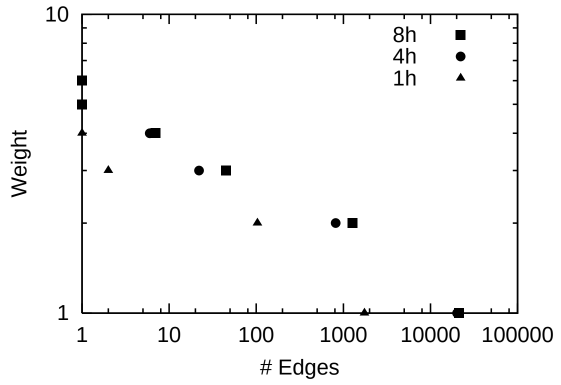


Figure 3.13: The distribution of weights in Kazaa data-sharing graphs for different time intervals.

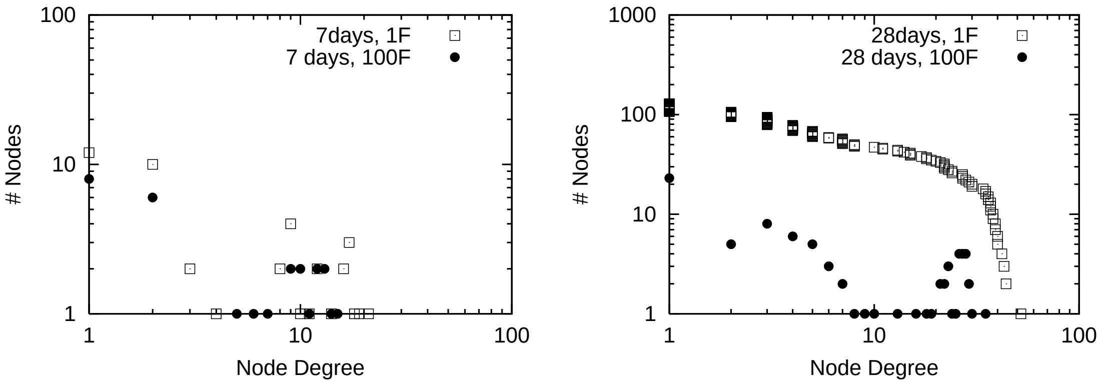


Figure 3.14: Degree distribution for D0 data-sharing graphs.

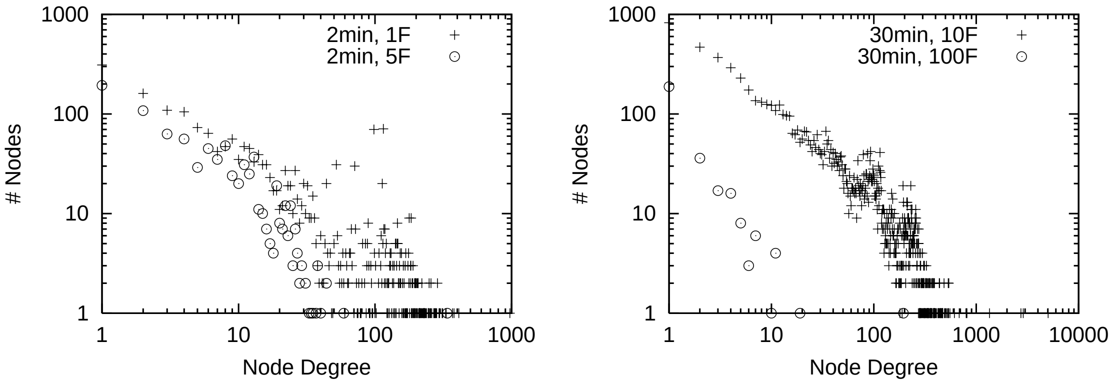
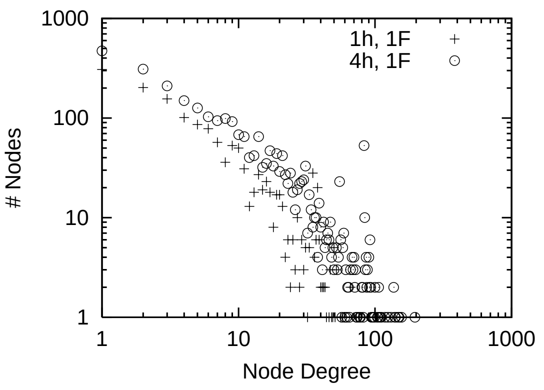


Figure 3.15: Degree distribution for Web data-sharing graphs

Table 3.2: Properties of data-sharing graphs for the three communities studied. CC_1 is the measured Watts-Strogatz clustering coefficient (Eq. 3.1), CC_2 is the measured clustering coefficient defined in Eq. 3.3; CC_r is the Watts-Strogatz clustering coefficient of random graphs (Eq. 3.4); l is the measured average path length and l_r is the average path length of random graphs (Eq. 3.5)

					#Connected	Largest connected component					Random Graph	
System	τ	μ	# Nodes	# Edges	Components	# Nodes	# Edges	CC_1	CC_2	l	CC_r	l_r
D0	7 days	1	46	153	5	35	142	0.741	0.648	2.114	0.238	2.538
D0	7 days	100	26	95	3	20	88	0.773	0.743	1.65	0.463	2.021
D0	7 days	1000	14	43	1	14	43	0.834	0.652	1.5	0.472	2.351
D0	28 days	1	129	777	9	107	757	0.716	0.641	2.476	0.133	2.388
D0	28 days	100	84	441	4	78	438	0.706	0.763	2.769	0.145	2.524
D0	28 days	1000	49	235	6	35	226	0.838	0.808	1.628	0.379	1.906
Web	120 s	1	2076	47610	100	1805	47256	0.786	0.634	2.666	0.029	2.296
Web	1800 s	1	6137	1866338	39	6049	1866271	0.808		2.056	0.102	1.519
Web	1800 s	100	143	196	20	102	172	0.720	0.130	3.6	0.033	8.851
Kazaa	1 hour	1	809	1937	97	548	1690	0.740	0.593	5.629	0.011	5.599
Kazaa	8 hours	1	3608	31252	79	3403	30555	0.652	0.473	3.611	0.005	3.705
Kazaa	8 hours	3	111	111	24	56	78	0.442	0.178	3.160	0.050	12.148



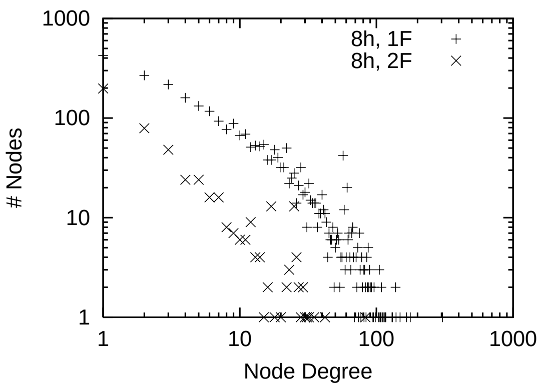


Figure 3.16: Degree distribution for Kazaa data-sharing graphs

The clustering coefficient is a measure of how well connected a node's neighbors are with each other. According to one commonly used formula for computing the clustering coefficient of a graph (Eq. 3.1), the clustering coefficient of a node is the ratio of the number of existing edges and the maximum number of possible edges connecting its neighbors. The average over all $|V|$ nodes gives the clustering coefficient of a graph (Eq. 3.2).

$$CC_u = \frac{\#\text{ edges between } u\text{'s neighbors}}{\text{Maximum }\#\text{ edges between } u\text{'s neighbors}} \tag{3.1}$$

$$CC_1 = \frac{1}{|V|}\sum_u CC_u \tag{3.2}$$

Another definition (Eq. 3.3) directly calculates the clustering coefficient of a graph as a ratio of the number of triangles and the number of triples of connected nodes, where connected triples of vertices are trios of nodes in which at least one is connected to the other two.

$$CC_2 = \frac{3 \times \text{Number of triangles on the graph}}{\text{Number of connected triples of vertices}} \tag{3.3}$$

The two definitions of the clustering coefficient simply reverse the operations—one takes the mean of the ratios, while the other takes the ratio of the means. The former definition tends therefore to weight the low-degree vertices more heavily, since they have a small denominator in Eq. 3.1.

According to the definition of clustering from Eq. 3.1, the clustering coefficient of a random graph is:

$$CC_r = \frac{2 \times |E|}{|V| \times (|V| - 1)} \tag{3.4}$$

The average path length of a graph is the average of all distances. For large graphs, measuring all-pair distances is computationally expensive, so an accepted procedure [120] is to measure it over a random sample of nodes. The average path length for the larger Web data-sharing graphs in Table 3.2 was approximated using a random sample of 5% of the graph nodes. The average path length of a random graph is given by Eq. 3.5.

$$l_r = \frac{log(|V|)}{log(|E|/|V|)} \tag{3.5}$$

We discover that data-sharing graphs for the three systems all display small-world properties. Figures 3.17, 3.18, and 3.19 show the small-world patterns—large clustering coefficient and small average path length—remain constant over time, for the entire period of our studies. Figure 3.20 summarizes the small-world result: it compares some instances of data-sharing graphs with small-world networks already documented in the literature. The axes represent the ratios of the data-sharing graphs metrics and the same metrics of random graphs of same size. Notice that most datapoints are concentrated around $y = 1$ ("same average path length") and $x > 10$ ("much larger clustering coefficient").

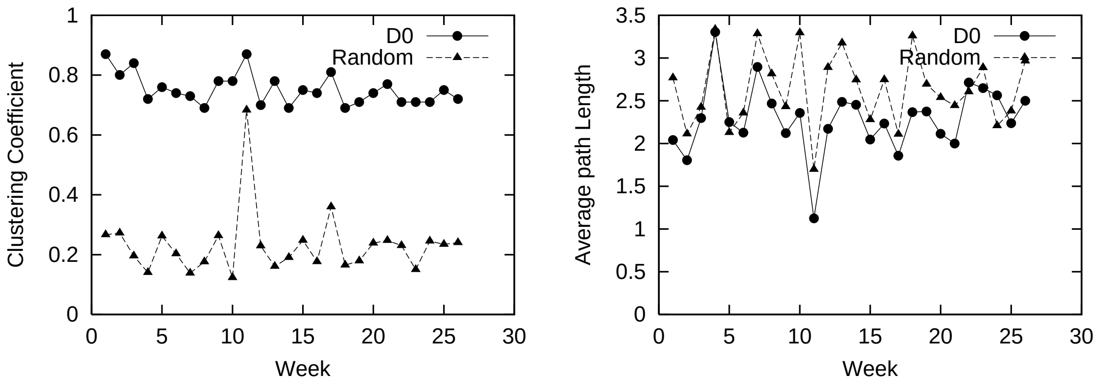


Figure 3.17: Clustering coefficients (left) and average path lengths (right) of D0 data-sharing graphs and random graphs of same size. Similarity criterion: 1 shared file during a 7-day interval.

We clearly see that data-sharing graphs of various durations and similarity criteria are small worlds. From the Watts-Strogatz model of small worlds—as loosely connected collections of highly connected subgraphs—two significant observations can be drawn. First, well connected clusters exist; due to the data-sharing graph definition, these clusters map onto groups of users with shared interests in files. Second, there is, on average, a small path between any two nodes in the data-sharing graph: therefore, for example, flooding with relatively small time-to-live would cover most of the graph.

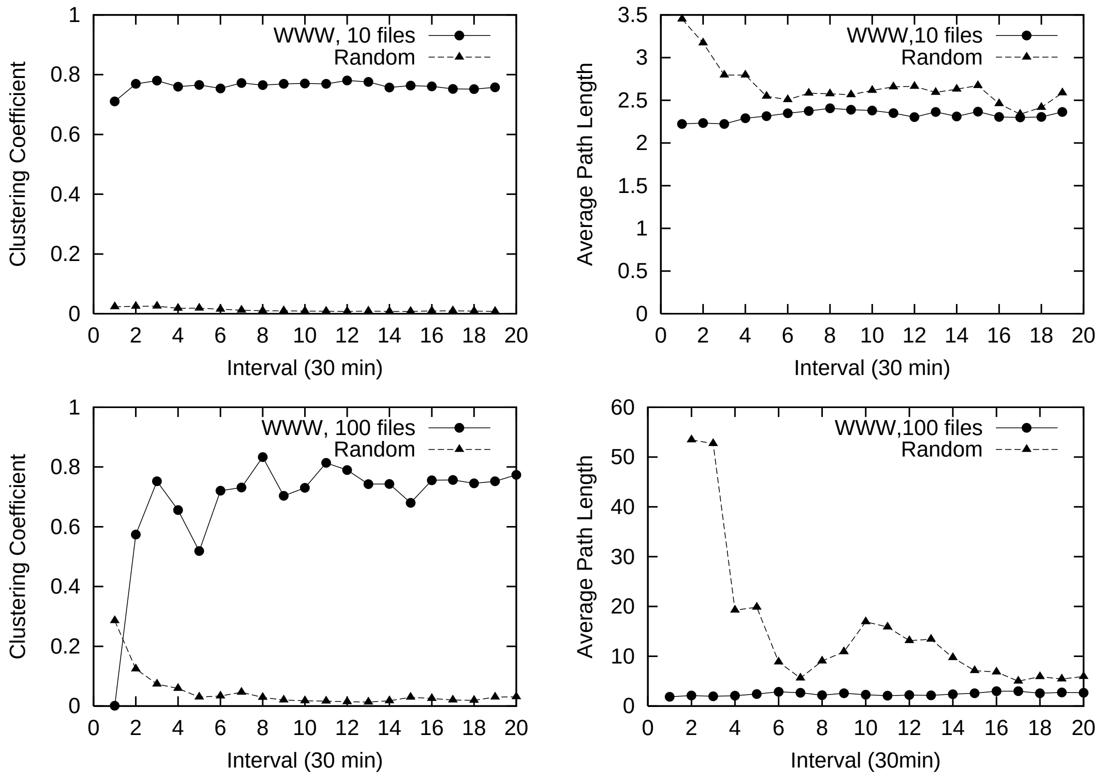


Figure 3.18: Clustering coefficients (left) and average path lengths (right) of WWW data-sharing graphs and random graphs of same size. Similarity criterion: 10, respectively 100 shared requests during a half-hour interval.

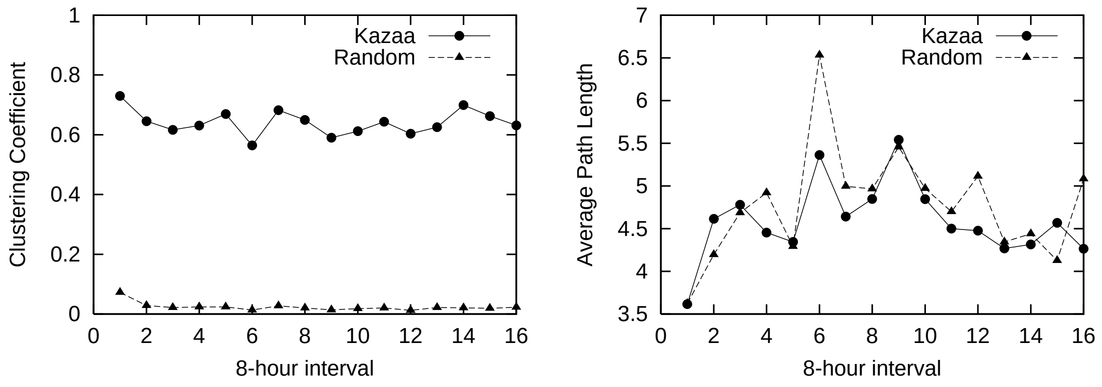


Figure 3.19: Clustering coefficients (left) and average path lengths (right) of Kazaa data-sharing graphs and random graphs of same size. Similarity criterion: 2 shared requests during an 8-hour interval.

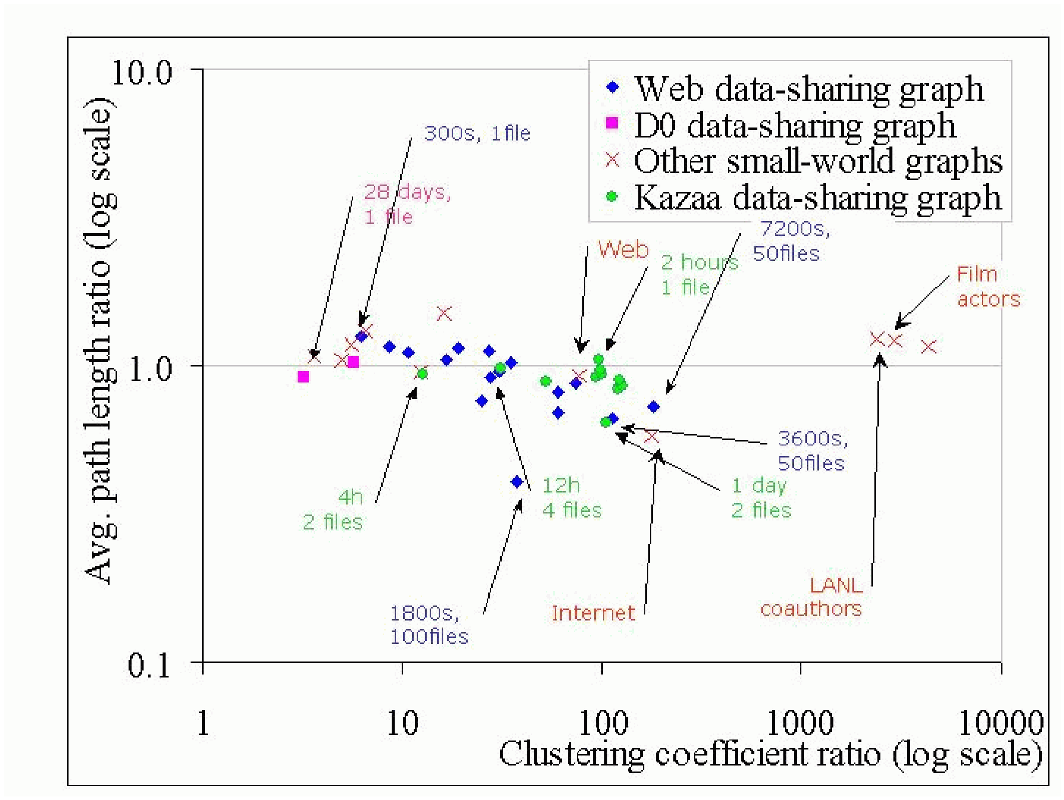


Figure 3.20: Small-world networks: data-sharing graphs and networks previously documented in the literature as small worlds.

3.5 Human Nature or Zipf's Law?

We observed small-world patterns in three different file-sharing communities: a scientific collaboration, the Web, and the Kazaa peer-to-peer system. Given the variety of our study sample, we could perhaps generalize this observation to any file-sharing user community. Thus, we seek to understand what causes these characteristics in data-sharing graphs and to answer the question:

Q5 *Are the small-world characteristics consequences of previously documented patterns or do they reflect a new observation concerning user's preferences in data?*

We explore two directions that help us answer the causality question. In Section 3.5.1 we focus on the definition of the data-sharing graph and question the large clustering coefficient as a natural consequence of the graph definition. In Section 3.5.2 we analyze the influence of well-known patterns in file access, such as time locality and file popularity distribution.

3.5.1 Affiliation Networks

An affiliation network (also called "a preference network") is a social network in which the participants (*actors* in sociology terminology) are linked by common membership in groups or clubs of some kind. Examples include scientific collaboration networks (in which actors belong to the group of authors of a scientific paper), movie actors (in which actors belong to the cast of a certain movie), and board directors (in which actors belong to the same board).

Affiliation networks are therefore bipartite graphs: there are two types of vertices, for actors and respectively groups, and edges link nodes of different types only (Figure 3.21, left). Affiliation networks are often represented as unipartite graphs of actors joined by undirected edges that connect actors in the same group. One observes now that the data-sharing graph with one-shared file threshold for the similarity criterion is such a one-mode projection of a bipartite affiliation network (Figure 3.21, right).

These one-mode projections of bipartite graphs have particular characteristics. Most relevant to this discussion is the clustering coefficient: inherently, the clustering coefficient is larger in these graphs than in random graphs of the same size, since the members of a group will form a complete subgraph in the one-mode projection. Consequently, our comparison with random graphs, although faithful to the Watts-Strogatz definition of small worlds, is misleading.

We therefore identified two possible sources of bias in our analysis: one is the implicitly large clustering coefficient of the unimodal affiliation networks, as just shown. Another is the degree distribution of the data-sharing graphs which, as in many other real networks, is far from the Poisson distribution of a random graph (Figures 3.14, 3.15, and 3.16).

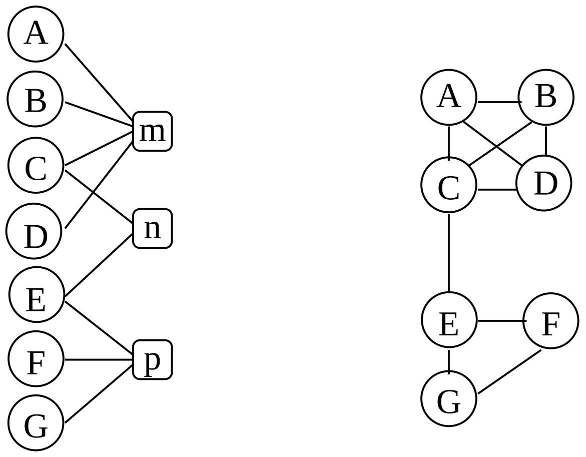


Figure 3.21: A bipartite network (left) and its unipartite projection (right). Users A-G access files m-p. In the unipartite projection, two users are connected if they requested the same file.

Newman et al. [88, 87] propose a model for random graphs with given degree distributions. These graphs, therefore, will not be random in the Erdős-Rényi sense, but will be random members of a class of graphs with a fixed degree distribution. The authors also adapt their model to affiliation networks and deduce a set of parameters of their unimodal projection. We use their theoretical model to estimate the clustering coefficient of unimodal projections of random affiliation networks of the size and degree distributions as given by traces and compare it with the actual values.

In a bipartite affiliation network, there are two degree distributions: of actors (to how many groups does an actor belong) and of groups (how many actors does a group contain). Let us consider a bipartite affiliation graph of N actors and M groups. Let us name p_j the probability that an actor is part of exactly j groups and q_k the probability that a group consists of exactly k members. In order to easily compute the average node degree and the clustering coefficient of the unipartite affiliation network, Newman et al. use three functions f_0, g_0, and G_0 defined as follows:

$$f_0(x) = \sum_{j=1}^{N} p_j x^j \tag{3.6}$$

$$g_0(x) = \sum_{k=1}^{M} q_k x^k \tag{3.7}$$

$$G_0(x) = f_0(g_0'(x)/g_0'(1)) \tag{3.8}$$

The average degree for the actors' one-mode projection of the affiliation network is:

$$AvgDegree = G_0'(1) \tag{3.9}$$

And the clustering coefficient is:

$$C = \frac{M}{N}\frac{g_0'''(1)}{G_0''(1)} \tag{3.10}$$

The definition of the clustering coefficient is that of Eq. 3.3.

It is therefore relevant to compare the clustering coefficient of data-sharing graphs with that given by Equation 3.10.

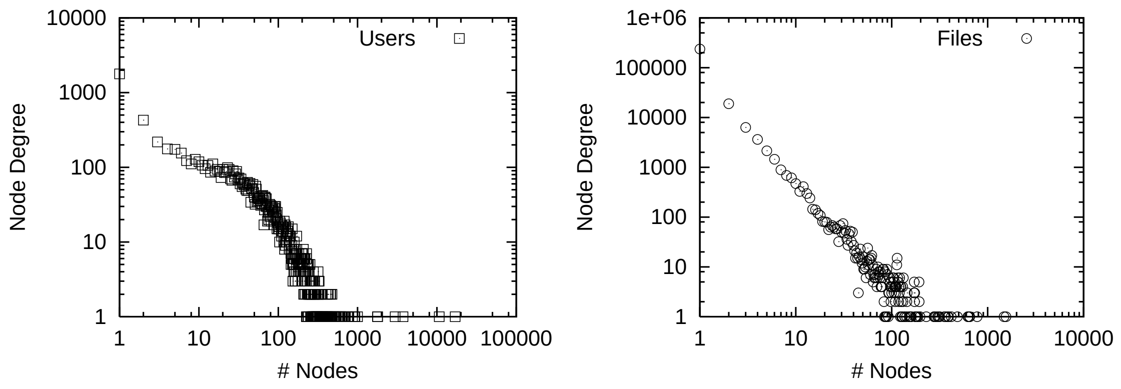


Figure 3.22: Degree distribution of user (left) and file (right) nodes of a bipartite affiliation network corresponding to a half-hour interval in the Boeing Web traces.

Figure 3.22 shows the corresponding values for the degree distribution p and q (but not normalized: i.e., it shows the number rather than the percentage of users that requested exactly k files) in a Web data-sharing graph with a similarity criterion of one shared request within a half-hour interval.

Table 3.3 shows that our intuition was correct: there is a significant difference between the values of measured and modeled parameters. Thus, the large clustering coefficient is not due to the definition of the data-sharing graph as a one-mode projection of an affiliation network with non-Poisson degree distributions.

Table 3.3 leads to two observations. First, the actual clustering coefficient in the data-sharing graphs is always larger than predicted and the average degree is always smaller than predicted. An interesting new question emerges: what is the explanation for these (sometimes significant) differences? One possible explanation is that user requests for files are not random: their preferences are limited to a set of files, which explains the actual average degree being smaller than predicted. A rigorous understanding of this problem is left for future work.

A second observation is that we can perhaps compare the file sharing in the three communities by comparing their distance from the theoretical model. We see that

Table 3.3: Properties of data-sharing graphs, measured and modeled as unimodal projection of affiliation networks. Clustering coefficient are measured using Eq. 3.3 and modeled using Eq. 3.10

				Clustering		Average degree	
	Interval	Users	Files	Theory	Measured	Theory	Measured
D0	7 days	74	28638	0.0006	0.65	1242.5	3.3
	28 days	151	67742	0.0004	0.64	7589.6	6.0
Web	2 min	3385	39423	0.046	0.63	50.0	22.9
	30 min	6757	240927	0.016		1453.1	304.1
Kazaa	1 h	1629	3393	0.55	0.60	2.9	2.4
	8 h	2497	9224	0.30	0.48	9.5	8.7

the Kazaa data-sharing graphs are the closest to the theoretical model and the D0 graphs are very different from their corresponding model. This is different from the comparison with the Erdős-Rényi random graphs (Table 3.2). The cause of this difference and the significance of this observation remain to be studied in the future.

3.5.2 Influences of Zipf's Law and Time and Space Locality

Event frequency has been shown to follow a Zipf distribution in many systems, from word occurrences in English and in monkey-typing texts [72] to city population [78]. It is also present in two of the three cases we analyze: the Web and Kazaa. Other patterns characteristic to data access systems include time locality, in which an item is more popular (and possibly requested by multiple users) during a limited interval and temporal user activity, meaning that users are not uniformly active during a period, but follow some patterns (for example, downloading more music files during weekends or holidays [98]). Thus, we ask:

Q6 Are the patterns we identified in the data-sharing graph, especially the large clustering coefficient, an inherent consequence of these well-known behaviors?

To answer this question, we generate random traces that preserve the documented characteristics but break the user-request association. From these synthetic traces, we build the resulting data-sharing graphs, and analyze and compare their properties with those resulting from the real traces.

Synthetic Traces

The core of our traces is a triplet of user ID, item requested and request time. Figure 3.23 identifies the following correlations in traces, some of which we want to preserve in the synthetic traces:

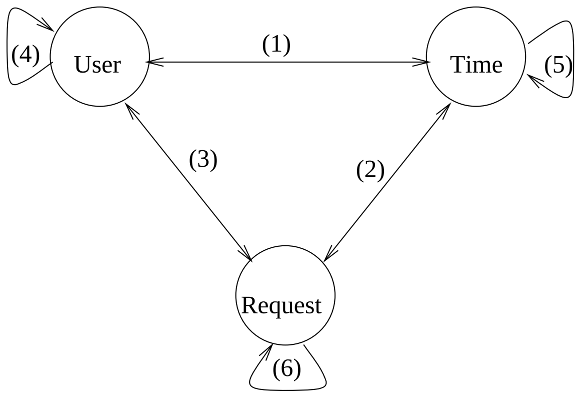


Figure 3.23: The relations between users, their requests, and their request times determine observed patterns like Zipf frequency of requests or time locality.

(1) User–Time: User's activity varies over time: for example, in the D0 traces, some users accessed data only in May.

(2) Request–Time: Items may be more popular during some intervals: for example, news sites are more popular in the morning.

(3) User–Request: This is the key to user's preferences. By breaking this relationship and randomly recreating it, we can analyze the effect of user preferences on the properties of the data-sharing graph.

(4) User: The number of items requested per user over the entire interval studied may be relevant, as some users are more active than others (see Figure 3.7 for the Web traces).

(5) Time: The time of the day (or in our case, of the periods studied) is relevant, as the Web traces show (the peak in Figure 3.5 right).

(6) Request: This is item popularity: number of requests for the same item.

Our aim is to break the relationship (3), which implicitly requires the break of (1), (2), or both. We also want to preserve relationships (4), (5), and (6).

One can picture the traces as a $R \times 3$ matrix, in which R is the number of requests in that trace and the three columns correspond to users, files requested, and request times, respectively. Now imagine the we shuffle the users column while the other two are kept unchanged: this breaks relations (3) and (1). If the requests column is shuffled, relations (3) and (2) are broken. If both user and request columns are shuffled, then relations (1), (2), and (3) are broken. In all cases, (4), (5), and (6) are maintained faithful to the real behavior: that is, users ask the same number of requests (4); the times when requests are sent are the same (5); and the same requests are asked and repeated the same number of times (6).

We generated synthetic traces in three ways, as presented above:

ST_1: No correlation related to time is maintained: break relations (1), (2), and (3).

ST_2: Maintain the request times as in the real traces: break relations (1) and (3).

ST_3: Maintain the user's activity over time as in the real traces: break (2) and (3).

Properties of Synthetic Data-Sharing Graphs

Three characteristics of the synthetic data-sharing graphs are relevant to our study. First, the number of nodes in synthetic graphs is significantly different than in their corresponding real graphs ("corresponding" in terms of similarity criterion and time). On the one hand, the synthetic data-sharing graphs for which user activity in time (relation (1)) is not preserved have a significantly larger number of nodes. Even when the user activity in time is preserved (as in the ST_3 case), the number of nodes is larger: this is because in the real data-sharing graphs, we ignored the isolated nodes and in the synthetic graphs there are no isolated nodes. On the other hand, when the similarity criterion varies to a large number of common requests (say, 100 in the D0 case, Figure 3.25), the synthetic graphs are much smaller or even disappear. This behavior is explained by the distribution of weights in the synthetic graphs (Figure 3.24): compared to the real graphs (Figure 3.11), there are many more edges with small weights. The median weight in the real D0 data-sharing graphs is 356 and the average is 657.9, while for synthetic graphs the median is 137 (185 for ST_3) and the average is 13.8 (75.6 for ST_3).

Second, the synthetic data-sharing graphs are always connected (unlike real graphs, that always have multiple connected components, as shown in Table 3.2). Even for similarity criteria with large number of common requests the synthetic graphs remain connected. This behavior is due to the uniform distribution of requests per user in the case of synthetic traces, which is obviously not true in the real case.

Third, the synthetic data-sharing graphs are "less" small worlds than their corresponding real graphs: the ratio between the clustering coefficients is smaller and the ratio between average path lengths is larger than in real data-sharing graph (Figure 3.26). However, these differences are not major: the synthetic data-sharing graphs would perhaps pass as small worlds.

These results show that user preferences for files have significant influence on the data-sharing graphs: their properties are not induced (solely) by user-independent trace characteristics, but human nature has some impact. So perhaps the answer to this section title ("Human nature or Zipf's law?") is "Both." However, it seems that identifying small-world properties is not a sufficient metric to characterize the natural interest-based clustering of users: we might need a metric of how small world

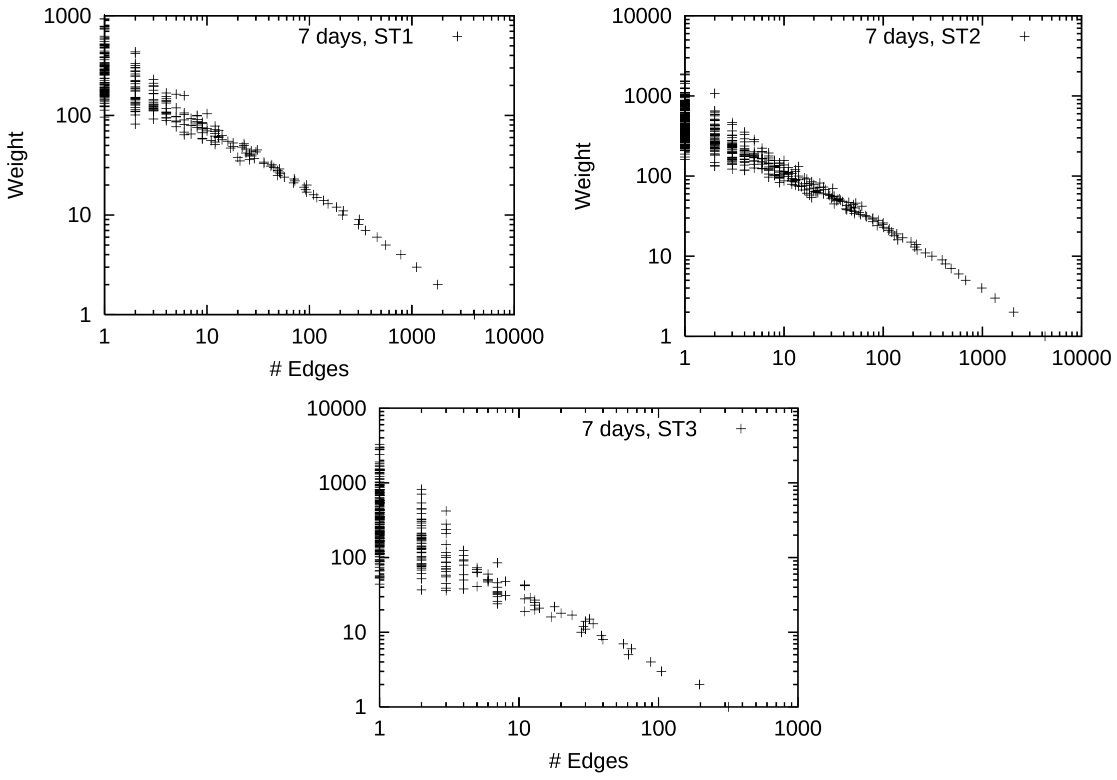


Figure 3.24: Distribution of weights in the synthetic data-sharing graphs built from shuffling the D0 traces.

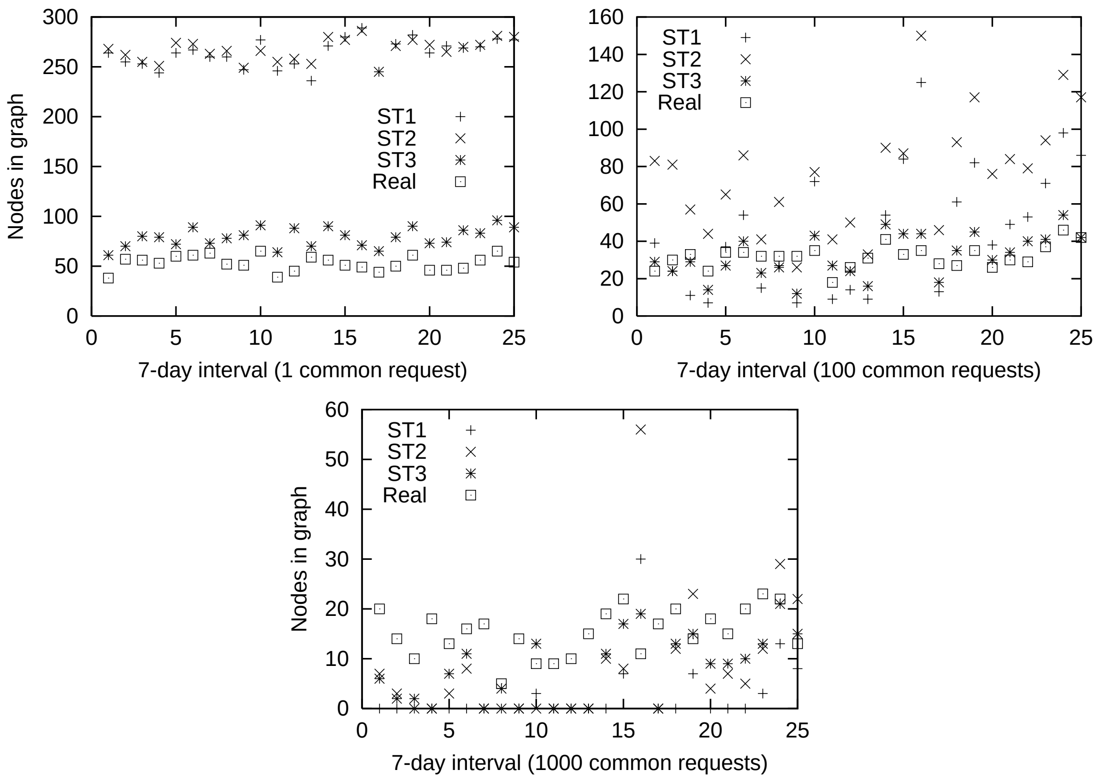


Figure 3.25: Number of nodes in data-sharing graphs in real and synthetic D0 traces

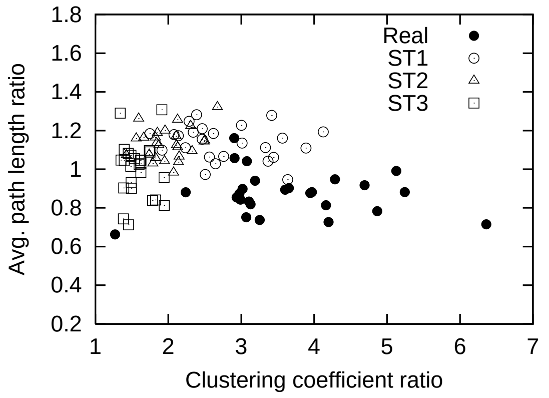


Figure 3.26: Comparison of the small-world data-sharing graphs as resulted from the real and synthetic D0 traces.

a small-world data-sharing graph is. This problem remains to be studied further in the future.

3.6 Small-World Data-Sharing Graph: Significance for Mechanism Design

It is interesting to notice that the structure we call the data-sharing graph can be applied at various levels and granularities in a computing system. We looked at relationships that form at the file access level, but intuitively similar patterns could be found at finer granularity, such as access to same memory locations or access to same items in a database. For example, a recent article [104] investigates the correlation of program addresses that reference the same data and shows that these correlations can be used to eliminate load misses and partial hits.

At a higher level, the data-sharing graph can identify the structure of an organization—based on the applications its members use, for example—by identifying interest-based clusters of users and then use this information to optimize an organization's infrastructure, such as servers or network topology.

In this section we focus on implications for mechanism design of the data-sharing graph from two perspective: its structure (definition) and its small-world properties. We stress that some of these ideas, while untested, are promising directions for future work. In Chapters 4 and 5 we present the design and performance of a mechanism inspired from the small-world characteristics of the data-sharing graph.

3.6.1 Relevance of the Data-Sharing Graph Structure

Some recommender systems have a similar flavor to the data-sharing graph. ReferralWeb [62] attempts to uncover existing social networks to create a referral chain of named individuals. It does this by inferring social relationships from web pages, such as co-authorship, research groups and interests, co-participation in discussion panels, etc. This social network is then used to identify experts and to guide searches around them.

Sripanidkulchai et. al came close to the intuition of the data-sharing graph [111]: they improve Gnutella's flooding-based mechanism by inserting and exploiting interest-based shortcuts between peers. Interest-based shortcuts connect a peer to peers who provided data in the past. This is slightly different from our case, where an edge in the data-sharing graph connects peers that requested the same data. However, the two graphs are likely to overlap significantly if peers store data of their own interest. Our study distinguishes by its independence from any underlying infrastructure (in this case, the distribution of data on peers and the location mechanism) and gives a theoretical explanation of the performance improvements in [111].

The data-sharing graph can be exploited for a variety of decentralized file management mechanisms in resource-sharing systems (such as peer-to-peer or Grids).

- In a writable file-sharing system, keeping track of which peers recently requested a file facilitates the efficient propagation of updates in a fully decentralized, self-organizing fashion (a similar idea is explored in [102]).
- In large-scale, unreliable, dynamic peer-to-peer systems file replication may be used to insure data availability [94] and transfer performance. The data-sharing graph may suggest where to place replicas closer to the nodes that access them. Similarly, it may be useful for dynamic distributed storage: if files cannot be stored entirely on a node, then they can be partitioned among the nodes that are interested in that file.
- In a peer-to-peer computing scenario, the relationships between users who requested the same files can be exploited for job management. If nodes store and share recently downloaded files, they become good candidates for running jobs that take those files as input. This can be used for scheduling, migrating or replicating data-intensive jobs.

3.6.2 Relevance of Small-World Characteristics

The idea underlying the data-sharing graph was first presented in [58] as a challenge to design a file-location mechanism that exploits the small-world characteristics of a file-sharing community. Meanwhile we completed the design and evaluation of a mechanism that dynamically identifies interest-based clusters, disseminates location information in groups of interested users, and propagates requests among clusters (Chapters 4 and 5). Its strengths come from mirroring and adapting to changes in user's behavior. File insertion and deletion are low cost, which makes it a good candidate for scientific collaborations, where use of files leads to creation of new files.

3.7 Summary

This chapter reveals a predominant pattern in diverse file-sharing communities, from scientific communities to the Web and file-swapping peer-to-peer systems. This pattern is brought to light by a structure we propose and that we call "data-sharing graph." This structure captures the relationships that form between users who are interested in the same files. We present properties of data-sharing graphs from three communities. These properties are relevant to and might inspire the design of a new style of mechanisms in peer-to-peer systems, mechanisms that take into account, adapt to, and exploit user's behavior. We also suggest some mechanisms that could benefit from the data-sharing graph and its small-world properties.

CHAPTER 4
FLASK: A FILE-LOCATION ALGORITHM FOR SMALL-WORLD COMMUNITIES

The preceding chapter shows that file-sharing communities represented as data-sharing graphs have small-world characteristics. The large clustering coefficient reveals that groups of users tend to be interested in the same sets of files and the small average path length shows that the distance between two users on the data-sharing graph is, on average, small.

Based on these observations, we designed a file-location mechanism that takes advantage of the naturally occurring small-world patterns. This mechanism, FLASK (a **F**ile-**L**ocation **A**lgorithm for **S**mall-world **K**ommunities) is presented in this chapter.

4.1 FLASK: Yet Another File-Location Mechanism

With the success of Napster and Gnutella technologies, significant research effort was concentrated on designing more sophisticated (albeit less successful in terms of wide acceptance) file-location mechanisms. This work has produced dozens of variants of distributed hash tables, several improvements in the Gnutella flooding protocol, and many other proposed mechanisms for locating files in unstructured networks. Why bother to design yet another?

4.1.1 Motivation: Different Requirements

We can characterize existing P2P location mechanisms according to their performance objectives and tradeoffs. In Gnutella, the emphasis is on accommodating highly volatile peers and on fast file retrieval, with no guarantees that files will always be located. In Freenet [28], the emphasis is on ensuring anonymity. In contrast, distributed hash tables (DHTs) such as CAN [95], Chord [112], Pastry [101], and Tapestry [122] guarantee that files will always be located at the cost of increased overhead for file-insertion and removal and lack of support for wildcard searches.

All these systems were built without reference to a target community: mainly because they were the “pioneers” in the peer-to-peer file-sharing domain, they do not address a specific user community or exploit user behavior, but rather, as was the case with Gnutella and Napster, *create* a user community. Over time, this user community was analyzed from various perspectives and important lessons were learned on participation [103, 18], free riding [4], query distribution [110, 71], overlay topology characteristics [98], and interest groups [59, 60].

Our approach is different: we studied a set of user communities and observed significant requirements not addressed by existing solutions. We detail these requirements below, and note that many are associated with scientific communities. For example, scientists tend to request publishing control over their own data, that is, they want to ensure that the files they share are advertised as they want. This makes solutions that rely on other nodes to advertise on their behalf (such as the DHT solutions) inappropriate. In addition, usage of scientific data often results in creation of new data that is of interest to the group to which the data producer belongs. For example, members of a science group access newly produced data to perform analyses or simulations. This work may result in new data that will be of interest to all scientists in the group, e.g., for comparison. Furthermore, not only do scientific communities introduce requirements (fast publication, local control) atypical of P2P music-sharing communities, but in fact some features of P2P systems are undesirable. For example, the desire for anonymity of access in scientific collaborations may be overruled by the need for auditing of access for security and monitoring reasons.

Not only are user requirements different, but, as observed in the previous chapter, some characteristics of user behavior can be exploited in mechanism design. These observations justify the design of yet another file-location mechanism.

A file-location mechanism appropriate for a variety of communities, including the scientific community, requires the following:

1. **Low-cost file insertion/removal**. Data usage in scientific communities is different than in, for example, music sharing environments: data usage often leads to creation of new files, inserting a new dimension of dynamism into an already dynamic system. This requirement is better served by unstructured search mechanisms (such as Gnutella) than by DHTs: in the latter the cost of inserting and deleting a file is the cost of a lookup.

2. **Support for node volatility**. One of the significant properties of P2P networks, revealed by many system characterization studies [103, 18], is highly intermittent node participation. Support for file volatility may be provided as self-configuring overlays, as in many current P2P solutions.

3. **Scalability with the number of files**: Among the scientific domains that have expressed interest in building data-sharing communities are physics (e.g., GriPhyN project [52]), astronomy (Sloan Digital Sky Survey project [107]) and genomics [54]. The Large Hadron Collider (LHC) experiment at CERN is a proof of the physicists' interest and pressing need for large-scale data-sharing solutions. Starting 2005, the LHC will produce Petabytes of raw data a year that needs to be pre-processed, stored and analyzed by teams comprising thousands of physicists around the world. In this process, even more derived data will be produced. Hundreds of millions of files will need to be managed, and storage at hundreds of institutions will be involved.

4. **Heterogeneity**. Resource capabilities are likely to be highly variable: storage space, network latency and bandwidth, fault tolerance and system load are likely to differ from node to node and to vary over time.

5. **Publishing control** refers to the ability of nodes to advertise or make accessible location information about the files they provide to the community. Publishing control is offered by many solutions. The DHT-based solutions trade off this ability for low file retrieval by (sometimes unsustained) assumptions about node cooperation.

6. **Support for collections**. A typical scenario in scientific collaborations is imposed by data-intensive applications: a data-analysis job takes a set of files as input data. In the D0 experiment, the average number of files per data-intensive job submitted is 117. Ideally, the location latency of a collection of files should depend sublineary on the size of the collection.

7. **Support for approximate matches and keyword searches**. This is a requirement often expressed for music-sharing systems, but it is a valid requirement for scientific communities, as well: for example, if data description is included in the filename (as musician's name is often included in a digital song title), keyword searches are useful. Keyword searches are not naturally provided by DHT-based algorithms.

4.1.2 Intuition

The small-world characteristics presented in Figures 3.17, 3.18, 3.19 and 3.20 describe a particular graph topology with two properties: first, the graph contains highly connected subgraphs; second, the average distance between nodes is small. Since the highly connected subgraphs (that we refer to as *clusters*) are formed around common data interests, this topology suggests new search methods that combine information dissemination and request propagation techniques. Specifically, our idea is to build an overlay that mirrors common user interests, to determine clusters of interest, to disseminate file location information to users with common interests and to propagate requests to different groups of interests. Figure 4.1 presents this intuition.

We distinguish three main components in FLASK:

1. **Overlay construction** refers to creating and maintaining connections between nodes that satisfy a similarity criterion. Basically, this requires mechanisms that allow users with similar interests to learn about each other without relying on centralized control or global information. Thus, the overlay must mirror the data-sharing graph and adapt to changing user behavior.

2. **Cluster identification**. Once the overlay is in place, FLASK needs to identify groups of users with common interests. The existence of these groups (or

clusters) is proven by the large clustering coefficient.

3. **File location**. Two components are necessary for locating files: information dissemination within clusters and request propagation among clusters.

   Assuming a probabilistic information dissemination technique such as gossip [63], file location information can be spread reliably and quickly to all members of the cluster. Therefore, requests for files known by nodes in the group are solved with one local lookup. If the answer is not found locally, then with high probability it does not exist in the local cluster either: the request should be sent to different clusters.

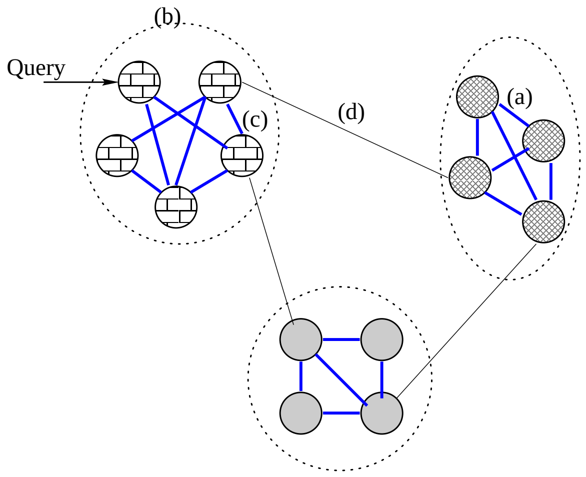


Figure 4.1: Small worlds seen as loosely connected collections of well connected clusters. FLASK components: 1) overlay construction (a); 2). Cluster identification (b); 3). File location, with its two parts: information dissemination within clusters (c) and request propagation among clusters (d)

There are multiple sources of dynamism in a file-sharing community: intermittent user (and therefore, resource) participation; changing user behavior, such as interest in data and frequency of requests; and variation in the set of shared files, by file insertion and deletion. Referring to Figure 4.1, taking into account these dynamic characteristics requires a continuous rewiring of the graph and reconsidering of clusters over time.

4.1.3 Related Ideas

While we are not aware of previous work that has looked at communities of users the way we do—by connecting users with the same interests in a graph—there are various ideas that partially overlap with the main ideas in FLASK.

One class of relevant ideas is related to optimizing the overlay in unstructured networks based on usage or interest. Two results are worth mentioning. Sripanidkulchai et al. [111] propose a mechanism that improves Gnutella's flooding-based search by adding shortcuts between peers. The motivating assumption is interest locality: if node A found some data on node B, it is likely that node B holds some other data that A will be interested in, so queries are first propagated along shortcuts; if unsuccessful, they rely on the typical Gnutella flooding. This idea leads to an overlay different from the data-sharing graph: in a data-sharing graph A would connect to B only if both are interested in the same files.

However, Sripanidkulchai et al. have not verified the motivating assumptions (i.e., if A is interested in a file on B, it is also interested in other files on B) on real data, presumably because of the challenges related to collecting information on where files are stored in today's P2P systems. The experimental setup that makes up for this lack of real traces introduces some side effects that build the very data-sharing graph. The mapping of files onto user storage is built via replication: after a successful query, a node downloads the file and makes it public. In this scenario, a node stores precisely the files in which it expressed interest: therefore, the connection between A and B exists in both overlays. The resulting overlay from [111] is therefore an overlap of the Gnutella overlay (the graphs collected and studied in [98]) and the data-sharing graph. The graph formed of shortcuts only is a sparser data-sharing graph (the number of shortcuts per node is limited to 10) defined over one-hour intervals (according to the experimental setup) with a similarity criterion of one shared file.

The improvements introduced by shortcuts are significant: 85% of queries can be solved using shortcuts only. However, it is unclear how much of this figure is due to the data-sharing graph, how much to file replication, and how much to the optimistic experimental assumptions of infinite caches: e.g., a node that asks for all files in the system will end up storing copies of all files on its local disk and will help any request coming from any neighbor.

Cohen et al. [29] propose another approach to guiding search in unstructured overlays: nodes connect to other nodes that *store* the same data, in an attempt to adapt the market-basket idea to P2P. The market-basket concept was determined on empirical evidence that shows that those who buy diapers (have small kids at home, therefore cannot go out, therefore) also buy beer. Translated to the P2P context, the market-basket idea shows that a node that stores beer is likely to store diapers, and a node that looks for diapers and stores beer would likely find diapers at some neighbor who stores beer.

Note that both ideas ([111, 29]) have in common the assumption that there exist a relationship among the files stored on a node: users do not *store* random things on their disks. We proved that users do not *ask* for random things: the proof that they do not store random things requires the assumption that users store only what they are interested in.

Three significant differences may be noted between our idea and those discussed above. First, we base the FLASK design on studies of user communities: that is, we support our assumptions with analysis of real file-sharing communities. Second, the pattern we exploit is based solely on users' expressed interests in data, totally independent of where those data are stored. Third, in FLASK we combine information dissemination with request propagation: the study of real user communities suggests where to disseminate information. The two approaches mentioned above focus solely on requested propagation. Consequently, the observations on user behavior are used to guide request propagation.

Combining information dissemination with query propagation is an idea that FLASK shares with other systems. Kelips [53] is a location mechanism in which groups (or clusters in our terminology) are formed and information is disseminated in groups, while requests are propagated among groups. The important difference is how clusters are formed. In Kelips, they are independent of user behavior: nodes are placed in clusters based on their IDs, as in a classical hash table.

Yet another aspect of FLASK is the differentiated activities for in-cluster and between-cluster communication. Similar ideas are found in various epidemic communication mechanisms, such as multi-level gossip [116, 64].

4.2 FLASK: Components

The FLASK algorithm follows the intuition provided by the data-sharing graph and its small-world characteristics. Nodes self-organize in an overlay that mirrors the data-sharing graph: thus, two nodes maintain an open connection if they requested at least μ common files within an interval of duration τ. How this overlay is built in a decentralized manner will be presented in Section 4.3. Nodes may contribute files to the community: they maintain an up-to-date summary of the files stored locally. Let us assume peers can identify their own group of interest and therefore distinguish between peers in their group and peers outside their group (how this can be obtained is presented in Section 4.4). Periodically, nodes exchange file location information with nodes in their own interest group. The information they share contains lists of filenames and the nodes that provide them. This information may include a (sub)list of locally stored files as well as remote files recently discovered. Soft-state mechanisms ensure that old information is erased: information received more than τ ago is considered outdated and discarded.

Periodically, nodes reevaluate their relationships with their neighbors to adapt to changes in interests: for example, they may decide to disconnect from some neighbors or make new connections to other peers. They also reevaluate the interest group, to accommodate the new neighbors, for example.

Nodes maintain the information received from peers in a local database. When a node receives a request, it searches its local database. Because of the information dissemination mechanism used, with high probability all nodes in a cluster maintain the same database. Therefore, if the answer is not found in the local database, with high probability it does not exist in the cluster: that is, the requested file is not stored on any of the nodes in the group and none of the nodes knows where that file is located. The request needs to be forwarded to a different cluster. Otherwise, if the answer is found locally, then it is returned to the user.

A node who wants to join the network learns of one or more nodes from out of band sources (for example, a web site as in Gnutella) and contacts them. If the connection is accepted, it will consider its new neighbors as being in the same cluster of interests and exchange information. Over time, assuming it does send its own requests for files, the new node may discover other nodes that better match its own data interests and will connect with those.

The following components are the building blocks in FLASK:

1. Overlay construction.

2. Cluster identification.

3. Information dissemination and request propagation.

Each of these components is described in the following sections. The benefits and costs of information dissemination in clusters of interest are studied in Chapter 5.

4.3 Overlay Construction

The overlay construction component deals with building the data-sharing graph on the fly in a decentralized manner: we need a mechanism to let users with the same interests learn of each other in a decentralized, adaptive manner.

We propose a solution that uses the data storage nodes as meeting points for users with the same interests in data. Assume a node A requests file F and, via the file location mechanism, learns that F is stored on node N. When A goes to *fetch* file F from N, N will record A's interest in its file and the time of the download. If user B then fetches the same file within a time shorter than an aging time, N informs B about A's interest in the same file F (see Figure 4.2). Thus, B can contact A if interested in connecting to new peers.

This basic idea can be expanded to fit the similarity criterion of the data sharing graph. For example, B may choose to contact A only if more than μ common files have been requested with an interval τ. The values μ and τ can be adapted by each node independently to satisfy local objectives: more or fewer connections, better interest overlap, etc. This approach has significant advantages: it allows for true adaptability to node heterogeneity relating, for example, to communication capabilities or high load/reduced processing power.

Given the documented Zipf distribution of file popularity in many systems, a relevant problem is that of avoiding hot spots when building the data-sharing overlay:

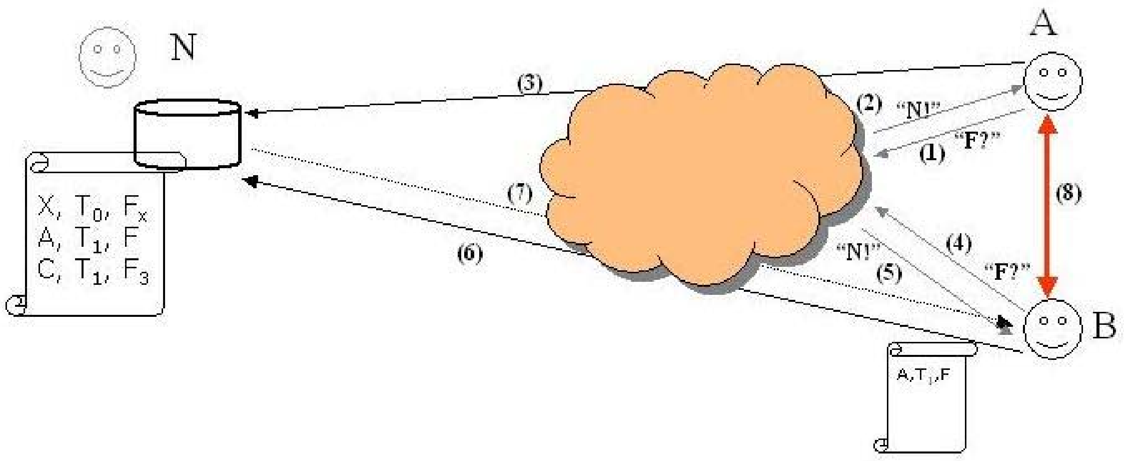


Figure 4.2: Overlay construction: (1) Node A requests file "F"; (2) A receives answer that "F" is stored on node N; (3) A contacts N to fetch file. N logs A's request and time; (4) Node B requests file "F"; (5) B receives answer that "F" is stored on node N; (6) B contacts N to fetch file; (7) N sends the relevant log with latest requests for file "F": this is how B learns of A.

for example, if all users check the same news sites in the morning, they will be considered (incorrectly, perhaps) as having common data interests. One way to overcome this problem has already been suggested: use a similarity criterion that has a larger number of shared files μ. However, this approach may eliminate relevant sharing information, while remaining inefficient: most users might go to `nytimes.com`, `yahoo.com` and `google.com` first thing in the morning, so $m = 2$ would not help. One solution is to weight the sharing of files by their popularity: node N will not announce F as a shared file if it is highly popular within last τ. Alternatively, it will announce it, but will also send the number of requests r for F, such that the highly popular files F_i will contribute only with $\frac{1}{r_i}$ to the number of files shared between two nodes.

There are some issues raised by this overlay construction solution:

1. How large do the logs at the storage node become? The log size is determined by the number and popularity of the files stored locally. While the number of files stored per node is not provided in traces, file popularity provides some useful insights. In D0, for example, the most popular files are requested 271 times over the 6-month interval. In the Web, this number is just under 50000 (for 24 hours), while in Kazaa it is almost 10000 (for five days). At the highest activity peak in the Web traces, the most popular file during a half-hour interval is requested 13272 times. However, given the Zipf distribution of file popularity, few files are requested many times: out of more than 12000 files requested, seven

are requested more then 1000 times and about 10% are requested more than 100 times.

These numbers do not provide a satisfactory estimation of the storage costs required. While our intuition is that these costs are not exaggerated, a more detailed estimation is necessary.

2. Effects of multiple file replicas. A common approach to improve reliability and performance is to replicate data. However, with multiple replicas of a file, the central meeting point for nodes interested in that file is lost: for example, node A might fetch file F from node N while node B might have discovered a replica of file F on node M. In this scenario, A and B do not learn of each other. However, we argue that this situation is unlikely to be maintained extensively in a system with high user activity: the nodes may eventually meet via other files of common interest.

The overlay construction mechanism presented in this section uses storage nodes as meeting points for users interested in the same files. It is, hence, fully decentralized (since storage is distributed) and adaptive to changes in user interests. While a more careful analysis of the mechanism is necessary (and is part of our future work plans), the mechanism is sound enough to provide a basis for understanding the potential benefits of information dissemination.

4.4 Clustering

The problem of detecting "clusters" has been studied for long and has produced a large body of literature [61]. Clustering is the problem of discovering natural groups in data sets by partitioning N given data points into a number of clusters, such that points within a cluster are more similar to each other that to points from different clusters. The problem is difficult because of the challenges imposed by defining similarity metrics and their dependence on application. Similarity between data points can often be represented as a (weighted) graph.

Of the many clustering techniques previously proposed, we discuss solutions to two problems very similar to our context. The first example refers to inferring communities on the web graph. The web graph consists of web pages connected via hyperlinks. Flake et al. propose a method based on maximum cut flow to identify "communities" of pages [37, 38]: a community is a set of web pages that link (in either direction) to more web pages in the community than to pages outside the community. Their solution requires the identification of two well connected nodes, one acting as a source and the other as a sink. Moreover, these nodes need to have some special properties: the source node must be a known member of the community that is to be identified, while the sink needs to be a hub, for example a set of web portals. These requirements are hard to meet in a dynamic graph such as the data-sharing graph.

Another relevant problem is to identify communities of scientists connected in a co-authorship graph. Girvan and Newman [47] use edge betweenness to identify the edges that connect communities. The betweenness of an edge is the number of shortest paths that contain that edge. The algorithm repeatedly removes the edge with the highest betweenness until a satisfactory partition is obtained. However, the algorithm is prohibitively expensive ($O(N^3)$) and requires global information on the graph.

Many other related solutions have been proposed in various contexts, such as web communities [67] and recommendation systems [73]. The particular challenges of our context limit us to solutions that use only local knowledge about the graph topology. However, one can observe from the intuition presented in Section 4.1.2 that a node only needs to know to which neighbors to gossip and to which to propagate unanswered requests. This simple observation reduces the clustering problem to an edge-labeling strategy: each node labels its own edges, based on local information only, as *long* or *short*. Nodes gossip along short edges and propagate requests along the long edges. Thus, a cluster is defined by a collection of nodes connected by short edges. Long edges maintain the connections between different clusters.

Multiple definitions can be proposed for long/short edges. We experiment with one definition that we call *triad labeling* and mirrors the small-world topology: the tightly connected nodes—which may have large clustering coefficients—are connected by short edges. We therefore define a short edge as an edge that is part of a triad (triangle in graph). In oder to avoid isolated nodes—one-node clusters—we also consider dead ends as short edges (Figure 4.3).

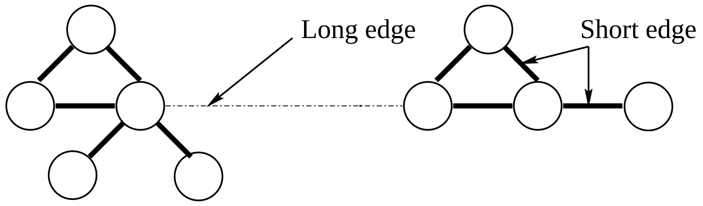


Figure 4.3: Triad labeling: edges in triangles and dead-ends are considered *short*, while others are considered *long*.

The formal definition of long/short edges in triad labeling is:

> *Definition*: An edge in graph is called short if it connects nodes that are part of a triad or if it is a dead end. Otherwise, an edge is called long.

Another simple idea for labeling is threshold labeling: in a weighted data sharing graph—where weights represent the number of requests common to two nodes—an edge is considered short if its corresponding weight is larger than a threshold value, otherwise it is long. The effectiveness of this approach, however, is highly sensitive to the choice of the threshold value, which is a non-trivial problem in our context. We therefore considered only the triad clustering method in our experiments.

4.5 Locating Files in Small-World Networks

The file-location component has two parts: information dissemination and request propagation. When a request is sent to a node, the node looks it up in its own index. If found, the answer is sent back to the requester. If not, the request is propagated to another cluster. Section 4.5.1 presents the information dissemination component while Section 4.5.2 discusses the request propagation component.

4.5.1 Information Dissemination

The basic idea in FLASK is offloading the request propagation component by disseminating *useful* information to *interested* parties. The previous sections presented a method to determine the *interested* parties: nodes in the same interest-based cluster. This section focuses on the remaining aspects of information dissemination: what information to disseminate (i.e., define *useful* information) and how to disseminate it.

Similar to identifying interested parties, identifying useful information is helped by the data-sharing graph and its small-world properties: the large weights in the weighted data-sharing graph show there is significant overlap in interests between pairs of users. Therefore, sharing *experience*—previously discovered file locations—may save nodes in the same cluster significantly many future searches.

As previously noted, the search space for request propagation is reduced to clusters: a request not answered is forwarded to another cluster. Therefore, if some file is not known on a node, it is assumed it is not known in that node's cluster. Consequently, in addition to disseminating recent experience, nodes also need to disseminate location information about the newly inserted files. This is necessary in order to support file insertion. Note that disseminating local summaries would not be necessary if FLASK fell back on another mechanism when searches fail. For example, experience sharing in interest-based clusters can be used to reduce the load on a central index: a newly inserted file never accessed before will not be known in the local cluster but can be located via the centralized mechanism.

In Chapter 5 we analyze separately the impact of disseminating recent experience (Section 5.2.2) and the impact of disseminating storage summaries (Section 5.2.3). We shall come back to the "what to disseminate" question after we discuss the "how to disseminate" aspect in the following.

Gossip Mechanisms

The basic mechanism for disseminating information in FLASK, gossip, is an epidemic communication mechanism that probabilistically guarantees information broadcast in a dynamic community of members.

Gossip protocols [64, 118] have been employed as scalable and reliable information dissemination mechanisms for group communication. Each node in the group knows a partial, possibly inaccurate set of group members. When a node has information to share, it sends it to f randomly chosen nodes in its set. A node that receives new information processes it (for example, combines it with or updates its own information) and gossips it further to f random nodes from its own set.

We use gossip protocols for two purposes: (1) to maintain accurate membership information in a potentially dynamic cluster and (2) to disseminate file location information to nodes in the local cluster. We rely on soft-state mechanisms to remove stale information: a node not heard about for some time is considered departed; a file not advertised for some time is considered removed.

For the membership mechanism, we employ SCAMP (Scalable Membership Protocol) proposed in [46]: nodes start with a small number of contact addresses (possibly neighbors in the data-sharing graph) and build up a partial view of the cluster membership of size $O((c+1)log(N))$, where N is the number of nodes in cluster and c is a mechanism parameter. A larger membership view is beneficial but not mandatory, since $log(N)$ acts as a performance threshold: the probability that a notification reaches everyone exhibits a sharp threshold if $log(N)$ or more members are known. Alternatively, smaller than $log(N)$ membership size hinders the performance of disseminating information to all nodes in cluster.

The membership algorithm and its analytical and experimental evaluations are presented in detail in [46]. Like any membership mechanism, SCAMP aims to collect and maintain information about the group members. It uses gossip techniques to disseminate membership updates and defines three algorithms in support of dynamic group membership: subscription, unsubscription, and failure recovery based on hearbeats.

A gossip-based information dissemination mechanism that uses SCAMP to maintain membership information has multiple advantages: first, it does not require complete membership information, which may be difficult to collect and provide in a decentralized, highly dynamic scenario. Second, it adapts fast to changes in membership. Finally, it is a scalable, decentralized scheme that probabilistically guarantees the dissemination of information to all group members. Each piece of information is gossiped by each node to f other nodes: thus, $Nlog(N)$ messages are communicated for the dissemination of one rumor. With high probability, assuming nodes gossip in parallel to one node in a round, the latency to propagate new information to all nodes is low ($O(log(N))$): experimental results presented in [63] show that the number of rounds necessary to reach all members is below 8 for $f \geq 8$ and a group of 20000 members.

Bloom Filters

Disseminating information may be costly in terms of storage and communication costs. In order to limit these costs, file location information can be compressed using Bloom filters.

Bloom filters [19] are compact data structures used for probabilistic representation of a set in order to support membership queries ("Is element x in set X?"). The cost of this compact representation is a small rate of false positives: the structure sometimes incorrectly recognizes an element as member of the set.

Consider a set A of n elements. Bloom filters encode A as a bit vector V of length m that can easily be tested for membership. For this, k hash functions are used: h_1, h_2, ..., h_k with $h_i : S \to 1..m$, where S is the element space.

The following procedure builds an m-bit Bloom filter for the set A by using k hash functions:

```
Procedure BuildBloomFilter(set A, hash_functions, integer m)
    filter = allocate m bits initialized to 0
    foreach ai  in A:
        foreach hash function hj:
            filter[hj(ai)] = 1
        end foreach
    end foreach
    return filter
```

Therefore, if $a_i \in A$, in the resulting Bloom filter V all bits corresponding to the hashed values of a_i are set to 1. Testing for membership of an element `elm` is equivalent to testing that all corresponding bits of V are set:

```
Procedure TestMembership (elm, filter, hash_functions)
    foreach hash function hj:
        if filter[hj(elm)] != 1 return False
    end foreach
    return True
```

One prominent feature of Bloom filters is that there is a clear tradeoff between the size of the filter and the rate of false positives. After inserting n keys into a filter of size m using k hash functions, the probability that a particular bit is still 0 is:

$$p_0 = (1 - \frac{1}{m})^{kn} \approx 1 - e^{\frac{kn}{m}} \tag{4.1}$$

In this case, perfect hash functions that spread the elements of A evenly throughout the space $1..m$ are assumed. In practice, good results have been achieved using MD5 and other hash functions [91].

The probability of a false positive (the probability that all k bits have been previously set) is:

$$p_{err} = (1 - p_0)^k = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-kn/m})^k \tag{4.2}$$

In Equation 4.2 p_{err} is minimized for $k = \frac{m}{n} ln2$ hash functions. In practice, however, only a small number of hash functions are used: the computational overhead of each additional hash function is constant while the incremental benefit of adding a new hash function decreases after a certain threshold (see Figure 4.4). For example, Bloom filters can be used to compress a set to 2 bytes per entry with false positive rates of less than 0.1% and lookup time of about 100 μs [97].

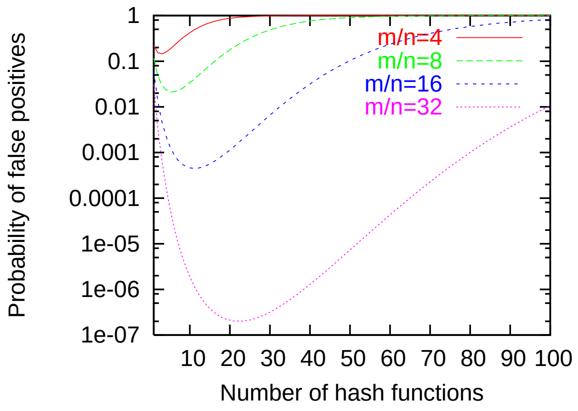


Figure 4.4: Bloom filter accuracy depends on the number of hash functions and the storage requirements.

Equation 4.2 is the base formula for engineering Bloom filters. It allows us to compute, for example, the minimal memory requirements (filter size) and the number of hash functions given the maximum acceptable false positives rate and number of elements in the set (as detailed in Figure 4.5).

$$\frac{m}{n} = \frac{-k}{ln(1 - e^{\frac{ln(p_{err})}{k}})} \tag{4.3}$$

A nice feature of Bloom filters is that they can be built incrementally: as new elements are added to a set, the corresponding positions are computed through the hash functions and bits are set in the filter. Moreover, the filter expressing the union of multiple sets is simply computed as the bit-wise OR applied over the corresponding filters.

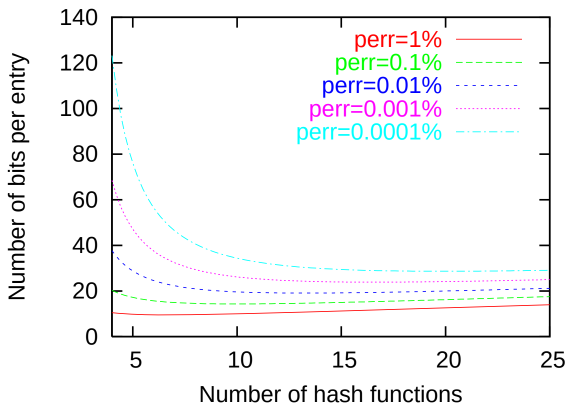


Figure 4.5: Tradeoffs computation-storage-accuracy for designing Bloom filters.

Gossiping about Files in FLASK

The mechanisms presented above, the gossip-based group communication and the Bloom filter-based data-compression technique, are the main ingredients of the information dissemination component in FLASK. Initially, nodes know only their neighbors in the data-sharing graph. They can also distinguish between “close” neighbors—nodes to which they are connected via short edges—and remote neighbors. The close neighbors are part of a node’s group of interest and comprise the initial view of the cluster membership.

Two types of information are collected from gossips: membership and file location. Each node sends tuples of the form:

```
<sender node IP address, storage node IP address, list of files>
```

The first field will add up to the membership information while the remaining two fields build up the file location database. The first field will always refer to nodes in the current cluster: this can be easily proven from the observation that gossips propagate along short edges initially, hence they collect information only from nodes that are connected via short edges.

In contrast, `storage node IP address` can be also from outside the current cluster, depending on what files are advertised: nodes may advertise own files as well as remote files of which they learned recently. If the local files are advertised, then the first two fields (`sender node IP address` and `storage node IP address`) are identical.

Upon the receipt of a gossip message, a node updates its local file database: if file lists are represented as Bloom filters, this is an low-cost bit-wise OR operation on lists of files from same storage node. To better support insertion and removal of files, an additional field containing a timestamp can be added to the tuple. This timestamp, with a valid value only for storage summaries of nodes in cluster (therefore, for which sender address = storage address), can be used to maintain information up to date with better granularity than τ. Since the timestamp is always emitted by the same node, it is sufficient information to identify the older data (that is, no additional time synchronizations are necessary).

As described in Section 4.5.1, nodes select random sets of nodes from the local view of the cluster membership and send them the updated file database (or just parts of it, for example only the new rumors).

The list of files can be compressed using Bloom filters or be left uncompressed, to support partial matches. Via gossip, all nodes learn with high probability of all files known within their cluster.

If Bloom filters are used, nodes need to know the following:

- The hash functions for encoding information. For simplicity, all nodes use the same hash functions. This is important to lookup queries in summaries sent from other nodes: the translation of the query into a Bloom filter (performed by the local node) and the summary against which it is compared (sent by a remote node) should be compatible.

- Consequently, nodes need to use the same values for m—the size of the Bloom filter. Since m depends on the desired fault positive upper limit and on the size of the set to be represented, an estimation of the expected number of files per node is needed.

These values are provided when nodes join the network.

4.5.2 Request Propagation

FLASK considerably diminished the influence of request propagation on performance: Chapter 5 will present trace-driven experimental results that show that more than 50% of requests can be answered based on cluster-disseminated information. For the remaining requests, however, FLASK relies on request propagation.

Propagating requests is trivial for nodes who maintain a long edge: the requests will be propagated along the long edge. This section discusses the case in which a node needs to propagate a request and has no long edge connecting it with a node in a different cluster.

A simple solution is to disseminate information on long-connections along with the membership information: nodes that maintain long edges are marked with a flag as “well-connected”. Poorly-connected nodes will therefore forward the requests they need to send outside the cluster to the well-connected nodes in the cluster, who will send them along the long edges. The well-connected nodes will therefore act as communication points between clusters. Sending a request to all well-connected nodes in the local cluster has clear advantages, as it initiates a flooding-like propagation of requests (faster and more robust against failure).

One problem, though, is maintaining the overlay connected. A graph-theory result shows that an average vertex degree larger than $ln(N)$ in a random graph with N nodes gives a connected graph [21]. Translated to our context, this would require that each node knows about $ln(N)$ nodes currently in the network, where N is the total number of nodes in FLASK (thus, not only in a cluster). While this is not an unreasonable requirement, it may not apply to the small-world topologies [6]. We plan to explore this aspect in the future via experimental evaluations.

CHAPTER 5
FLASK: EXPERIMENTAL EVALUATION

This chapter studies the performance of the main FLASK components in trace-driven simulations. Section 5.1 evaluates the performance of the clustering mechanism. Section 5.2 studies the costs and benefits of disseminating information within interest-based clusters. The requirements that constitute the motivation for designing FLASK (presented in Section 4.1.1) are revisited in Section 5.3. Section 5.4 summarizes the main results of this chapter.

All results presented in this chapter are based on specific definitions of the underlying data-sharing graphs. As a reminder, two parameters define a data-sharing graph: the time interval τ and the minimum number of common requests μ that defines an edge between two users in the graph.

5.1 Decentralized, Local Information-based Clustering

The clustering component in FLASK has a strong influence on the overall performance of the mechanism: too large clusters may lead to unacceptable overhead on their members, while too small clusters may not reduce the search space efficiently. As shown in the previous chapter, the clustering (graph partitioning) problem is difficult, even more so under our assumption that only local information is known. The metrics relevant for clustering are average cluster size and cluster size distribution. This section presents these two metrics for our three sets of traces.

We have proposed a clustering method (Section 4.4) that uses decentralized, local information-based *triad labeling* whereby an edge that connects two nodes that have a common neighbor is labeled as *small* and all other edges are *long*. We evaluate the performance of this method in the following.

5.1.1 Triad Labeling

Triad labeling leads to highly skewed cluster sizes in all traces. Typically, a large cluster that comprises 60-80% of all nodes is created, along with a fair number of much smaller clusters. Since this pattern appears in all cases, we present only the average cluster sizes: Figures 5.1, 5.3, and 5.5 show the average cluster size at different times for the three systems. Figures 5.2, 5.4, and 5.6 present the average cluster size for each of the data-sharing graph definitions analyzed.

The difference in average cluster size from week to week in D0 is due to variation in user activity: for example, the peak in week 16 in Figure 5.1 corresponds to the peak in day 108 in Figure 3.3.

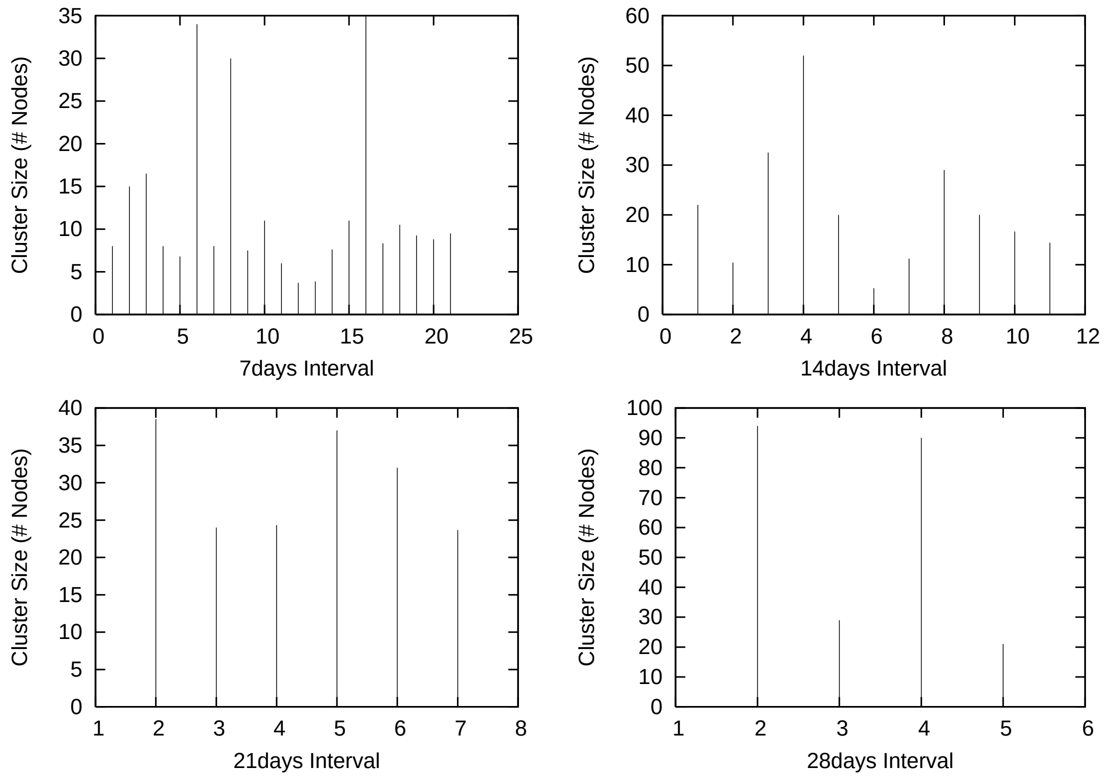


Figure 5.1: Average cluster size over time for D0 traces for $\mu = 100$, τ =7, 14, 21 and 28 days.

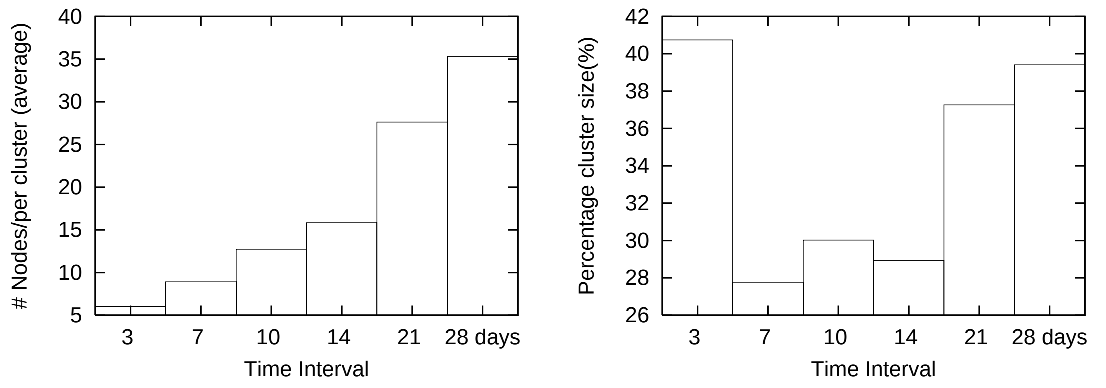


Figure 5.2: Average cluster size resulting from triad labeling in D0.

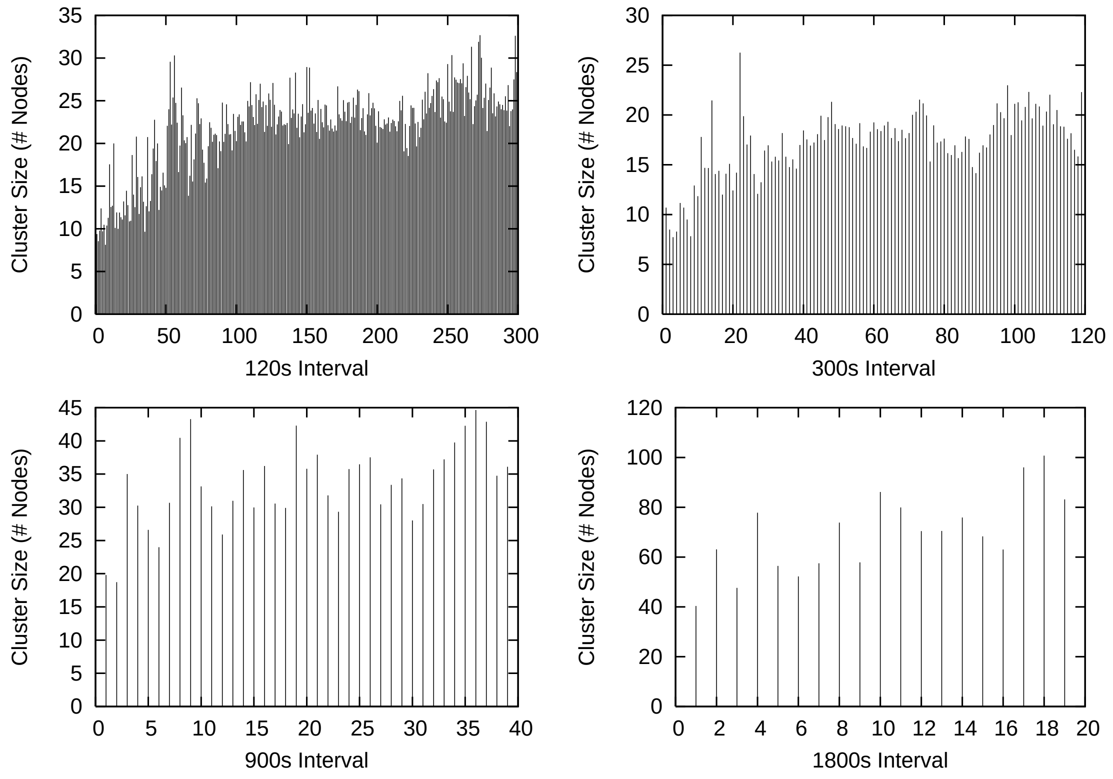


Figure 5.3: Average cluster size over time for Web traces for 2, 5, 15 and 30-minute intervals (1, 5, 10 and respectively 10 shared requests).

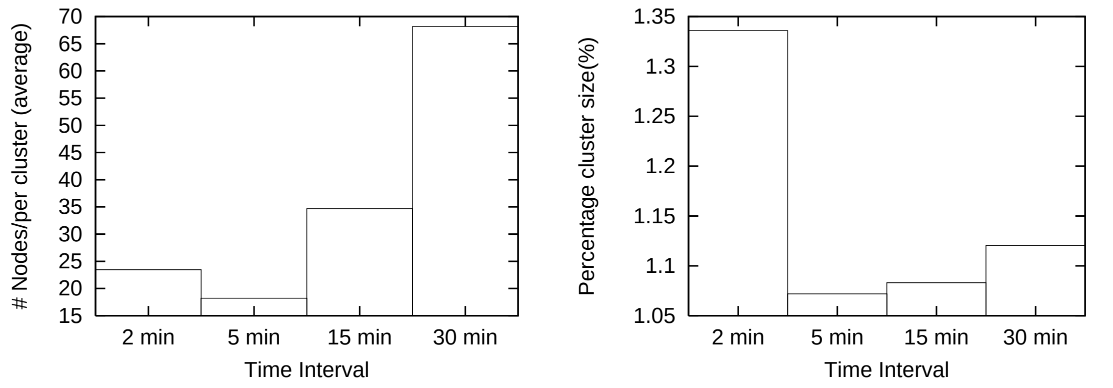


Figure 5.4: Average cluster size resulting from triad labeling in Web.

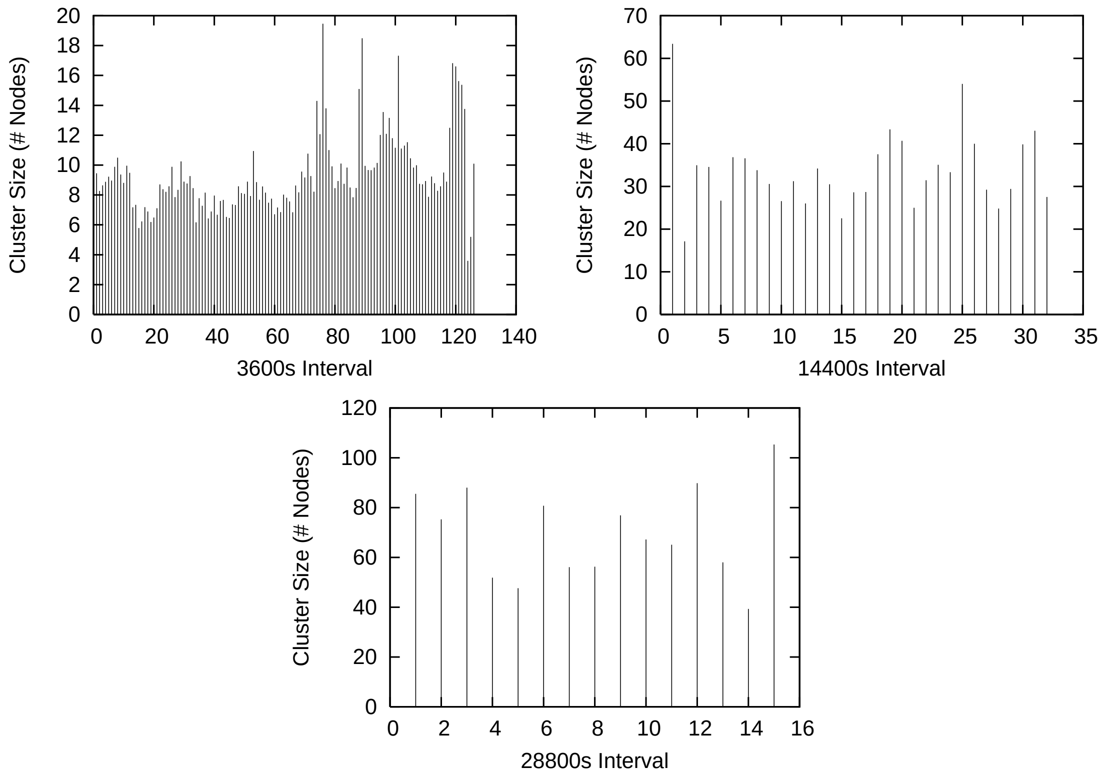


Figure 5.5: Average cluster size over time for Kazaa traces for 1, 4, and 8-hour intervals, 1 shared request.

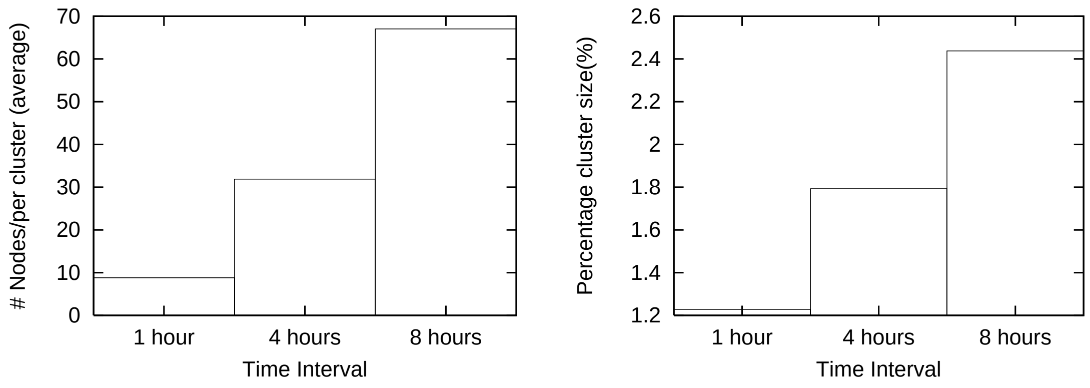


Figure 5.6: Average cluster size resulting from triad labeling in Kazaa.

Note that average cluster size is in the order of tens for all traces and data-sharing graph definitions: for D0 it varies from 10 to 35 nodes per cluster, for the Web it is between 20 and 65 and for Kazaa between 10 and 70. However, given the different sizes of the data-sharing graphs, the number of clusters resulted is highly different: under 10 in D0 (where the data-sharing graphs have under 200 nodes); up to 200 in the Web (where data-sharing graphs are in the order of tens of thousands of nodes); and up to 150 in Kazaa (for data-sharing graphs of thousands of nodes).

5.1.2 Discussion

The triad labeling method requires only local information. However, it creates unbalanced-sized clusters, since it identifies a giant cluster and many small ones. To understand if this behavior is dictated by the clustering approach or is a property of the graph, we ran a global-knowledge clustering algorithm [114, 115]. The same results were obtained: a large cluster often comprising more than 60% of all nodes and many small clusters. Moreover, the clusters obtained with the two methods overlap significantly.

While this test does not aim to be a satisfactory proof of the accuracy of the triad labeling, it shows that large clusters may be wired into the graph topology. Thus, additional steps are required to limit the cluster size.

One way to do it is to allow nodes to refuse to label edges as small when the number of nodes in cluster exceeds some limit. However, this approach assumes that the current cluster size can be approximated in a decentralized fashion, which might prove difficult.

Alternatively, more sophisticated labeling strategies can be employed, for example to include criteria unrelated to user interests, such as network nearness (in network latency) or node lifespan. Low latency between peers can significantly improve the information propagation time. Selecting peers with longer lifespan seems to considerably improve performance [24] by assuring longer lived connections between peers.

5.2 Information Dissemination: Costs and Benefits

The main metric of success in disseminating information is hit rate, which is the percentage of requests answered locally from the information previously disseminated. There are multiple benefits associated with a high hit rate: first, response latency for hits is that of a local lookup, therefore low. Second, no messages are inserted into the network. However, not all hits are due to information dissemination: for example, requests for files that are stored locally.

Costs include communication and storage: how much information is circulated within clusters, how much information a node needs to send, and how much storage per node is needed to maintain the information disseminated.

The performance and costs of information dissemination is discussed in the following sections. Section 5.2.1 presents the experimental setup. Two types of information can be disseminated: experience and storage summaries. Section 5.2.2 analyzes in detail the advantages and the costs of experience sharing within clusters while Section 5.2.3 focuses on storage summary dissemination.

5.2.1 Experimental Setup

For each set of traces, we consider multiple instances of data-sharing graphs. As a reminder, a data-sharing graph is defined by two parameters: the time interval τ and the minimum number of shared requests between two nodes μ. For each definition, we build the data-sharing graph for the interval τ and "freeze" it at the end of the interval. We then apply the triad labeling technique to identify clusters of interest and disseminate information within clusters. In all cases, the information disseminated is about location of files. We analyzed two instances: first, the location of the files *stored* on the local peer is disseminated. We call this information *storage summaries*. Second, the location of files discovered in the previous interval is disseminated. We call this case *experience sharing*. We then measure how many queries sent during the following interval can be answered from the information just disseminated.

We do not rely on caching information disseminated more than one interval τ ago for multiple reasons. First, the overlay changes at each interval and so do clusters. Second, file location information is likely to be volatile (since files and nodes frequently and unpredictably join and leave the system), which may lead to high percentage of inaccurate information.

Note that in a real implementation, the overlay varies and adjusts to user behavior smoothly over time. This may lead to better hit rates, since the overlay and clusters adapt faster to changing in user interests.

The dissemination of storage summaries poses a particular problem. In a P2P scenario, files are stored on user nodes. However, information on this distributed storage does not appear in traces. In the D0 experiment, storage is centralized. In the Web and Kazaa traces, while we have information on the servers where files are stored, there is no correlation between users and these storage servers for two reasons. First, the set of servers and the set of users are overlapping but are not identical, as users are Boeing employees (for Web) or ISP clients (for Kazaa), while servers may be anywhere on the Internet. Second, even the existing correlation between users and servers from the same ISP/institution was lost in trace anonymization.

To overcome this problem, we generated a set of mappings of files onto user storage. In all cases, we considered only one copy of each file stored in the system. While this assumption may be unrealistic, it simplifies the setup and eliminates additional sources of bias. Most importantly, it gives a lower bound on performance metrics. Therefore, all results presented in the following must be considered lower bounds.

The synthetic mappings are:

1. *First requester mapping* (labeled `1st` on plots) associates a file with the user who first requested it. This mapping favors users active at the beginning of traces.

2. *Any requester mapping* (labeled `"any"` on plots) associates a file with a random user who requested that file.

3. *Random mapping* (labeled `"random"` on plots) associates a file with a random user, regardless of that user's interest in the file.

Not all users store files in these mappings: Table 5.1 shows the number and percentage of users who store and therefore contribute files in each community.

Table 5.1: Number (and percentage) of users who contribute files.

Traces	Nodes	Files	Random Providers (%)	1st Requester Providers (%)	Any Requester Providers (%)
D0	317	193684	153(48.26)	219 (69.08)	241 (76.02)
Web	60826	4794439	53646 (88.19)	46183 (75.92)	46434 (76.33)
Kazaa	14404	116509	10397 (72.18)	11638 (90.79)	11648 (80.86)

Table 5.2: Requests for files stored on the requester node. For Web, data are collected based on 100 samples of 5 minutes taken at equal intervals during the 10-hour traces.

System	Random	1st Requester	Any Requester
D0	1.16%	22.16%	23.15%
Web	0.08%	37.86%	36.53%
Kazaa	0.09%	34.49%	36.72%

The mappings chosen are just some examples of possible scenarios. `1st` and `any` requester mappings are realistic scenarios under the assumption that nodes store files of their own interest. However, in these cases, a significant number of queries (22%–38%) are for files stored on the requester node. Table 5.2 shows the percentage of requests in the system whose answers are on the local (requester's) disk: this information is necessary for correctly evaluating the benefits of disseminating information in clusters and will be relevant in Section 5.2.3.

5.2.2 Experience Sharing

In this scenario, peers gossip the experience accumulated during the previous interval τ. This experience consists of the set of file locations learned (as responses to

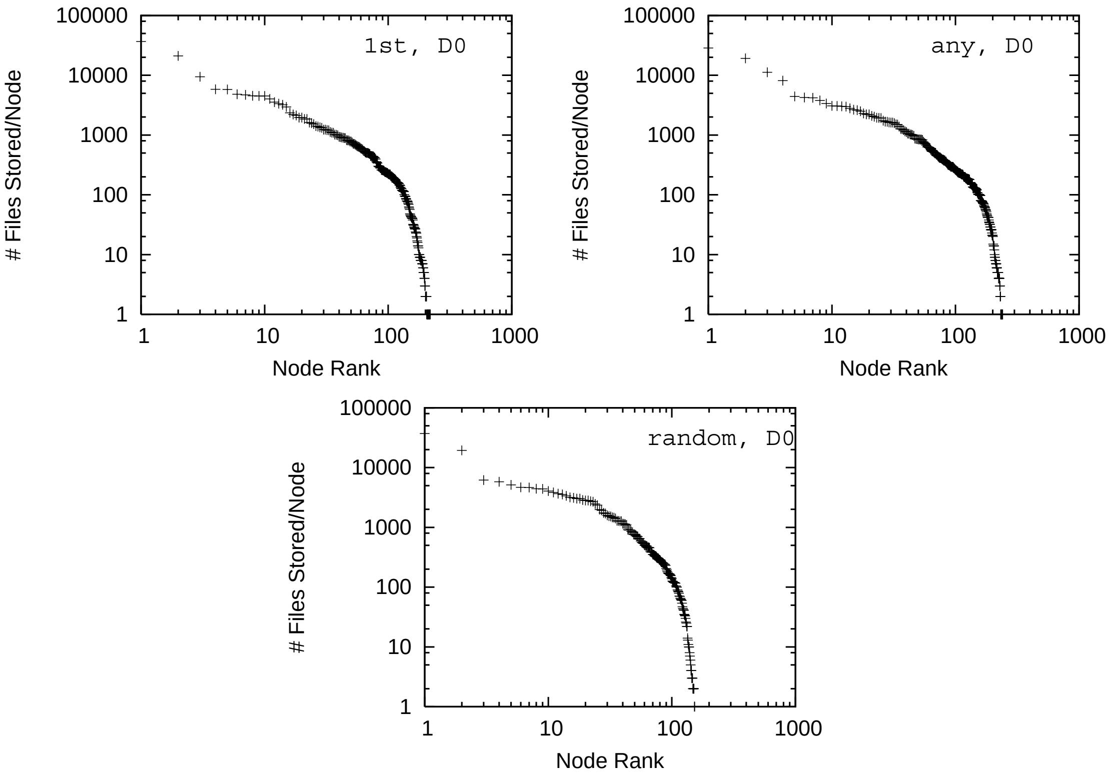


Figure 5.7: Number of files stored per user node in D0.

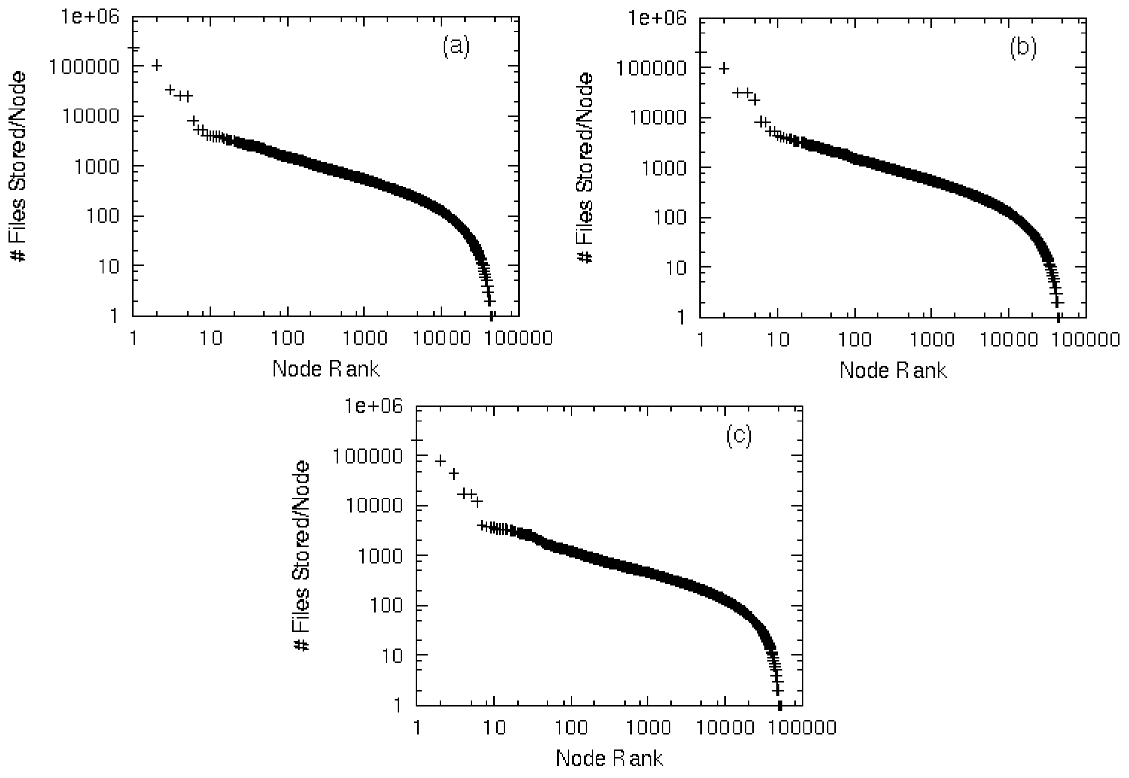


Figure 5.8: Number of files stored per Web user node. (a) First requester mapping; (b) Any requester mapping; (c) Random mapping.

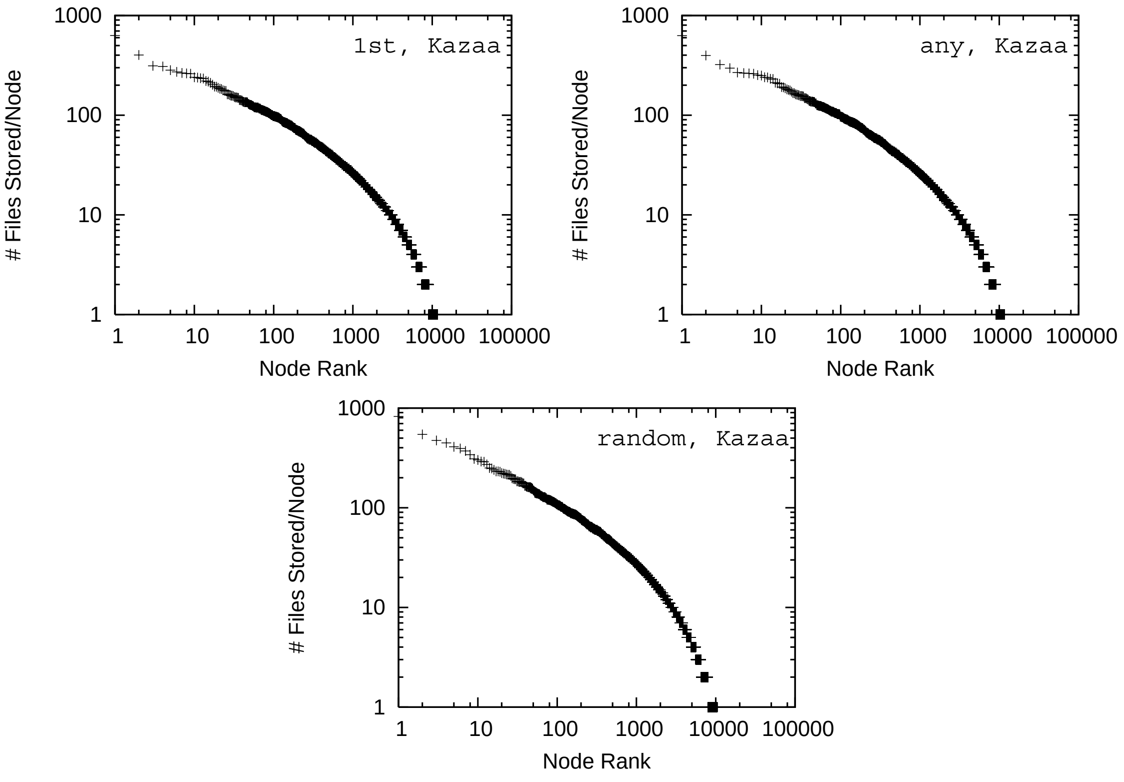


Figure 5.9: Number of files stored per Kazaa user node.

queries) during this interval. Under the assumption that files can be removed from or inserted frequently into one's storage, old experience quickly becomes outdated: that is, the information received two intervals ago is most likely outdated and will not be propagated or even maintained in the local database.

This means that no influence from caching exists except for when the same user asks same requests in consecutive intervals: their later requests can therefore be answered from the local cache and should not be attributed to information dissemination. However, the percentage of requests of this sort is low: it varies up to 15%, with averages between 5% and 10% for different intervals in D0. Surprisingly, the Kazaa users show the same behavior, with similar average cache rates (around 7%): while it is conceivable that scientists repeat requests for the same data to run their computations, it is less intuitive that a Kazaa user would repeat requests for same music at intervals of one hour. This behavior may be explained in by unsatisfactory answers: files were not found, were not downloaded properly, or were corrupted.

The results in this section are based on the assumption that all requests are answered within the interval in which they are asked. For an approximation of the hit rate under the (realistic) scenario that not all requests are answered, one needs to consider the percentage and the popularity of unanswered queries to obtain an estimation from the results presented below. However, the results we present here are also relevant for evaluating the potential of a hybrid scenario: for example, use information dissemination within clusters to reduce the load on a centralized index such as the one used in Napster.

D0

Figure 5.10 shows the hit rate per interval for different values of τ. Observe that the hit rate decreases with the increase in duration (summarized in Figure 5.12). Interestingly, this behavior is not found in the other communities: for Web (Figure 5.15) and Kazaa (Figure 5.18) the hit rate increases (albeit slightly) with longer intervals. A possible explanation for this different behavior is a stronger time locality in scientific communities: scientists repeatedly request the same files as they repeat their experiments on the same data. Once they finish with that region of the problem space, they move to a new set of files. This assumption may be true at the group level but it is not true at the individual level: the percentage of a user's repeated requests at consecutive intervals in general increases slightly with the duration of the interval (6.35%, 7.5%, 7.6%, 8.7%, 9.1%, and 7.75% for $\mu = 100$ and τ =3, 7, 10, 14, 21, and 28 days).

One effect of the imperfect clustering algorithm based on edge labeling is the creation of the large cluster. Figure 5.11 evaluates the impact of the largest cluster on hit rate, by comparing the hit rate of the overall system with that when the benefits of the largest cluster are ignored. The plots show the difference between the two hit rates: a negative value therefore shows that the largest cluster decreases the overall

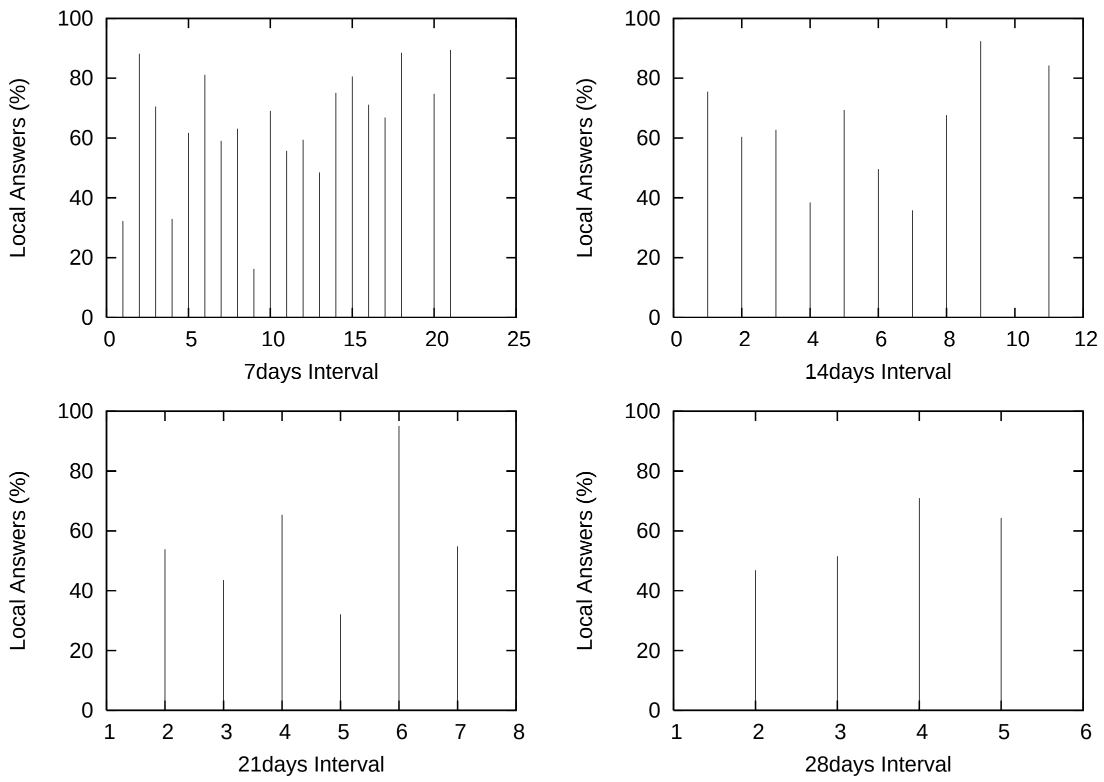


Figure 5.10: Average percentage of requests solved locally (based on information within cluster): 7-, 14-, 21- and 28-day intervals, 100 shared request, triad labeling.

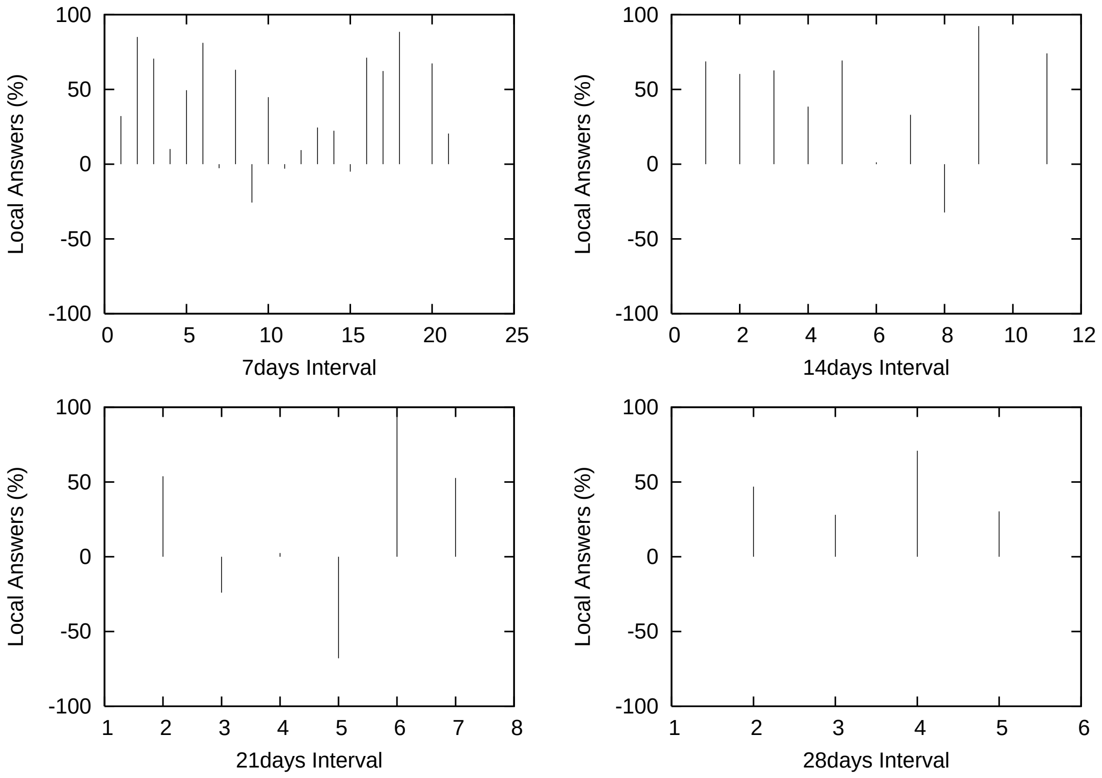


Figure 5.11: The effect of the largest cluster (triad labeling, 7-, 14-, 21- and 28-day intervals, 100 shared request) on average hit rate: the difference between the average hit rate and the average hit rate for all except the largest cluster.

hit rate.

Figure 5.12 summarizes the hit rate results by presenting the average for different τ intervals. It also highlights the effect of the largest cluster on the overall hit rate. It is remarkable that in D0 this influence is significant for longer intervals.

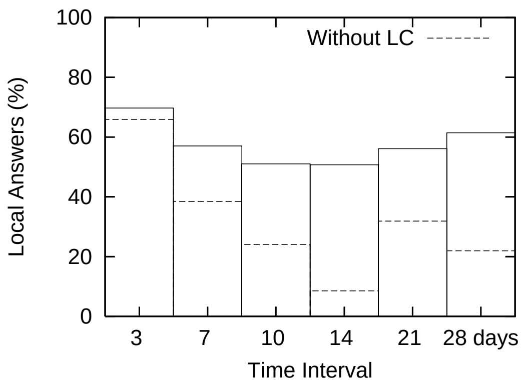


Figure 5.12: Average percentage of local answers with triad labeling on D0 traces.

The benefits of disseminating information in D0 are significant: more than 50% of queries do not impose any costs on the network and their answer latency is that of a local lookup. The price for these advantages is paid in dissemination costs and storage of the information disseminated. Table 5.3 presents the raw costs of information dissemination: storage space required per node is a function of the number of files disseminated within a cluster; communication costs per node depends on the number of files disseminated within the cluster and the size of the cluster.

Table 5.3: Experience dissemination costs in D0.

Data-sharing graph definition	Average # filenames stored per node
3 days, 100F	8787
7 days, 100F	18182
10 days, 100F	24912
14 days, 100F	32239
21 days, 100F	44911
28 days, 100F	56924

The longest filename in the D0 traces has 183 characters. Assuming a pair filename–location takes 300 bytes on average, the storage cost to maintain all file information disseminated in cluster is obtained by multiplying the number of files with the storage cost per file. Over a three-day interval this cost is 2.6MB per node. For the 28-day interval, this cost grows to 15MB per node.

Given the gossip-based information dissemination mechanisms used, the communication costs are estimated by multiplying storage costs with the natural logarithm from the number of nodes in cluster. (This is because each message needs to be sent to approximately $ln(N)$ peers in cluster to ensure the message will reach all N members.)

Consequently, communication costs are not high, either: 5.2MB of data transmitted per node in a 3-day interval, growing to 60MB for the 28-day interval case. In D0, there are fewer than 200 nodes in the network during any of the intervals considered: this results in at most 12GB of data exchanged over a month. To put this in context, compare with the Gnutella traffic as measured in late 2000 [98]: 6GB per node or 1.2TB of data exchanged by 200 Gnutella nodes over a month. Therefore, the information dissemination component of FLASK—arguably the most resource consuming—requires 2 orders of magnitude less communication than the (full) Gnutella mechanism in late 2000.

Moreover, as proposed in the previous chapter, the information disseminated can be compressed using Bloom filters. In this case, fewer resources are consumed, but approximate matches or keyword searches cannot be supported. Assuming that the Bloom filters are tuned to represent up to 1000 files with false positive rate under 0.1%, each node's storage summary requires 2KB (2 bytes per entry and 10 hash functions). With experience sharing, in the worst case scenario each node has partial information from every node in the network (not only in its local cluster). In the case of D0, this means each node eventually stores 317 Bloom filters of 2KB each, which is under 1MB of storage space. Interestingly, this storage cost does not grow with the length of the interval τ: a larger number of files known locally will fill in the (previously sparse) fixed-size summary vectors.

Web

The same experiment on the web traces leads to different results. First, the hit rate seems to increase over time (Figure 5.13). This is the effect of the increased activity—number of queries per time unit—shown in Figure 3.7.

Second, the hit rate seems to increase (slightly) with the increase of interval duration, while user's cached repeated requests remain constant (under 2.5%).

Third, the effect of the unbalanced-sized clusters is often negative: Figure 5.14 shows that the largest cluster leads to smaller hit rates in some intervals. However, its average effect is still positive (between 2% and 13%, depending on the data-sharing graph definition).

Figure 5.15 summarizes these results: the hit rate is around 40% and the poor labeling technique does not (significantly) increase the overall hit rate.

Dissemination costs for the Web traces are higher than in the D0 case (Table 5.4). Nodes will need to store 48MB of data for half-hour intervals and will send 240MB of uncompressed data each. The communication costs are unrealistically high: for user

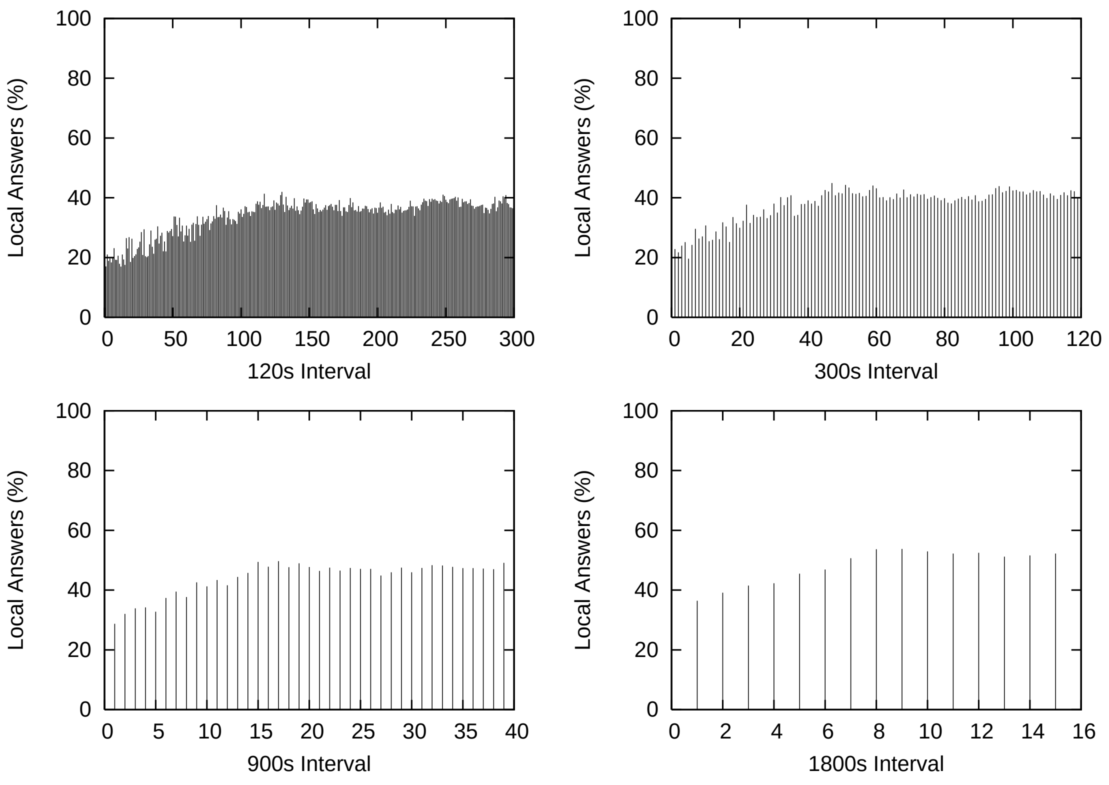


Figure 5.13: Average percentage of requests solved locally (based on information within cluster): 2-, 5-, 15-minute, and half-hour intervals, with $\mu = 1$, 5, 10, and 10 shared requests.

Table 5.4: Experience dissemination costs in the Web.

Data-sharing graph definition	Average # filenames stored per node
120s, 1F	21424
300s, 5F	41442
900s, 10F	111137
1800s, 10F	192079

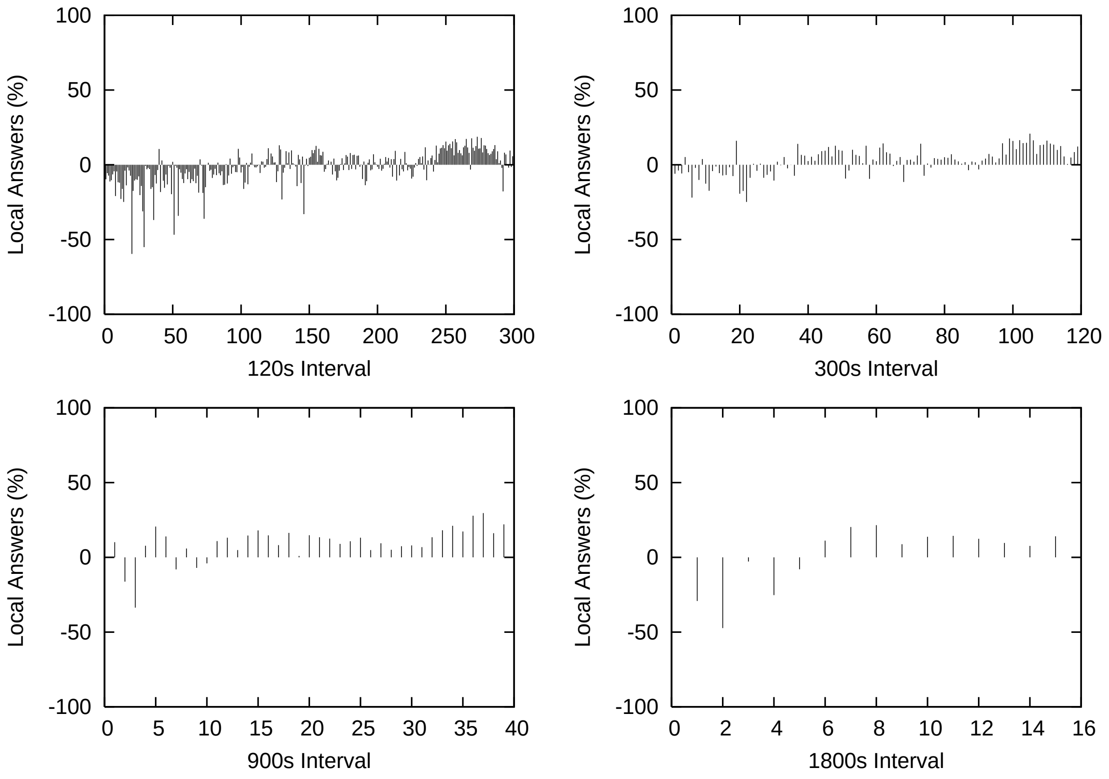


Figure 5.14: The effect of the largest cluster (triad labeling, 2-, 5-, 15-minute and half-hour intervals, 1, 5, 10, 10 shared requests) on average hit rate: the difference between the average hit rate and the average hit rate for all except the largest cluster.

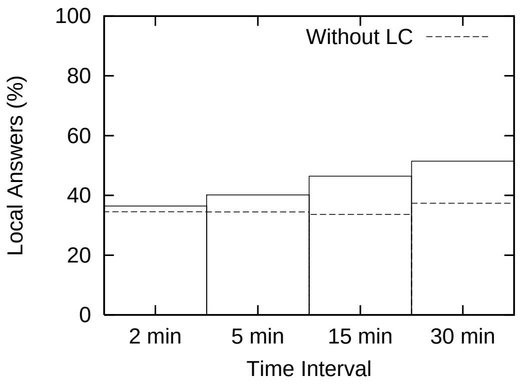


Figure 5.15: Average percentage of local answers with triad labeling on Web traces.

behavior typical of web browsing, compressing information is mandatory.

Kazaa

Yet another set of results are obtained using the Kazaa traces: Figure 5.17 shows constantly high hit rate (around 70%) for all durations, in spite of the decrease in the user's cached repeated requests (from almost 8% for one-hour to 5.5% for 8-hour intervals). The wavy pattern in Figure 5.16 is likely due to increases in activity, evident in daily patterns (Figure 3.9). On the other hand, the influence of the largest cluster increases with duration: from 13% for 1-hour to 44% for 8-hour intervals (Figure 5.17). These results are summarized in Figure 5.18.

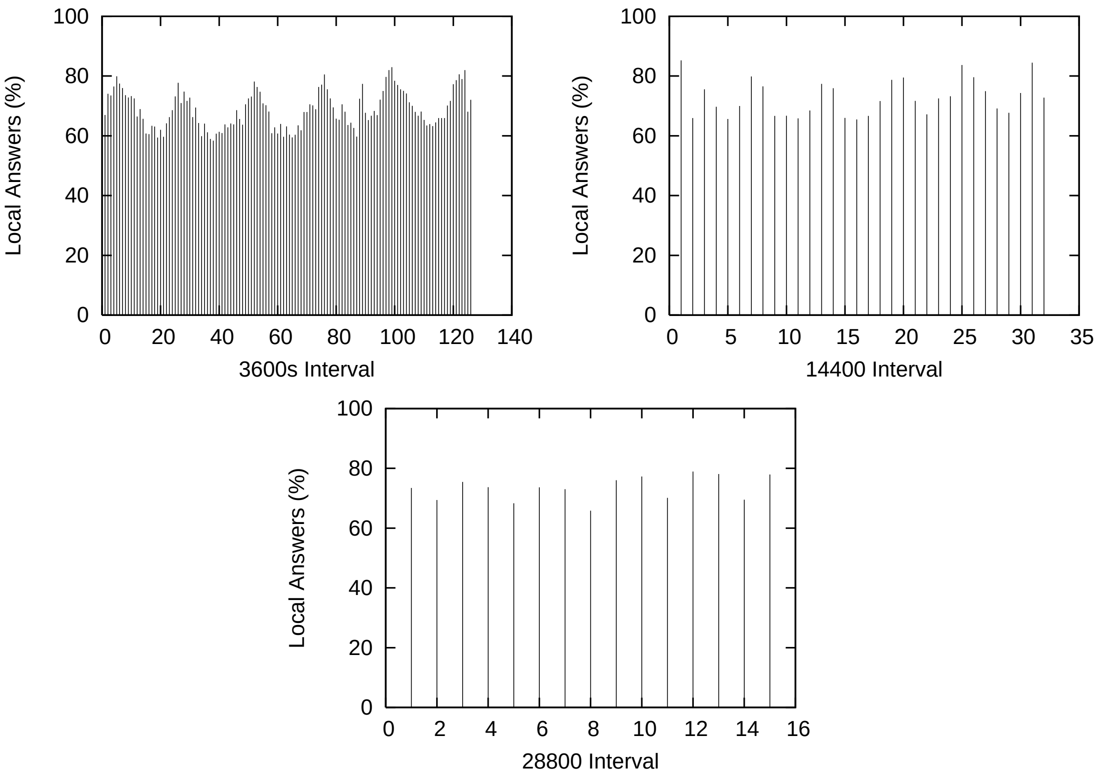


Figure 5.16: Average percentage of requests solved based on information disseminated within Kazaa triad-labeling clusters: 1-, 4-, and 8-hour intervals, 1 shared request.

Dissemination costs for Kazaa are smaller than in the other two cases (Table 5.5): 3MB of storage is necessary for storing the data disseminated over 8-hour intervals and 15MB of data are to be sent by each node over an 8-hour interval. For about 3000 nodes, about 45GB of data are transferred over 8-hour intervals, which is 4 times less data than transferred by the same number of Gnutella users at the end of year 2000.

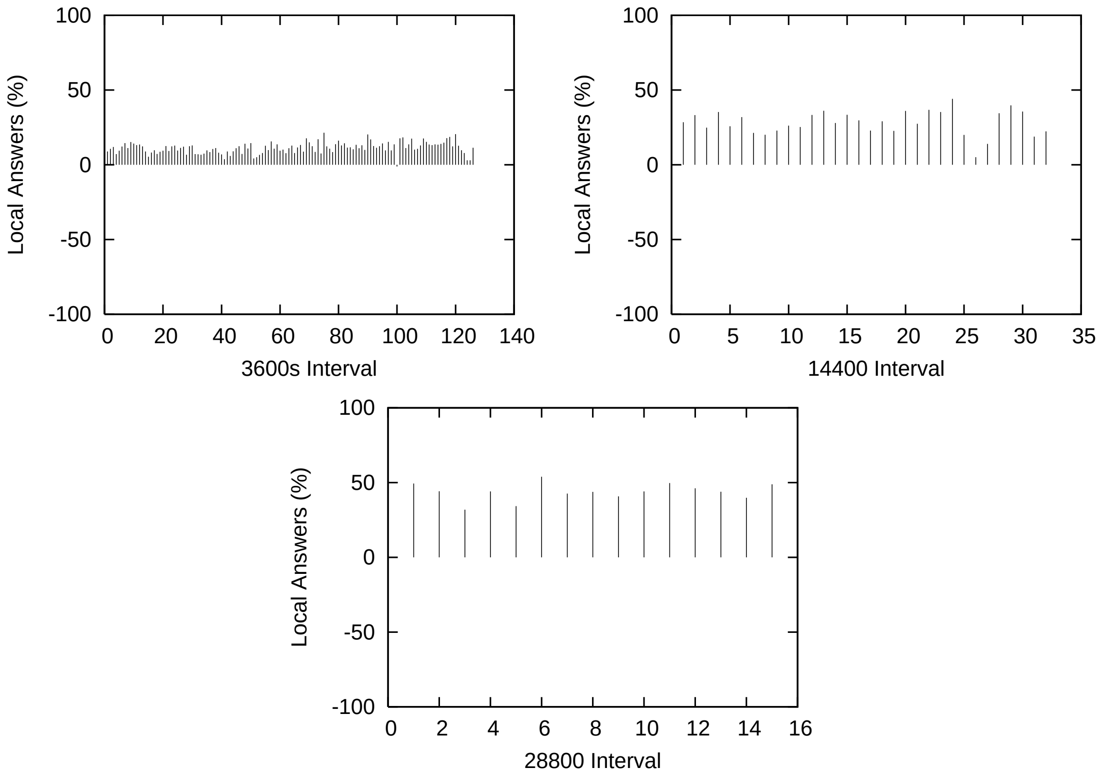


Figure 5.17: The effect of the largest cluster (triad labeling, 1-, 4- and 8-hour intervals, 1 shared request) on average hit rate in Kazaa: the difference between the average hit rate and the average hit rate for all except the largest cluster.

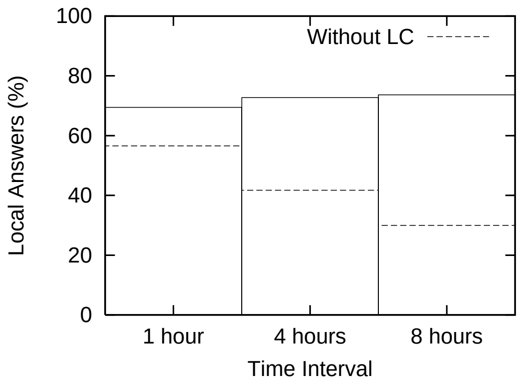


Figure 5.18: Average percentage of local answers with triad labeling on Kazaa traces.

Table 5.5: Experience dissemination costs in Kazaa.

Data-sharing graph definition	Average # filenames stored per node
1 hour, 1 file	1345
4 hours, 1 file	5303
8 hours, 1 file	10178

Despite the quantitative differences, the benefits of disseminating experience within clusters formed with triad labeling are significant for all three communities, ranging from 40 to 70% of the queries being answered from local storage. Also, eliminating the unreasonably good effects of the imperfect clustering algorithm, the hit rate is between 30% and 65%. Shorter time intervals for the data-sharing graph are preferable to keep costs down. Interestingly, the average hit rate over all except the largest cluster is higher for shorter intervals.

However, dissemination costs are unreasonably high for the web traces. One can rightly argue that web requests are considerably more frequent than requests for files in the context we consider because of the inherently lower file granularity: web files are much smaller than music or data files, they can therefore be "consumed" faster, and therefore requested more frequently. Nevertheless, if requests for files are so frequent, compressing data is necessary.

5.2.3 Disseminating Storage Information

In this section we look at the benefits of disseminating storage information to peers in the same cluster: each peer publishes what files it stores.

The same metric of success—hit rate—is relevant in this case. However, in this case, the hit rate has multiple components. First, a significant percentage of queries refer to files stored on the requestor's node, as shown in Table 5.2. This component is not the merit of the FLASK mechanism, but rather an artifact of the setup. Second, as already explained in the previous section, a giant cluster leads to a larger hit rate, as it includes a large percentage of all peers and, consequently, a large percentage of the stored files. Given the unrealistic costs that a giant cluster implies, this behavior must be corrected by smarter clustering mechanisms. Until then, though, Figures 5.21, 5.25, and 5.28 delimit the effect of the giant cluster on the benefits of information dissemination.

We summarize here some notable results presented in the following sections: first, as in the previous experiment, information dissemination performs differently in the three systems. For example, in D0 and Kazaa, information dissemination leads to 20% to 55% of requests being found locally (that is, no latency involved in locating the files). The Web traces are different, as only 5 to 13% of requests are found in this way.

Second, the results are significantly poorer than in the previous experiment: after eliminating the effects of the unrealistically "helpful" factors such as the largest cluster and local storage, the remaining hit rate is not larger than 15%.

D0

Figure 5.19 presents the hit rates over time for the three mappings and different interval lengths. Note that the `1st` and `any` mappings yield larger hit rates than the `random` mapping. For D0 (see Table 5.2) 22% and, respectively, 23% of requests are for files stored locally, that can be retrieved easily via local mechanisms. The hit ratio due solely to information dissemination is between 27% and 50% for `1st` mapping, between 21% and 52% for `any` mapping, and between 25% to 57% for `random` mapping: roughly between 20% and 50% of the queries can be resolved locally from the information disseminated within clusters. However, what differentiates these results is the influence of the largest cluster (Figure 5.20): the average hit rate over all clusters except the largest is between 0 and 27% for `1st` mapping, between 10% and 65% for `any` mapping, and close to 0% for `random` mapping.

Compared to the benefits of disseminating past experience (Section 5.2.2) summarized in Figure 5.12, we observe the following. First, the hit rate in the storage information dissemination case depends strongly on file location. Second, the hit rates obtained in the two cases (experience sharing and storage information dissemination) are not very different. This does not happen for the other sets of traces, as we shall see in the following.

Storage summaries contain from 0 to 37,000 items per node (Figure 5.7). Following the same reasoning as before, if a filename-location pair is 300 characters (bytes) long, the node with the largest number of stored files will have a 10MB uncompressed summary. A 10MB message is obviously too costly to gossip. On the other hand, it is not necessary to disseminate the full summary repeatedly: gossiping incremental changes or portions of the summary may be a better approach. However, using Bloom filters reduces the communication costs considerably, since a maxium of 37,000 files per node can be represented with a 75KB Bloom filter.

Another cost is the space needed to store the information received from peers in a cluster: Figure 5.22 gives the average number of files disseminated within a cluster (this time, taking into account the largest cluster, which gives a worst case scenario). Storage requirements increase with the increase of the interval τ to up to 160,000 filenames (approximately 48MB) to be stored on each node. While this is not an unreasonable cost in todays' PCs, Bloom filters would reduce these costs to about 2.7MB (considering 75KB per Bloom filter and 35 nodes per cluster on average, as shown in Figure 5.2).

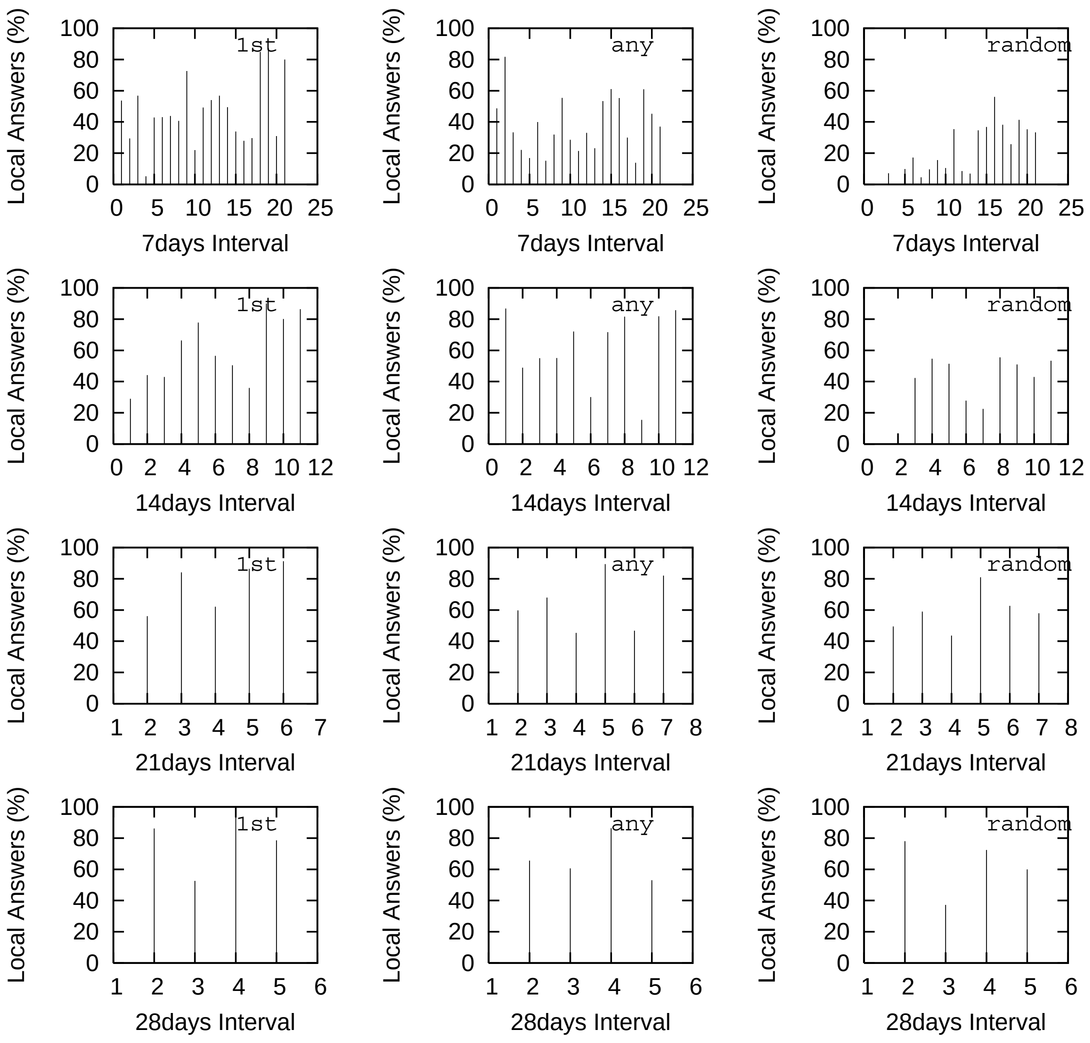


Figure 5.19: Average percentage of requests solved locally (based on information within cluster): 7-, 14-, 21- and 28-day intervals, 100 shared request, triad labeling.

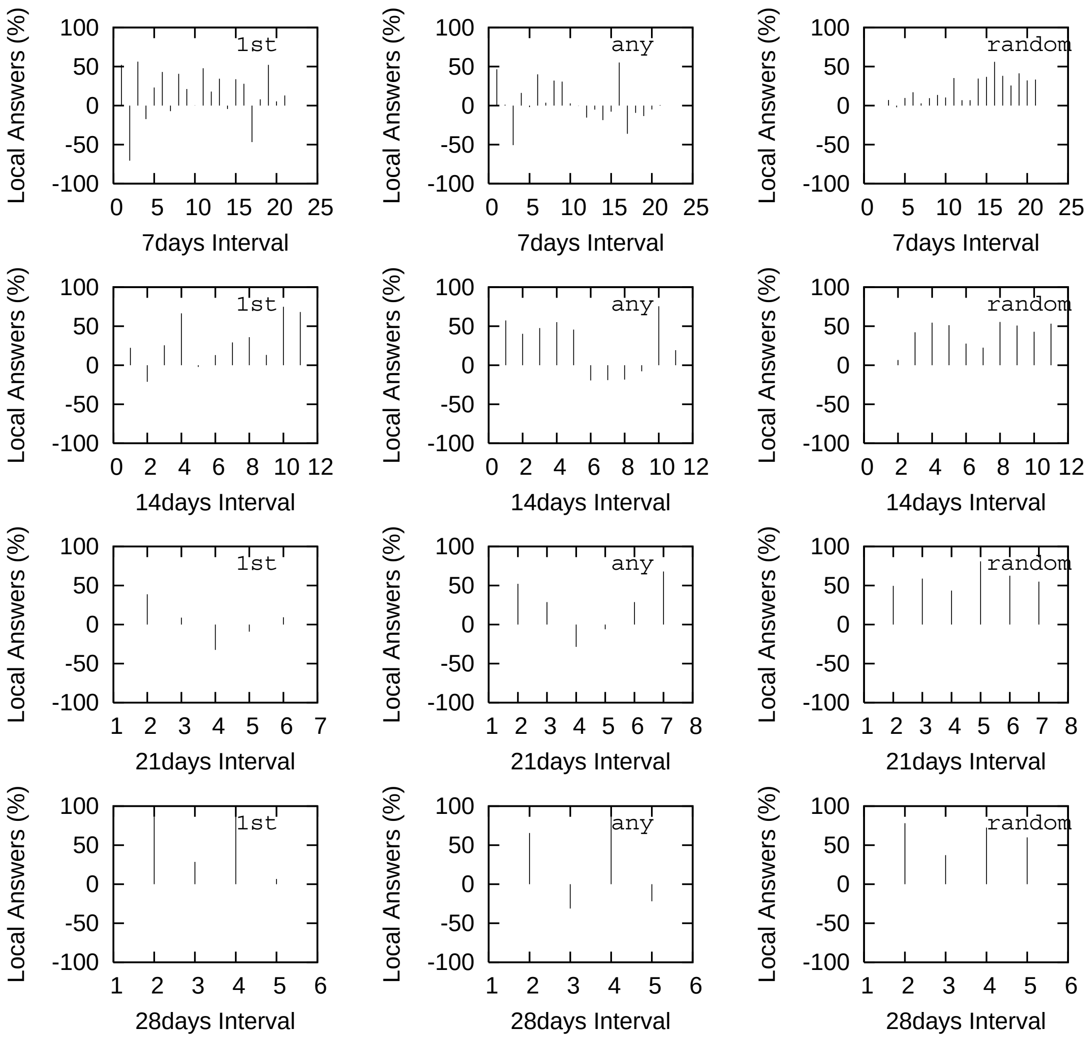


Figure 5.20: The effect of the largest cluster (triad labeling, 7-, 14-, 21- and 28-day intervals, 100 shared request) on average hit rate: the difference between the average hit rate and the average hit rate for all except the largest cluster.

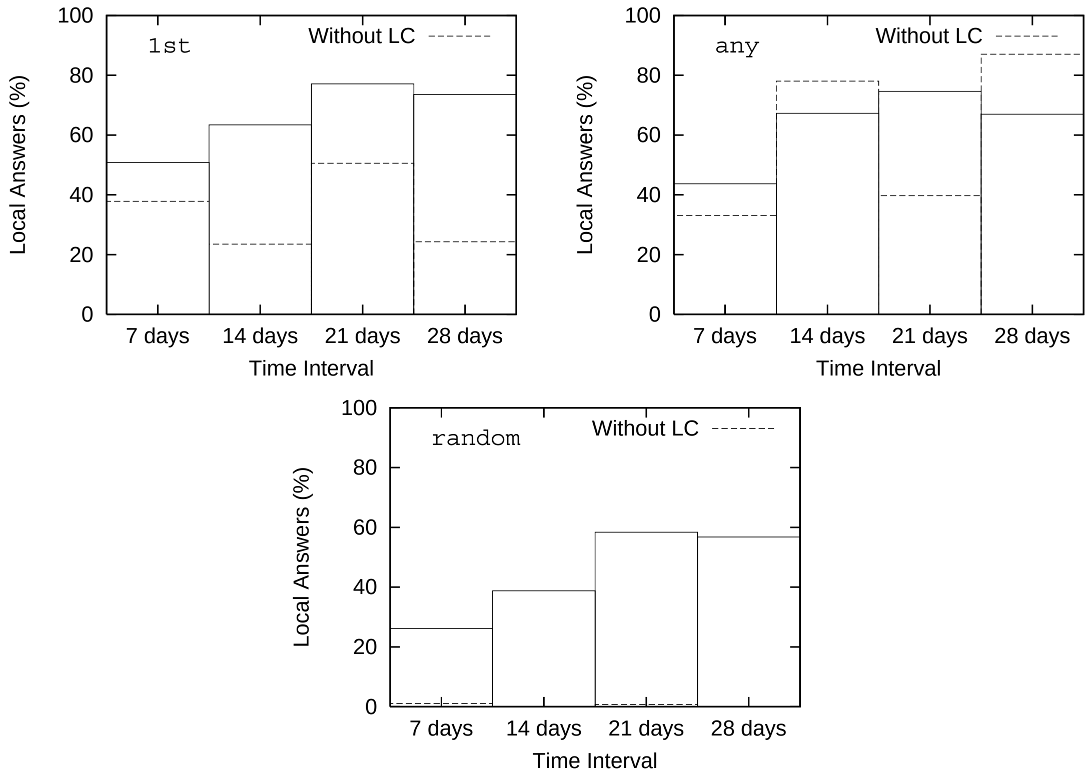


Figure 5.21: Average percentage of local answers with triad labeling on D0 traces.

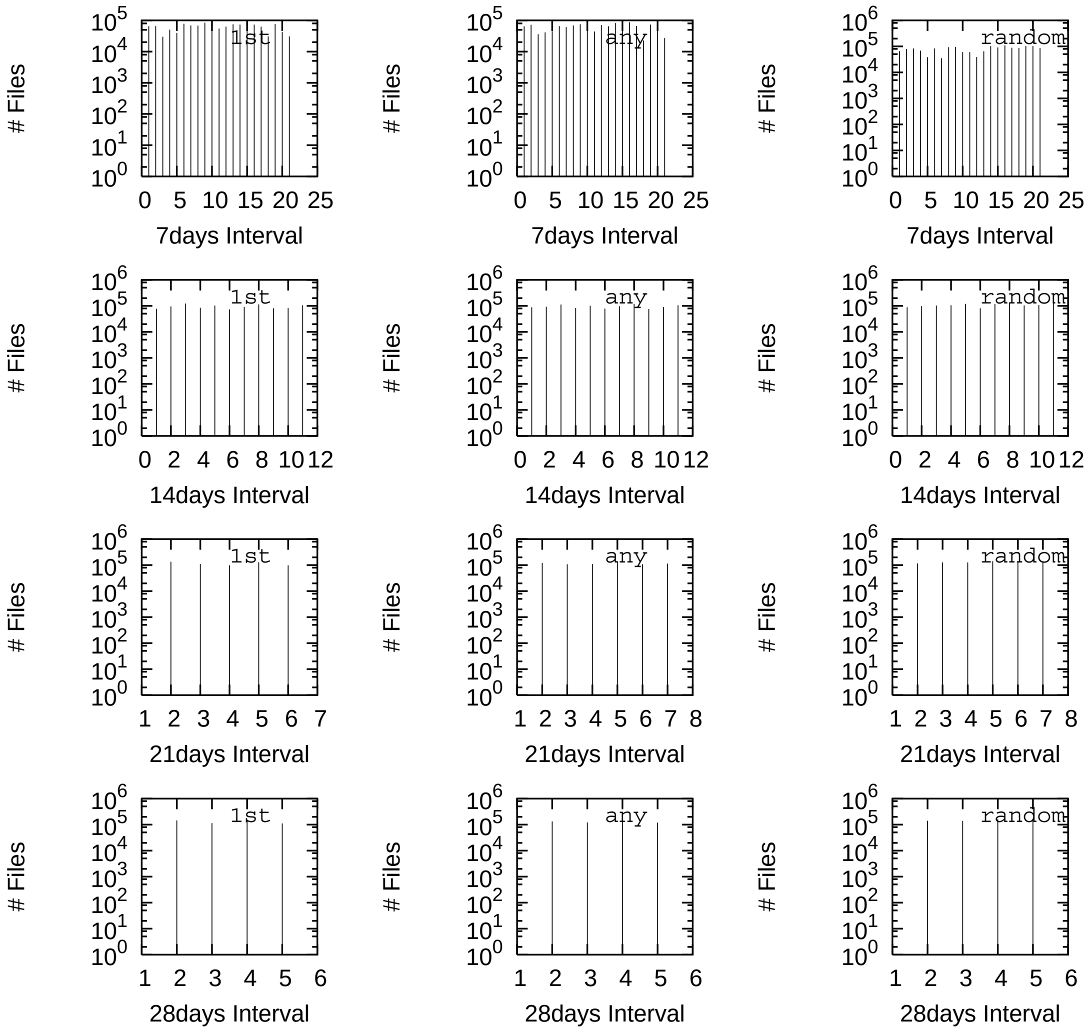


Figure 5.22: Average number of files disseminated per cluster in D0.

Web

In the Web traces the bias introduced by `1st` mapping is more evident than in the D0 case: requests asked at the beginning of the traces are answered from the local storage in considerably higher percentage (Figure 5.23). However, the average hit rate is only slightly larger for the `1st` mapping setup (Figure 5.25).

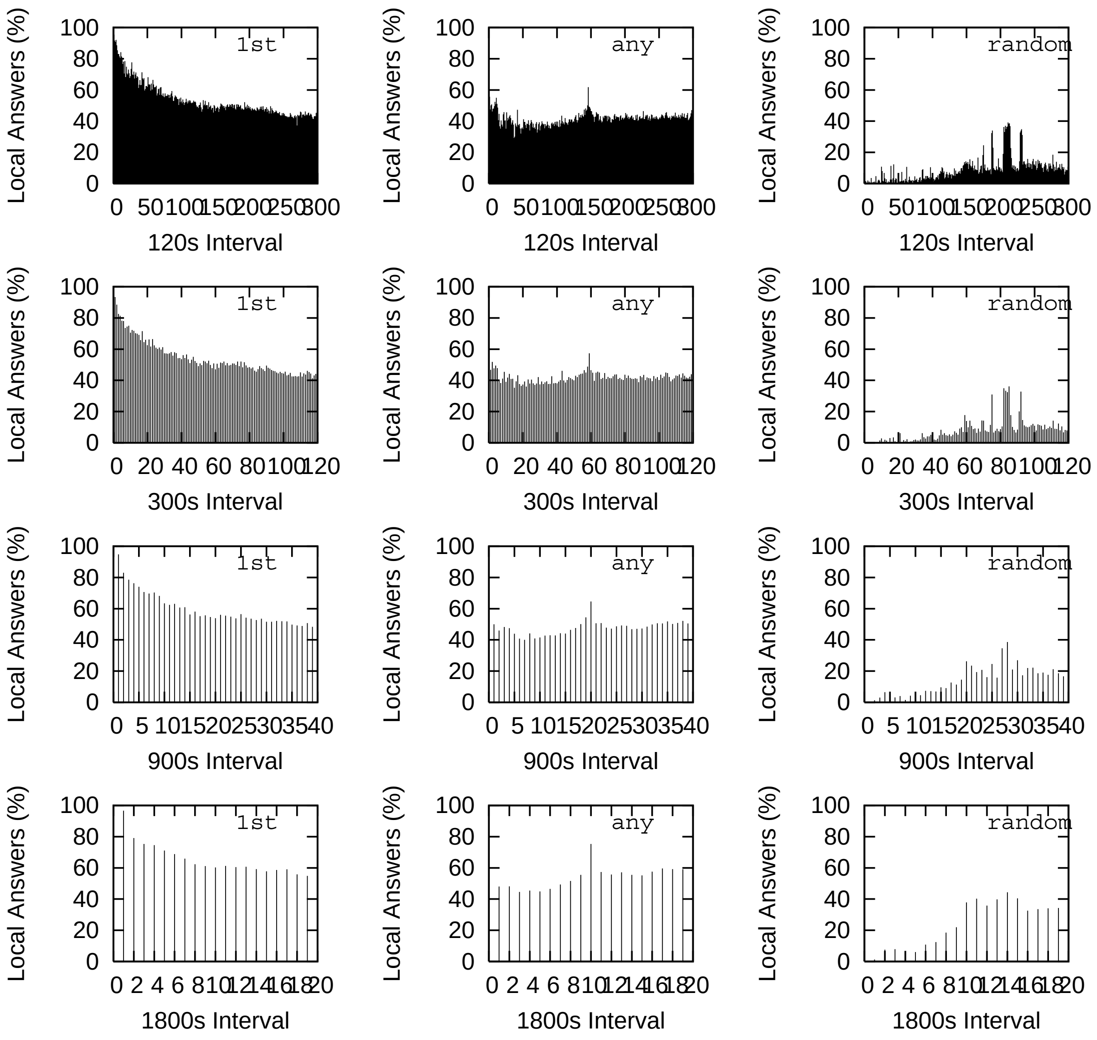


Figure 5.23: Average percentage of requests solved locally (based on information within cluster): 2-, 5-, 15-minute and half-hour intervals, 1, 5, 10, 10 shared requests, triad labeling.

The random mapping scenario performs poorly: the increase in hit rate over time is due to the increase in activity (number of requests per second). This leads to a larger largest cluster, therefore to potentially many more files publicized in this largest cluster. The effect of the largest cluster and its increase over time is shown in Figure 5.24.

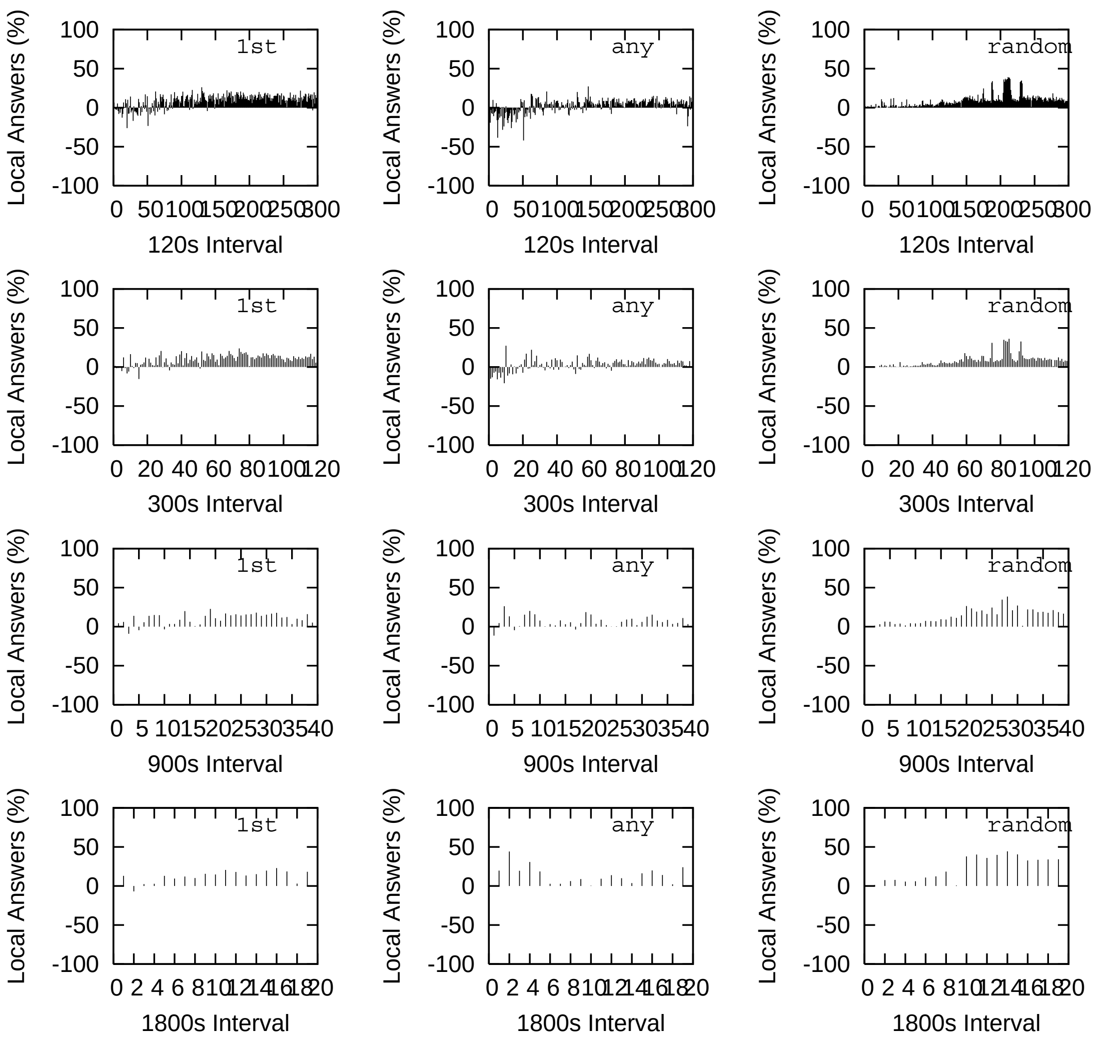


Figure 5.24: The effect of the largest cluster (triad labeling, 2-, 5-, 15-minute and half-hour intervals, 1, 5, 10, 10 shared requests) on average hit rate: the difference between the average hit rate and the average hit rate for all except the largest cluster.

The hit rate due to information dissemination only (therefore ignoring requests for local files) is between 10% and 23% for `1st` mapping, between 6% and 21% for any mapping, and between 10% and 30% for random mapping (Table 5.2 gives 36-37% of local queries for `1st` and any mappings). Of these values, about 5-10% is the influence of the largest cluster (Figure 5.25).

The benefits of disseminating storage summary in interest-based clusters are limited: under 10% of requests are solved due to information dissemination. In the case

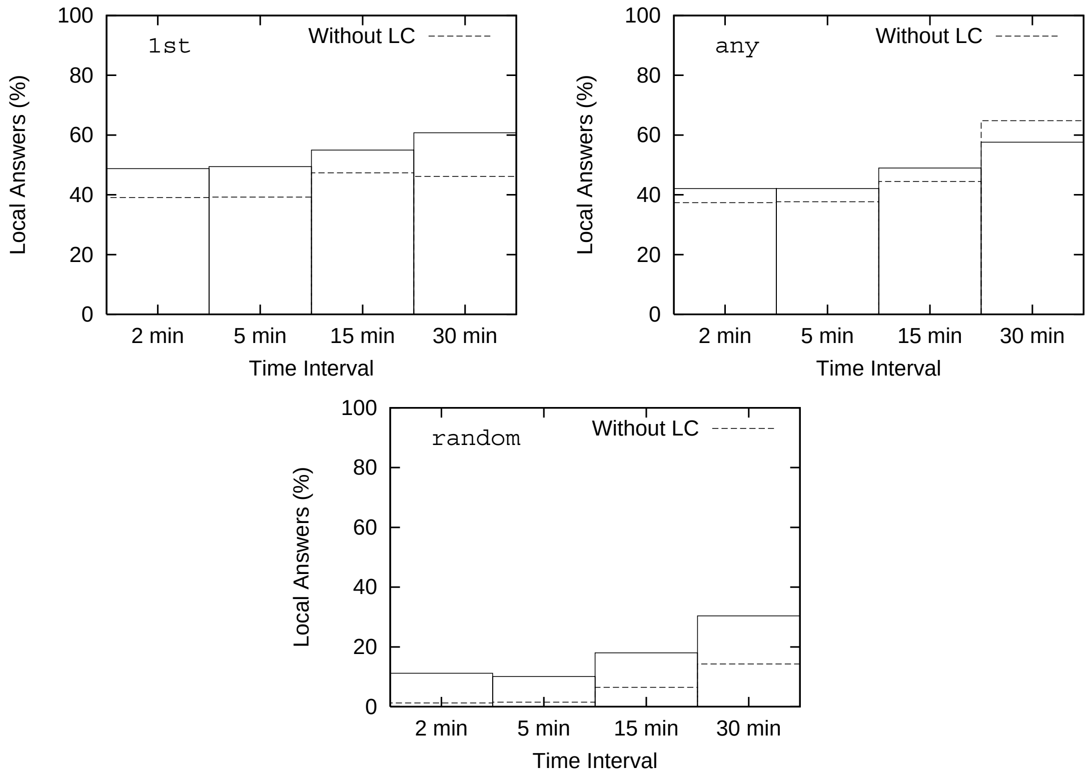


Figure 5.25: Average percentage of local answers with triad labeling on Web traces.

of random mapping the hit rate is mainly the result of the largest cluster. However, the costs are also low: with few exceptions, nodes have under 10,000 files stored locally (Figure 5.8), which limits the storage summary size to 3MB. For a cluster size of about 100 nodes and the average number of files per node of 105, the amount of information disseminated within cluster is under 16MB (with approximately 157KB sent by each node).

Compared to the experience sharing scenario, storage summary dissemination using the Web traces performs poorly: the hit rate is 4 times lower. The costs are also lower: nodes gossip about fewer files when share storage summaries than when share experience. That is, nodes request many more files than store, which is different than in D0.

Kazaa

The Kazaa traces show the same decrease over time of the hit rate in the `1st` mapping scenario (Figure 5.26). The total hit rate in the random mapping case is larger than for the Web traces, mainly due to a larger effect of the largest cluster (20 to 60%, as shown in Figure 5.28). Also, the hit rate increases more significantly in Kazaa with the increase in interval.

Not counting the queries for local storage, the hit rate due to information dissemination is from 38 to 68% for `1st` mapping, 44 to 74% for `any` mapping, and 21 to 59% for `random` mapping. However, the influence of the largest cluster is again significant: without it, the hit rate is up to 16% for `1st` and `any` mapping and around 0% for `random` mapping.

Compared with the other communities, dissemination costs in Kazaa are the lowest. Uncompressed information requires 21MB storage space per node and 10MB of data sent by each node during 8-hour intervals (the average number of files stored per node is under 100, as seen in Table 3.1). When compressed information is disseminated, these costs decrease to 140KB storage space per node and 700KB data sent per node per 8-hour interval.

5.3 Requirements Revisited

In Section 4.1.1 we specified a set of requirements, many emerging from scientific communities. This section discusses how each requirement is met (or not) in FLASK.

5.3.1 File Insertion and Removal

FLASK deals with newly inserted files in an implicit way: since nodes gossip information about locally stored files and their recent experience, newly inserted files will be naturally advertised in the local cluster. If Bloom filters are used, the insertion of a new file requires the local Bloom filter be updated. This is a simple operation, since

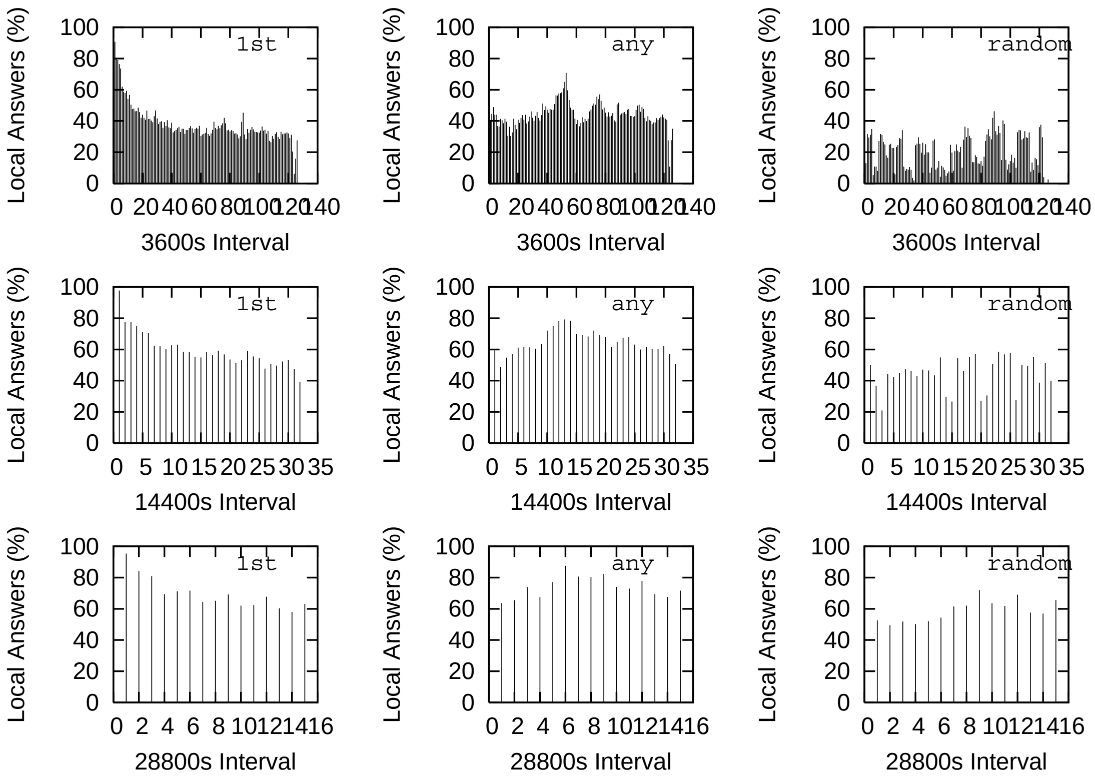


Figure 5.26: Average percentage of requests solved based on information disseminated within Kazaa triad-labeling clusters: 1-, 4-, and 8-hour intervals, 1 shared request.

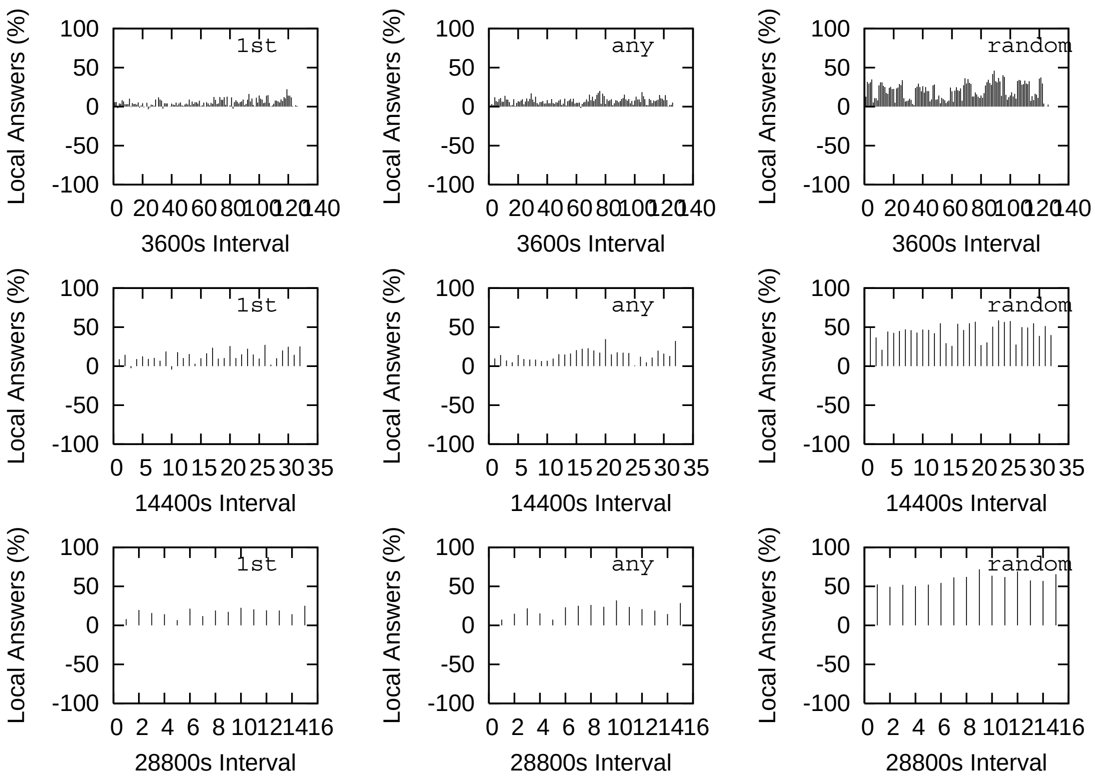


Figure 5.27: The effect of the largest cluster (triad labeling, 1-, 4- and 8-hour intervals, 1 shared request) on average hit rate in Kazaa: the difference between the average hit rate and the average hit rate for all except the largest cluster.

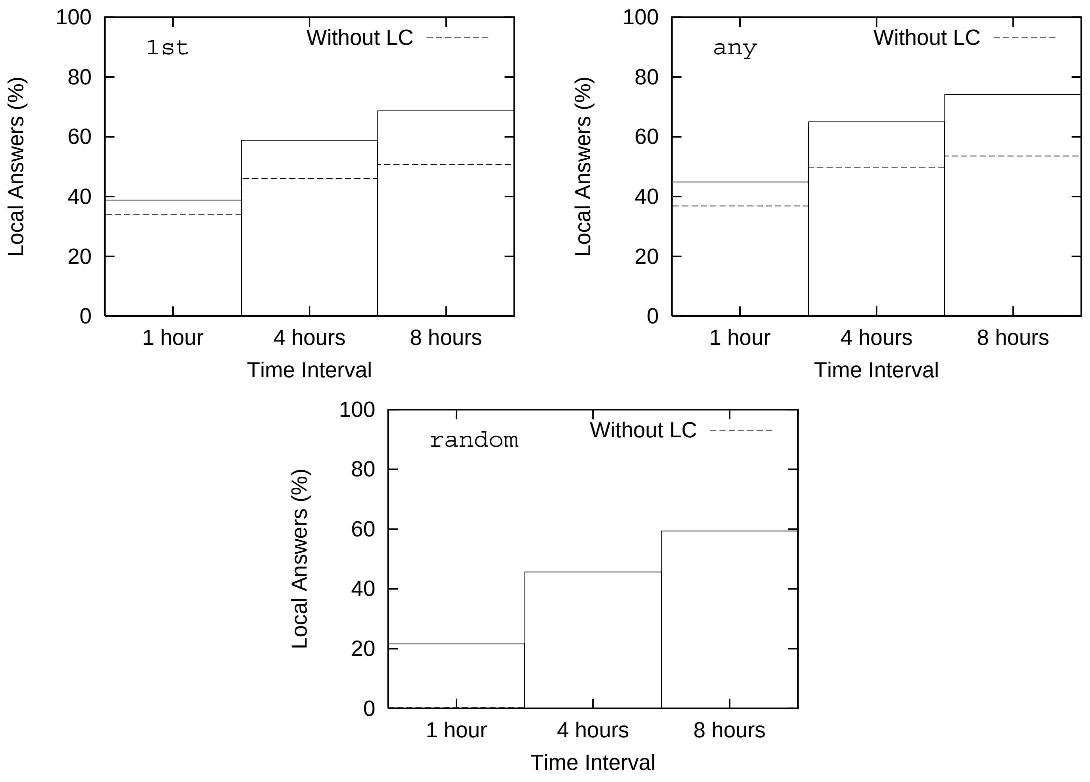


Figure 5.28: Average percentage of local answers with triad labeling on Kazaa traces.

the representation of the new file is to be computed by applying the hash functions: then only an OR is needed to get the new filter.

Removing an item from a Bloom filter is more costly, as it requires entirely re-computing the filter. Methods to avoid this overhead were employed by maintaining bit counters [36]. Furthermore, if file insertion and removal are frequent, the filter can be recomputed at regular intervals, to include multiple updates at once.

5.3.2 Node Volatility

Node volatility is implicitly accommodated by the soft state mechanisms and the semi-unstructured overlay. In structured networks like those based on DHTs, the joining or departure of a node implies rewiring the overlay and migrating some data to other nodes. These costs are justified by the guarantees for fast file location. FLASK has the advantage of unstructured networks such as Gnutella: the departure of a node is observed after the node did not communicate for a while. A node joins at any point in the network and slowly migrates towards its group of interest.

Information does not live for longer than 2τ in FLASK. During this interval, inaccurate information may lead to false positives (for files stored on nodes that just left or failed). Also, there is a latency of τ associated with propagating file information from a newly arrived node. The value of τ can be chosen to balance information accuracy and costs.

5.3.3 Scalability with the Number of Files

Bloom filters attenuate the effect of increasing the number of files at the cost of an increase of false positives. Using Bloom filters, therefore, not only reduces bandwidth consumption but also gives better scalability with the number of files. However, this also requires a smart choice of the Bloom filter size m at the network bootstrap.

Assuming the ideal false positive rate is under 0.01% and 10 hash functions are used, than the size of the Bloom filter can be chosen to 3 bytes per entry. Assuming each node shares not much more than 1000 files, this gives a 3 KB per Bloom filter to represent all the files stored on a node. The size of the Bloom filters is fixed: a much lower number of files would not fully take advantage of the space allocated. A larger number of files would lead to a decrease in the available number of bits per entry: for example, if the number of files turns out to be twice as much as the predicted maximum (i.e., 2000 files stored per node), this leads to 1.5 bytes per entry. From Figure 4.5 we see that this increases the false positive rate to about 1%. A better provisioning (that is, more room per entry in Bloom filters) allows for more graceful degradation of performances but increases the communication costs.

Assuming 1000 clusters of 100 nodes each and 1000 files per node, this allows for the sharing of hundreds of millions of files.

5.3.4 Heterogeneity

Multiple studies have shown that nodes in a large-scale, wide-area, multi-institutional network are highly heterogeneous in both resource capabilities [103, 40] and behavior [4]. The main resources consumed in FLASK are storage and communication. The behaviors that may affect FLASK (and, in fact, any P2P system) are node participation and activity (in number of file requests).

In the following we discuss how the heterogeneity along these dimensions impact FLASK performance: what if some nodes are weaker and cannot contribute the necessary resource requirements? What if some nodes send huge numbers of requests?

1. Limited storage space. While storage requirements are not significant, it is still possible that some nodes might not have that space available at some moment. If a node fails to record the information disseminated within its own cluster, it will unnecessarily forward requests to other clusters. These requests might still be answered, although with a higher response time. However, a weak node will affect only the requests that come to it, without impairing the performance of the other nodes.

2. Limited communication bandwidth. Failing to disseminate one's own information increases the rate of false negatives and the query latency for the files stored locally: queries will be unnecessarily forwarded to different clusters. Failing to disseminate other node's information may lead to incorrect assumptions on that node's availability. However, because of the reliability of the gossip mechanism (information is often redundant since it travels on multiple paths), this effect is significantly alleviated.

3. Limited participation. Some nodes participate longer than others. In DHTs, the departure of a node implies overlay rewiring and data migration that affects $O(log(N))$ of the N nodes. In many cases, this is a small price to pay for low response latency and search guarantees. However, this price becomes significant if nodes depart (or fail) frequently as it seems to be the case in deployed P2P systems [18].

   In FLASK, as discussed previously, intermittent node participation does not impose costs on other nodes.

4. High activity. The high activity of a node in FLASK may be attenuated by the information previously disseminated, since many requests may be answered this way. That is, only a part of its requests will be propagated. The more active a node is, the larger its cluster is likely to be (since it is likely to have many neighbors in the data-sharing graph, therefore in the overlay). To guard against too large clusters, a good clustering algorithm is needed.

5.3.5 Publishing Control

A frowned-upon assumption in DHT-like overlays is the node's willingness to cooperate: a node is assumed to store storage summaries of other nodes. This problem is overcome in FLASK since nodes are responsible for advertising one's own files.

5.3.6 Support for Collections

As Figure 3.1 presents, requests in D0 are for collections of files, a feature characteristic to scientific communities. Figure 5.29 presents the support for collections with experience-sharing in D0: it presents empirical cumulative distribution functions for various intervals.

The results show that 40 to 50% of collections have all their files disseminated within the same cluster. About 40% of requests for collections of files found none of the files locally. On average, 44 to 60% of the requests from a collection are in the cluster due to experience sharing. In file location solutions based only on request propagation, a request for a set of N files translates into N requests for one file. In our case, a request for a set of N files propagates on average less than $\frac{N}{2}$ requests into the network.

Interestingly, the support for larger collections is better: collections with more than half of their files found locally are larger, on average, than collections with less than half of their files found locally (Table 5.6).

Table 5.6: Average size (in number of requested files) of collections for which more than 50% files (respectively less) are found locally. Results from multiple data-sharing graph definitions (τ, μ) are showed.

Interval	$\mu = 100$ files		$\mu = 500$ files	
τ	$> 50\%$	$< 50\%$	$> 50\%$	$< 50\%$
7 days	162.52	123.24	203.59	101.36
14 days	154.02	121.60	185.16	144.16
21 days	139.79	94.71	153.32	108.70
28 days	83.20	305.90	50.78	164.94

Figure 5.30 presents the CDF for collection support when storage information is disseminated within clusters. Fewer collections (less than 20%) are fully supported within a cluster: this is because storage does not reflect (changing) user interest. This explanation is best supported by the random mapping example (Figure 5.30 right), where more than 70% of the collections find no files in the local cluster.

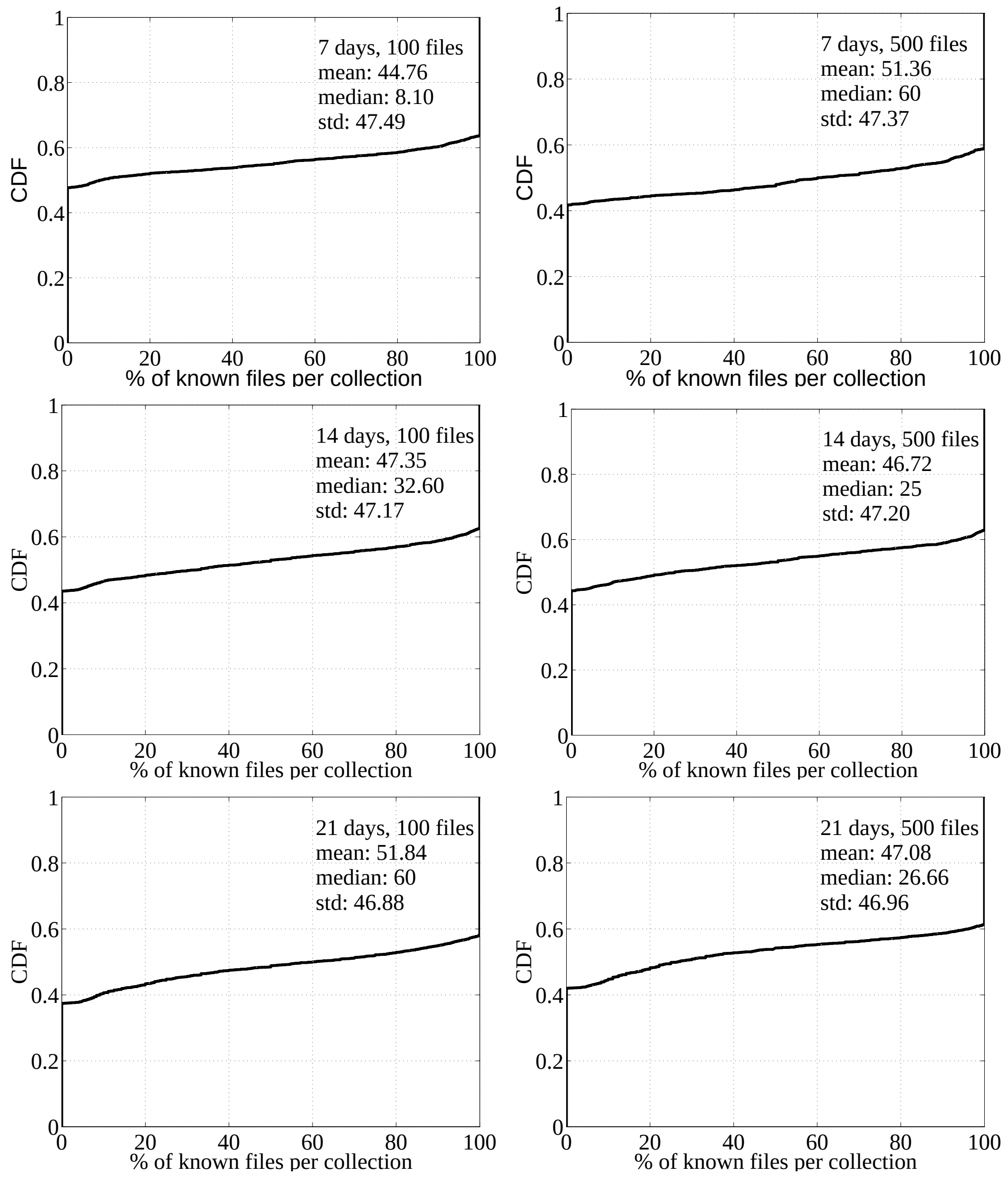
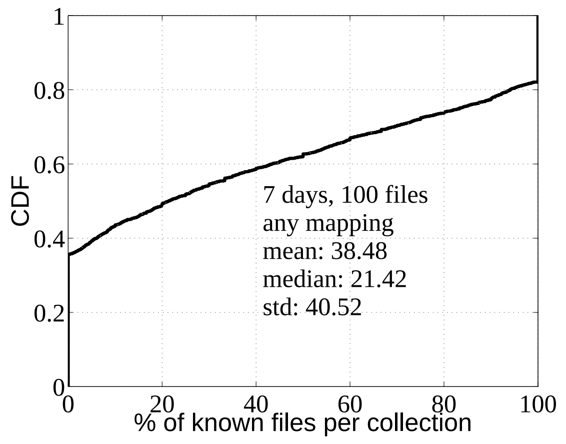


Figure 5.29: Cumulative distribution function of requests per collection found in the local cluster after experience dissemination. A collection is the set of requests sent to support the same computational task (or *project*, in the D0 terminology).



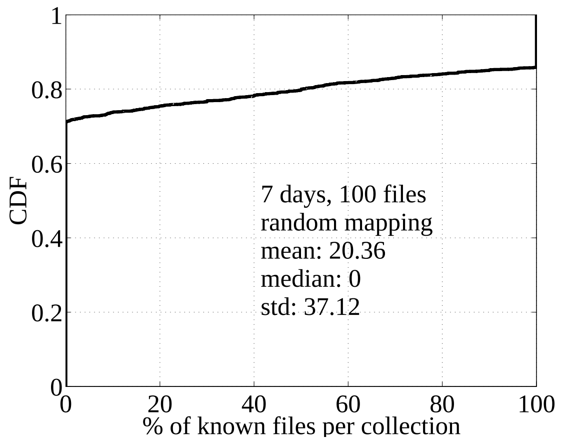


Figure 5.30: Support for requests for collections of files with storage summary dissemination (empirical cumulative distribution).

5.3.7 Approximate Matches

Approximate matches or keyword searches are supported only if no compression is used. As discussed previously, this option is more costly in terms of storage space and communication costs, but it is still a viable solution in many cases.

FLASK has an advantage over search mechanisms in unstructured networks (such as Gnutella) in supporting approximate matches: a larger number of matches are found faster, due to the larger amount of information on nodes.

5.3.8 Not a Panacea

FLASK is designed with particular requirements and user behavior in mind: like most such mechanisms, it does not satisfy all requirements that various user communities might have. In the following we discuss what FLASK does not provide by itself (and how some requirements may be satisfied with additional mechanisms):

- Guarantees that files are always located. FLASK components rely on probabilistic mechanisms: there is a chance that existing files may not be located. However, it supports better the location of "the needle in the haystack" than other mechanisms on unstructured networks, such as Freenet or Gnutella: the less popular information (the "needle") also gets duplicated (by dissemination), although potentially less than the "hay".

- Lack of anonymity and privacy. Unlike many mechanisms that aim to wipe off usage history, FLASK *relies* on and exploits the sharing of information. In FLASK information about who accessed what files is exposed in multiple ways:

first, during the overlay construction, nodes get to learn who else accessed the files they accessed. Second, from the information dissemination component, many users are provided with information about what data other users have accessed. This lack of privacy guarantees may be unacceptable even in some scientific (e.g., competing) communities. While there may be ways around this problem, additional mechanisms must be incorporated in FLASK.

- Attribute-based searches. Even in file-sharing collaborations, Grid users often require attribute-based searches. File metadata typically specifies creation time and authorship, creation procedure such as experimental parameters for a particle accelerator or code version, file size, data format, and so on. File metadata may have considerably storage requirements. Moreover, file metadata cannot be compressed with techniques such as Bloom filters, that support membership queries. However, unlike generic resources, file metadata are rather static, therefore dissemination may have considerable benefits. Nevertheless, the storage and communication costs imposed by disseminating metadata information may be prohibitively high.

5.4 Summary

We explored the experimental space along two variables: the definition of data-sharing graph (namely, different values for τ) and the type of information disseminated. We studied the dissemination of two types of information: experience and storage summaries. Experience dissemination is a form of short-lived cache sharing. Storage summary dissemination is advertising one's own resources in a cluster of common interest. We evaluated design ideas using real traces from three file-sharing communities: a scientific community, the web, and a P2P file-sharing community. The performance metrics evaluated are hit rate due to information dissemination and communication and storage costs.

Experience sharing gives good results: pessimistic evaluations of the average hit rate (that is, ignoring the effect of the largest cluster) leads to hit rates between 30% and 65%. Two costs are associated with information dissemination: storage and network costs. In the case of experience sharing, the storage costs are reasonable even for longer time intervals τ, up to 48MB per node. Network costs are relatively high in some cases (for example, for the Web traces) when information is not compressed. However, using Bloom filters to compress information leads to significantly lower network costs.

Compared to experience sharing, file summary sharing performs poorly: hit rates are lower (on average around 10%) despite higher storage and communication costs. This is mainly due to the disconnection between the files stored on a node and its file interests: in our experiments, the storage is fixed, that is, the mapping of files onto nodes does not evolve over time, as interest in files do. Moreover, there is only one copy for each file in the system. We opted for this setup in order to eliminate other influences, such as those of multiple file replicas or replica migration.

We have evaluated the implications of disseminating either recent experience or local storage summaries. Disseminating only experience does not support the discovery of newly inserted files or of less popular files ("needles in the hay"). Disseminating only storage information is sufficient for locating newly inserted and less popular files, but its performance depends on other aspects of data management, such as file replication (caching) or distribution of files on nodes. Either approach can be used to complement another file location technique: for example, disseminating recent experience in interest-based clusters can significantly reduce the load on a central index while the central index will ensure that new or unpopular files are always located.

However, FLASK needs the dissemination of both types of information. In this case, the hit rate will be higher than the highest hit rate from either case. However, the costs will also be higher than the highest storage and communication costs. It is here that Bloom filters show their power: not only that compressing information using Bloom filters significantly reduces the costs, but the mechanism is more scalable. The increase in communication and storage costs when both types of information are disseminated is small, since the extra information will mostly fill in the Bloom vectors, without taking new space.

To conclude, the main results presented in this section are:

- Information dissemination in interest-based clusters has significant benefits for file-location performance and reasonable costs.

- Under our experimental assumptions, disseminating node experience is more efficient than disseminating storage summaries. However, disseminating storage summaries may perform significantly better if the distribution of files on nodes reflects user interests. Simple mechanisms can do this: for example, caching files on the nodes that requested them. Other approaches can also be beneficial, such as usage-independent file replication (often required for robustness and availability).

- Compressing information with Bloom filters significantly reduces communication and storage costs.

- Off the three mappings of files on nodes that we considered, random mapping leads to the lowest hit rate. This result confirms the intuition that maintaining the relationship between local storage and local user interests is important for good performance from storage summary dissemination. No significant difference in average performance is found between `any` and `1st` mappings.

- FLASK satisfies the requirements identified in Section 4.1.1. Off all, specific to scientific communities is the requirement for collection support: sets of files are requested at once as input data for computations. 40% to 50% of these requirements for sets of files are located easily in one local lookup.

CHAPTER 6
LESSONS FOR RESOURCE DISCOVERY

In Chapter 2 we decomposed the resource discovery problem into four basic components: membership, overlay function, preprocessing, and request processing. Based on the patterns discovered in Chapter 3, we proposed in Chapter 4 a file-location mechanism. File location is a simplified instance of resource discovery: the only resources shared are files, uniquely identifiable by their single attribute, their filename. However, FLASK does not use filenames to build search-efficient structures, such as distributed hash tables. This design approach allows FLASK ideas to remain applicable to the resource discovery problem, where resources have multiple and dynamic attributes.

In this chapter we discuss how FLASK ideas can be applied to the general resource discovery problem. The discussion focuses on two aspects. First, we present the correspondence between the four components of a general resource discovery mechanism and the FLASK components. Second, we discuss the applicability of FLASK to the general resource location problem.

6.1 FLASK in the General Resource-Discovery Framework

In Chapter 2 we analyzed various resource discovery techniques that rely mainly on the request propagation component. Based on large-scale emulations, we concluded that in large unstructured networks, request propagation alone may lead to poor performance.

In FLASK, information dissemination takes a significant amount of load off the request propagation component: less than 50% of requests need to be propagated into the network. The information dissemination component in FLASK corresponds to the preprocessing component of the general resource discovery framework. The design of the preprocessing component, however, influenced the design of the other three components. In the following we discuss how each of these resource discovery components is designed in FLASK. The order in which they are discussed here does not follow the order in which they were introduced in Chapter 2, but allows for better readability.

- The preprocessing component in FLASK includes cluster identification and information dissemination. In addition, the preprocessing component is responsible for rewiring the overlay to adapt to changes in user interests.

- The request processing component corresponds to the request propagation rule: requests are forwarded to nodes that are not part of the local group of interest. An important distinction from the request propagation solutions studied in Chapter 2 is that the destination nodes need not be direct neighbors of the sender node. From this perspective, request propagation in FLASK is a hybrid between guided and random walk searches: requests are forwarded outside the local cluster but it is not specified to which cluster they should be forwarded.

- The overlay function in FLASK is responsible for building the data-sharing graph based on user interests: it connects nodes with similar file interests.

- The membership component in FLASK has two components. One component specifies how nodes join the FLASK network; another component functions at the cluster level: it maintains information about the current peers in a cluster. The membership mechanisms in FLASK are borrowed from previous research results and have been selected based on the design of the other components.

6.2 FLASK as a Resource Discovery Solution

The most important result proven by FLASK's design and performance is the clear benefits of information dissemination. By shifting load from the request processing component to the preprocessing component, FLASK considerably improves response latency and scalability with respect to the number of both requests and participants. However, it is not straightforward to generalize FLASK to locate diverse resources that include computers, network, storage and instruments.

Multiple distinctions between the assumptions made in FLASK and the general context of resource discovery result from the type of resources shared: files are a particular type of resource with specific sharing and attribute characteristics.

FLASK considers files have only one, immutable attribute, their filename. In the general resource discovery problem, resources are often described by multiple attributes. Moreover, interest in one resource can be expressed by any subset of its attributes. For this reason, common interests are harder to identify in the case of general resources: for example, does a user interested in a computer with 2 GB of memory have the same interest as a user who requests a Linux machines with at least 2 GB of memory, connected by at least 10 Mbps?

Another difference comes from the sharing characteristics of files and general resources: while users may share a large number of files each, they will most likely have a small number of other resources to share. This significantly finer granularity may not be most efficient when disseminating information on computational resources.

Another challenge comes from the volatility of general resource attributes. Even if files are frequently inserted into and removed from the system, they live longer than, for example, a machine with CPU load of 5%. Disseminating potentially volatile resource descriptions may lead to inaccurate, misleading information. Consequently, resources need to be advertised by their more static attributes, such as operating system, CPU type, and total memory.

Finally, there are intrinsic distinctions between data and computational resources when seen as goods. Goods can be classified as consumable (such as oil) and non-consumable (such as lighthouses): a larger number of oil consumers decreases the available quantity of oil; a larger number of boats does not decrease the utility of lighthouses and does not increase the cost of their operation. Files are closer to non-consumable goods: the access to file is limited by available bandwidth, but data itself remain available no matter how many users download it. Computational resources are closer to consumable goods: simultaneous access to computational cycles limits the utility. Consequently, nodes may be less willing to cooperate by disseminating resource location information when they compete for resources.

However, designing and evaluating FLASK suggested a powerful idea: grouping resource providers in such a way that their collaboration better serves their users' interests. While the mechanisms used in FLASK for grouping nodes may not be appropriate for locating general resources, the intuition remains applicable. However, instead of grouping nodes with same interests, in this scenario it seems more efficient to group nodes with resources that complement each other's needs. In the following such a strategy is sketched; its rigorous design and evaluation remains an important direction for future work.

The architecture is that of the general resource discovery framework: nodes provide information about resources from a site and act on behalf of the users on that site. As already mentioned, nodes create and maintain pairwise relationships with other nodes such that their aggregated resources serve their user community better than the nodes' individual pools. These relationships are repeatedly re-evaluated and updated to adapt to changing user interests.

This approach naturally accommodates requests for aggregated resources: for example, if local users frequently request resources located far in network distance, then the local node looks for distant peers and negotiates a collaboration. Collaborations may be anything from mutual exchange of resource descriptions to promises for preferred resource allocation to reservations.

CHAPTER 7
DISCUSSION

This thesis focuses on resource discovery in Grid environments of the scale and lack of reliability of today's P2P networks. The characteristics of this target environment require solutions that are fully decentralized, scalable with the number of users and resources, and tolerant to intermittent resource participations (whether voluntary or due to failures). We define the solution space for resource discovery mechanisms by proposing a taxonomy. This taxonomy proves to be a useful tool for discussing previous work but does not reduce the size of the (large) solution space. Our approach to finding efficient design solutions is to analyze and exploit usage patterns. By focusing on one instance of resource-sharing environments, namely file-sharing systems and examining user behavior in different communities, we discover an important usage pattern: users naturally form interest-based groups. This pattern has potential for system design: we design a mechanism for locating files that exploits and benefits from this naturally emerging pattern. However, file location is a simplified instance of resource discovery: nevertheless, important lessons for resource discovery are revealed by the design and evaluation of this file location mechanism. The most important such lesson is the proof that disseminating information to the right part of the network offers considerable performance advantages.

7.1 Contributions

The main contributions of this thesis are:

1. **A formulation of the resource discovery problem** and its requirements **from a novel perspective** that unifies two distinct instances of resource-sharing environments, namely Grid and peer-to-peer. Looking at resource discovery from the perspective of these two environments enables a useful exchange of experience between previously disconnected communities: for example, Grid solutions for resource discovery have not focused on scalability or support for intermittent participation in the absence of central control; previous P2P solutions on resource discovery have not focused on attribute-based searches, but considered the ideal case of searches for globally unique identifiers. If Grids are to grow in the number of users and resources, they need to provide scalable mechanisms and to support intermittent resource participation. If P2P are to go beyond file-swapping applications, they need to support the sharing of a variety of resource types, many not identifiable by name.

2. **A general resource discovery framework**. This framework, based on four building blocks, provides a common description language for various resource discovery solutions and defines the solution design space. It also offers the opportunity of better understanding and employing previous research results into the design of each of the components.

3. **A scalable emulator for resource-sharing environments** and mechanisms for evaluating resource discovery solutions. For the emulator design we identified and modeled the main sharing and usage properties. For example, we modeled various sharing characteristics, such as fair and unbalanced resource sharing (where the number of resources shared per node is constant or, respectively, highly variable) and various user request distributions. We emulated up to 32,768 nodes (2^{15}), where a node provides information about the resources shared by one administrative organization. Using this emulator, we evaluated a set of resource discovery mechanisms based solely on request propagation in various sharing environments. In addition to quantitative estimates of costs and performance, we learned two important things. First, request propagation alone performs poorly in large-scale unstructured networks. Second, the request propagation performance and, consequently, that of the resource discovery service it serves, are highly dependent on the sharing characteristics.

4. **Evidence that** correctly evaluating **user behavior** (including both resource usage and resource sharing) **can yield significant performance advantages**. This evidence is provided via two convergent paths. The first path shows experimentally that for different sharing environments, different discovery strategies work better: for example, if users provide resources uniformly (that is, about the same number and of uniformly distributed types), simple techniques based on random walks work well. If the sharing is highly skewed, with some nodes providing many resources and most being free riders, more sophisticated resource propagation techniques are necessary.

   The second path shows that real traces can suggest design solutions by revealing patterns that can be exploited. Therefore, not only that performance depends on usage characteristics, but understanding usage characteristics can suggest appropriate design solutions.

5. The unearthing of **a previously unknown usage pattern in real file-sharing communities** that reveals commonality of user interests in resources. We discovered this pattern by proposing a new way to look at data: a structure called "the data-sharing graph." We studied this structure on traces from three diverse file-sharing communities: a scientific collaboration (The D0 Experiment), the Web, and a P2P music-swapping system (Kazaa). In all cases, the data-sharing graph was shown to be a small-world graph. The small-world

graph topology proves that users form interest-based groups and these groups are relatively close to each other in graph distance.

6. **A file-location mechanism** (FLASK) that exploits this emergent pattern in user behavior and satisfies the requirements of a mixed Grid–P2P scenario. FLASK identifies clusters of users with same interest in data, disseminate information within clusters, and propagates requests between clusters. To this end, we propose a decentralized technique that mirrors user interests in data onto an overlay structure and adapts the overlay to changes in user interests. We also propose a mechanism for identifying clusters based on local information only. The dissemination of file location information within clusters of interest has significant performance benefits: based on real traces, we show that more than 50% of the requests can be solved based on the information previously disseminated. Moreover, we show that FLASK satisfies the requirements raised by typical Grid communities: collections of files are efficiently located; users maintain publishing control; the mechanism is scalable with the number of nodes and files in the network; heterogeneity does not impair the overall performance of the system; and, finally, file insertion and removal as well as intermittent user participation are supported.

7.2 Future Research

Important problems in resource discovery remain to be solved. Some research directions are a natural continuation of this thesis, others are more general problems in resource discovery.

The discovery of small-world patterns in data-sharing graphs leads to three future research directions. The first covers the particular improvements to the FLASK components that have been described in the previous chapters.

The second direction investigates in more detail the feasibility of FLASK-like solution for the general problem of resource discovery. As already mentioned in Chapter 6, an important lesson from FLASK is the impact of information dissemination on search performance. At the same time, challenging problems are emerging from the various requirements formulated for resource discovery. One such requirement is support for aggregated resources: for example, requests for 100 computers connected via network bandwidth higher than 10Mbs. A centralized solution can solve this problem simply, but centralized design is not always possible or efficient. Information dissemination techniques such as those proposed by FLASK seem to be one direction to pursue.

The third direction is a departure from the subject of this thesis and aims to answer the question: In what other problems in large resource-sharing environments can the emergent, interest-based clustering be exploited for better performance? One example may be data replica placement to better serve a large community of users:

identifying clusters of interest can be one solution. However, vicinity in network metrics become an important factor in determining the feasibility of such a solution.

An important research problem in distributed systems is identifying the relation between resource discovery and other services. In this thesis, we looked at resource discovery as a service in itself, but its existence is often required in support to other services, such as scheduling. How decoupled should these services be? Fully decoupled services—the approach taken in this thesis—have clear advantages. For example, a stand-alone resource discovery service can be used by multiple services, not only for scheduling. In itself, resource discovery can be a valuable tool for evaluating virtual organizations: users often need browsing capabilities to asses the diversity or the availability of the resources provided by a virtual organization. However, a decoupled resource discovery component introduces latencies that, in the context of highly dynamic resource attributes, may hinder the performance of dependent services, such as scheduling.

Alternative approaches that incorporate resource discovery into more complex services have been proposed. In SHARP [45], for example, the discovery of resources is an inherent effect of the peer-to-peer service agreements. As a consequence, only potentially available resources are discovered and their discovery can be promptly utilized. This mechanism trades off some of the requirements of resource discovery for efficient, flexible, and decentralized service negotiation. Better understanding of the necessary tradeoffs is an important problem in large-scale resource sharing environments.

REFERENCES

[1] Abello, J., Pardalos, P., and Resende, M. On maximum clique problems in very large graphs. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science 50* (1999), 119–130.

[2] Abramson, D., Sosic, R., Giddy, J., and Hall, B. Nimrod: A tool for performing parameterized simulations using distributed workstations. In *4th IEEE Symposium on High Performance Distributed Computing (HPDC-4)* (1995).

[3] Adamic, L., Huberman, B., Lukose, R., and Puniyani, A. Search in power law networks. *Physical Review. E 64* (2001), 46135–46143.

[4] Adar, E., and Huberman, B. A. Free riding on Gnutella. *First Monday 5*, 10 (2000).

[5] Aiello, W., Chung, F., and Lu, L. A random graph model for massive graphs. In *The 32nd Annual ACM Symposium on Theory of Computing* (2000), pp. 171–180.

[6] Albert, R., and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of Modern Physics 74* (2002), 47–97.

[7] Allen, G., Dramlitsch, T., Foster, I., Goodale, T., Karonis, N., Ripeanu, M., Seidel, E., and Toonen, B. Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In *SC'2001* (2001), ACM Press.

[8] Anderson, D. P., Cobb, J., Korpella, E., Lebofsky, M., and Werthimer, D. Seti@home: An experiment in public-resource computing. *Communications of the ACM 45*, 11 (2002), 56–61.

[9] Anderson, D. P., and Kubiatowicz, J. The worldwide computer. *Scientific American*, 3 (March 2002 2002).

[10] Annis, J., Zhao, Y., Voeckler, J., Wilde, M., Kent, S., and Foster, I. Applying Chimera virtual data concepts to cluster finding in the Sloan Sky Survey. In *SC'2002* (2002).

[11] Anstreicher, K., Brixius, N., Goux, J.-P., and Linderoth, J. T. Solving large quadratic assignment problems on computational Grids. *Mathematical Programming 91*, 3 (2002), 563–588.

[12] Avery, P., Foster, I., Gardner, R., Newman, H., and Szalay, A. An international virtual-data Grid laboratory for data intensive science. Technical Report GriPhyN-2001-2, 2001.

[13] Ballintijn, G., Steen, M. v., and Tanenbaum, A. Scalable naming in global middleware. In *13th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-2000)* (2000), pp. 624–631.

[14] Barabási, A.-L. *Linked: The New Science of Networks.* Perseus Publishing, 2002.

[15] Barabási, A.-L., and Albert, R. Emergence of scaling in random networks. *Science 281* (1999), 509–512.

[16] Barabási, A.-L., Albert, R., and Jeong, H. Scale-free characteristics of random networks: The topology of the World Wide Web. *Physica A 286* (2000), 69–77.

[17] Barford, P., Bestavros, A., Bradley, A., and Crovella, M. Changes in web client access patterns characteristics and caching implications. Tech. Rep. BUCS-TR-1998-023, Boston University, 1998.

[18] Bhagwan, R., Savage, S., and Voelker, G. Understanding availability. In *2nd Internernational Workshop on Peer-to-Peer Systems (IPTPS'03)* (Berkeley, CA, 2003), I. Stoica and F. Kaashoek, Eds., Springer-Verlag.

[19] Bloom, B. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 7 (1970), 422–426.

[20] Boeing proxy logs, ftp://researchsmp2.cc.vt.edu/pub/boeing/boeing.990301-05.notes.

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

[22] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. Web caching and Zipf-like distributions: Evidence and implications. In *InfoCom* (New York, NY, 1999), IEEE Press.

[23] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. Graph structure in the web. *Computer Networks: The International Journal of Computer and Telecommunications Networking 33*, 1-6 (2000).

[24] Bustamante, F., and Qiao, Y. Friendships that last: peer lifespan and its role in p2p protocols. In *International Workshop on Web Content Caching and Distribution* (2003).

[25] Cancho, R. F., and Solè, R. V. The small world of human language. *Proceedings of the Royal Society B 268* (2001), 2261–2266.

[26] Chandra, T. D., Hadzilacos, V., Toueg, S., and Charron-Bost, B. On the impossibility of group membership. In *15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)* (New York, NY, 1996), pp. 322–330.

[27] Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W., Iamnitchi, A., Kesselman, C., Kunszt, P., Ripeanu, M., Schwartzkopf, B., Stockinger, H., Stockinger, K., and Tierney, B. Giggle: A framework for constructing scalable replica location services. In *SC'02* (2002).

[28] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Designing Privacy Enhancing Technologies* (Berkeley, CA, 2000), vol. 44-66, Springer-Verlag.

[29] Cohen, E., Fiat, A., and Kaplan, H. Associative search in peer to peer networks: Harnessing latent semantics. In *Infocom* (San Fancisco, CA, 2003).

[30] Cohen, E., and Shenker, S. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM* (2002).

[31] Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C. Grid information services for distributed resource sharing. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)* (2001), IEEE Press, pp. 181–184.

[32] The DZero Experiment., http://www-d0.fnal.gov.

[33] Doar, M. A better model for generating test networks. *IEEE Global Internet* (1996), 86–93.

[34] Dorogovtsev, S., and Mendes, J. Evolution of networks. *Advances in Physics 51*, 4 (2002), 1079–1187.

[35] Faloutsos, M., Faloutsos, P., and Faloutsos, C. On power-law relationships of the internet topology. In *SIGCOMM* (1999), pp. 251–262.

[36] Fan, L., Cao, P., Almeida, J., and Broder, A. Z. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking 8*, 3 (2000), 281–293.

[37] Flake, G. W., Lawrence, S., and Giles, C. L. Efficient identification of web communities,. In *6th ACM SIGKDD* (2000), pp. 150–160.

[38] Flake, G. W., Lawrence, S., Giles, C. L., and Coetzee, F. M. Self-organization of the web and identification of communities. *IEEE Computer 35*, 3 (2002), 66–71.

[39] Foster, I. The emergence of the Grid. In *Nature Yearbook of Science and Technology*. Nature Publishing Group, 2001.

[40] Foster, I., and Iamnitchi, A. On death, taxes, and the convergence of peer-to-peer and Grid computing. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)* (Berkeley, CA, 2003).

[41] Foster, I., and Kesselman, C., Eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[42] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. The physiology of the Grid: An open grid services architecture for distributed systems integration. Tech. rep., Globus Project, 2002.

[43] Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications 15*, 3 (2001), 200–222.

[44] Freeman, L. Some antecedents of social network analysis. *Connections 19* (1996), 39–42.

[45] Fu, Y., Chase, J., Chun, B., Schwab, S., and Vahdat, A. Sharp: An architecture for secure resource peering. In *The 19th ACM Symposium on Operating Systems Principles (SOSP)* (October 2003).

[46] Ganesh, A., Kermarrec, A., and Massoulie, L. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers 52* (Febriary 2003).

[47] Girvan, M., and Newman, M. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA 99* (2002), 8271–8276.

[48] The Globus project, http://www.globus.org.

[49] Golding, R. A., and Taylor, K. Group membership in the epidemic style. Technical Report UCSC-CRL-92-13, University of California, Santa Cruz, Jack Baskin School of Engineering, March 1992 1992.

[50] Gribble, S. D., Brewer, E. A., Hellerstein, J. M., and Culler, D. Scalable, distributed data structures for internet service construction. In *4th Symposium on Operating Systems Design and Implementation (OSDI 2000)* (San Diego, CA, 2000).

[51] Gribble, S. D., Welsh, M., Behren, R. v., Brewer, E. A., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, A. D., Katz, R. H., Mao, Z., Ross, S., and Zhao, B. The ninja architecture for robust internet-scale systems and services. *Special Issue of Computer Networks on Pervasive Computing* (2001).

[52] The Grid Physics Network (GriPhyN) project, http://www.griphyn.org.

[53] Gupta, I., Birman, K., Linga, P., Demers, A., and van Renesse, R. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)* (2003).

[54] The Human Genome Project, http://www.nhgri.nih.gov.

[55] Iamnitchi, A., and Foster, I. On fully decentralized resource discovery in grid environments. In *International Workshop on Grid Computing* (Denver, Colorado, 2001), IEEE.

[56] Iamnitchi, A., and Foster, I. A peer-to-peer approach to resource location in grid environments. In *Grid Resource Management*, J. Weglarz, J. Nabrzyski, J. Schopf, and M. Stroinski, Eds. Kluwer Publishing, 2003.

[57] Iamnitchi, A., and Ripeanu, M. Myth and reality: Usage behavior in a large data-intensive physics project. Tech. Rep. TR2003-4, GriPhyN, 2003.

[58] Iamnitchi, A., Ripeanu, M., and Foster, I. Locating data in (small-world?) peer-to-peer scientific collaborations. In *1st International Workshop on Peer-to-Peer Systems (IPTPS'02)* (2002), LNCS Hot Topics series, Springer-Verlag.

[59] Iamnitchi, A., Ripeanu, M., and Foster, I. Data-sharing relationships in the Web. In *12th International World Wide Web Conference (WWW12)* (Budapest, Hungary, 2003).

[60] Iamnitchi, A., Ripeanu, M., and Foster, I. Small-world file-sharing communities. In *Infocom* (Hong Kong, China, 2004).

[61] Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: a review. *ACM Computing Surveys 31*, 3 (1999), 264–323.

[62] Kautz, H., Selman, B., and Shah, M. ReferralWeb: Combining the social networks and collaborative filtering. *Communications of the ACM 40*, 3 (1997), 63–65.

[63] Kermarrec, A.-M., Massoulie, L., and Ganesh, A. Reliable probabilistic communication in large-scale information dissemination systems. Tech. Rep. MSR-TR-2000-105, Microsoft Research Cambridge, October 2000.

[64] Kermarrec, A.-M., Massoulie, L., and Ganesh, A. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems 14*, 3 (March 2003).

[65] Kleinberg, J. The small-worlds phenomenon: an algorithmic perspective. In *32nd ACM Symposium on Theory of Computing, 2000* (Portland, OR, 2000).

[66] Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. Oceanstore: An architecture for global-scale persistent storage. In *9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)* (2000).

[67] Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. Trawling the Web for emerging cyber-communities. *Computer Networks (Amsterdam, Netherlands: 1999) 31*, 11–16 (1999), 1481–1493.

[68] Kutten, S., and Peleg, D. Deterministic distributed resource discovery. In *19th Annual ACM Symposium on Principles of Distributed Computing (PODC'02)* (Portland, Oregon, 2000).

[69] Lee, O., and Benford, S. An explorative approach to federated trading. *Computer Communications 21(2)* (1998).

[70] Legrand, A., Marchal, L., and Casanova, H. Scheduling distributed applications: The simgrid simulation framework. In *3rd IEEE Symposium on Cluster Computing and the Grid (CCGrid'03)* (Tokyo, Japan, 2003).

[71] Leibowitz, N., Ripeanu, M., and Wierzbicki, A. Deconstructing the kazaa network. In *Workshop on Internet Applications* (San Francisco, CA, 2003).

[72] Li. Random texts exhibit Zipf's law-like word frequency distribution. *IEEETIT: IEEE Transactions on Information Theory 38* (1992).

[73] Linden, G., Smith, B., and York, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing 7*, 1 (January/February 2003), 76–80.

[74] Litzkow, M. J., Livny, M., and Mutka, M. W. Condor - a hunter of idle workstations. In *8th Intl. Conf. on Distributed Computing Systems* (San Jose, Calif., 1988), pp. 104–111.

[75] Livny, M. High-throughput resource management. In *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds. Morgan Kaufmann, 1999, pp. 311–337.

[76] Loebel-Carpenter, L., Lueking, L., Moore, C., Pordes, R., Trumbo, J., Veseli, S., Terekhov, I., Vranicar, M., White, S., and White, V. SAM and the particle physics data Grid. In *Computing in High-Energy and Nuclear Physics* (Beijing, China, 2001).

[77] Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. Search and replication in unstructured peer-to-peer networks. In *6th annual ACM International Conference on supercomputing (ICS)* (2002).

[78] Makse, H. A., Havlin, S., and Stanley, H. E. Modeling urban growth patterns. *Nature 377* (1995), 608–612.

[79] Mockapetris, P. Domain names–concepts and facilities. In *RFC 1034*. 1987.

[80] Mutka, M., and Livny, M. The available capacity of a privately owned workstation environment. *Performance Evaluation 12*, 4 (1991), 269–84.

[81] Negra, M. D. Cms collaboration. Tech. Rep. CERN LHCC 94-38, CERN, 1994.

[82] Newman, M. Scientific collaboration networks: I. Network construction and fundamental results. *Phys. Rev. E 64* (2001).

[83] Newman, M. Scientific collaboration networks: II. Shortest paths, weighted networks, and centrality. *Phys. Rev. E 64* (2001).

[84] Newman, M. The structure of scientific collaboration networks. *Proc. Natl. Acad. Sci. USA 98* (2001), 404–409.

[85] Newman, M. The structure and function of complex networks. Tech. Rep. cond-mat/0303516, Los Alamos Archive, 2003.

[86] Newman, M., Forrest, S., and Balthrop, J. Email networks and the spread of computer viruses. *Phys. Rev. E 66*, 035101 (2002).

[87] Newman, M., Strogatz, S., and Watts, D. Random graphs with arbitrary degree distribution and their applications. *Phys. Rev. E 64*, 026118 (2001).

[88] Newman, M., Watts, D., and Strogatz, S. Random graph models of social networks. *Proc. Natl. Acad. Sci. USA 99* (2002), 2566–2572.

[89] Plaxton, C. G., Rajaraman, R., and Richa, A. W. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)* (1997).

[90] Prudhomme, T., Kesselman, C., Finholt, T., Foster, I., Parsons, D., Abrams, D., Bardet, J.-P., Pennington, R., Towns, J., Butler, R., Futrelle, J., Zaluzec, N., and Hardin, J. NEESgrid: A distributed virtual laboratory for advanced earthquake experimentation and simulation: Scoping study. Technical Report 2001-01, NEESgrid, 2001.

[91] Ramakrishna, M. V. Practical performance of Bloom filters and parallel free-text searching. *Communications of the ACM 32*, 10 (1989), 1237–1239.

[92] Raman, R., Livny, M., and Solomon, M. Matchmaking: Distributed resource management for high throughput computing. In *7th IEEE Symposium on High Performance Distributed Computing (HPDC-7)* (1998), IEEE Press.

[93] Ranganathan, K., and Foster, I. Design and evaluation of dynamic replication strategies for a high performance data Grid. In *International Conference on Computing in High Energy and Nuclear Physics* (2001).

[94] Ranganathan, K., Iamnitchi, A., and Foster, I. Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In *Global and Peer-to-Peer Computing on Large Scale Distributed Systems Workshop* (2002).

[95] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A scalable content-addressable network. In *SIGCOMM* (2001), ACM.

[96] Redner, S. How popular is your paper? An empirical study of the citation distribution. *European Physical Journal B 4* (1998), 131–134.

[97] Ripeanu, M., and Foster, I. "a decentralized, adaptive, replica location service". *In proceedings of 11th IEEE International Symposium on High Performance Distributed Compuing (HPDC-11)* (July 2002).

[98] Ripeanu, M., Foster, I., and Iamnitchi, A. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *Internet Computing 6*, 1 (2002), 50–57.

[99] Ripeanu, M., Iamnitchi, A., and Foster, I. Cactus application: Performance predictions in Grid environments. In *European Conference on Parallel Computing (EuroPar)* (2001), vol. LNCS 2150, Springer-Verlag, pp. 807–816.

[100] Ripeanu, M., Iamnitchi, A., and Foster, I. Performance predictions for a numerical relativity package in Grid environments. *International Journal of High Performance Computing Applications 15*, 4 (2001).

[101] Rowstron, A., and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware* (2001), pp. 329–350.

[102] Saito, Y., Karamanolis, C., Karlsson, M., and Mahalingam, M. Taming aggressive replication in the Pangaea wide-area file system. In *OSDI* (2002).

[103] Saroiu, S., Gummadi, P. K., and Gribble, S. D. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking (MMCN)* (San Jose, CA, USA, 2002).

[104] Sendag, R., Chuang, P.-f., and Lilja, D. J. Address correlation: Exceeding the limits of locality. *Computer Architecture Letters 2* (May 2003).

[105] Seti@home: The search for extraterestrial intelligence, http://setiathome.berkeley.edu.

[106] Shirky, C. What is p2p... and what isn't? In *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, A. Oram, Ed. O'Reilly, 2001.

[107] Sloan Digital Sky Survey, http://www.sdss.org/sdss.html.

[108] http://www.slyck.com.

[109] Song, H., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., and Chien, A. The microgrid: a scientific tool for modeling computational Grids. In *Supercomputing* (2000).

[110] Sripanidkulchai, K. The popularity of Gnutella queries and its implications on scalability, 2001.

[111] Sripanidkulchai, K., Maggs, B., and Zhang, H. Efficient content location using interest-based locality in peer-to-peer systems. In *INFOCOM* (San Francisco, 2003).

[112] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM* (San Diego, USA, 2001).

[113] Thomas, M. P., Mock, S., and Boisseau, J. Development of web toolkits for computational science portals: The npaci hotpage. In *9th IEEE Symposium on High Performance Distributed Computing (HPDC-9)* (2000).

[114] van Dongen, S. *Graph Clustering by Flow Simulation.* PhD thesis, University of Utrecht, May 2000.

[115] van Dongen, S. MCL – a cluster algorithm for graphs, http://micans.org/mcl/.

[116] van Renesse, R., Minsky, Y., and Hayden, M. A gossip-style failure detection service. Tech. Rep. TR98-1687, Cornell University, Computer Science Department, 1998.

[117] van Steen, M., Homburg, P., and Tanenbaum, A. Globe: A wide-area distributed system. *IEEE Concurrency* (1999), 70–78.

[118] Vogels, W., R., v. R., and Birman, K. Using epidemic techniques for building ultra-scalable reliable communication systems. In *Workshop on New Visions for Large-Scale Networks: Research and Applications* (Vienna, VA, 2001).

[119] Watts, D., and Strogatz, S. Collective dynamics of 'small-world' networks. *Nature 393* (1998).

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

[121] Wilcox-O'Hearn, B. Experiences deploying a large-scale emergent network. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)* (Cambridge, MA, USA, 2002), P. Druschel, F. Kaashoek, and A. Rowstron, Eds., Springer-Verlag.

[122] Zhao, B. Y., Kubiatowicz, J. D., and Joseph, A. D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. CSD-01-1141, Berkeley, 2001.