Plethora: An Efficient Wide-Area Storage System *

Suresh Jagannathan, Ananth Grama, and Ronaldo Ferreira
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{suresh,ayg,rf}@cs.purdue.edu
Phone: (765) 494 0971, FAX: (765) 494 0739

Abstract

Trends in conventional storage infrastructure motivate the development of foundational technologies for building a wide-area read-write storage repository capable of providing a single image of a distributed storage resource. The overarching design goals of such an infrastructure include client performance, global resource utilization, system scalability (providing a single logical view of larger resource and user pools) and application scalability (enabling single applications with large resource requirements). Such a storage infrastructure forms the basis for second generation data-grid efforts underlying massive data handling in high-energy physics, nanosciences, and bioinformatics, among others.

This paper describes some of the foundational technologies underlying such a repository, Plethora, for semi-static peer-to-peer (P2P) networks implemented on a wide-area Internet testbed. In contrast to many current efforts that focus entirely on unstructured dynamic P2P environments, Plethora focuses on semi-static peers with strong network connectivity and a partially persistent network state. In a semi-static P2P network, peers are likely to remain participants in the network over long periods of time (e.g., compute servers), and are capable of providing reasonably high availability and response-time guarantees. The repository integrates novel concepts in locality enhancing overlay networks, transactional semantics for read-write data coupled with hierarchical versioning, intelligent replication for robustness. While mentioning approaches to other problems, this paper focuses on the problem of routing data request to blocks, while integrating caching and locality enhancing overlays into a single framework. We show significant performance improvements resulting from our routing techniques.

*This research has been supported by the National Science Foundation.

1 Plethora: Introduction and Design Principles

The Plethora project at Purdue University aims to build a wide-area read-write storage repository for supporting a single seamless distributed storage resource. In contrast to many current efforts that focus entirely on unstructured dynamic peer-to-peer (P2P) environments, Plethora focuses on semi-static peers with strong network connectivity and a partially persistent network state. In doing so, it alleviates many of the constraints found in conventional P2P networks, focusing instead on mechanisms for supporting expressive storage access and management semantics, exploiting existing Internet infrastructure, and providing performance guarantees in terms of end-user latencies, global resource utilization, and robustness.

Supporting a wide-area read-write storage system is a critical need for next generation distributed applications. A large number of scientific experiments result in extremely large datasets, either from computational simulations or dense instrumentation. Common examples of such applications include sky surveys (Keck and Sloan Digital Sky Survey), high-energy physics experiments, and remote sensing applications, most of which assume a distributed infrastructure. Storing these large datasets and supporting efficient mechanisms for accessing them is critical for using these datasets. The associated problems of a single large storage structure, caching mechanisms, access semantics, and reliability are addressed by Plethora. An associated problem is one of providing efficient access to distributed data in collaborative environments. These environments are critical components of many scientific and design efforts. Other wide-area applications rely on mobility of services for optimizing performance (for eg., process migration in metacomputing environments [14]). A wide-area storage repository can vastly simplify the design of such mobile services. Finally, many applications require a high degree of availability and robustness from the underlying storage infrastructure. All of these applications provide overarching motivation for Plethora and guide its design principles.

We identify a number of technical challenges that must be overcome to support Plethora efficiently. Due to the application driven nature of the storage infrastructure, it is imperative to define appropriate semantics for data access. The suitability of these semantics is determined by the performance that can be achieved over current wide-area networks. While strong consistency models based on invalidates and eager updates are desirable from an application's viewpoint, they do not scale from a system perspective. For example, due to inherent latency costs in wide-area networks, end-user performance or global resource utilization is likely to suffer under a consistency model that initiates frequent inter-node communication. On the other hand, weak consistency models that weaken the points at which local and global copies of shared data must be synchronized lead to program behavior that is often inconsistent with the assumptions made by well-understood programming models on when updates become visible. A suitable compromise must be designed.

The guiding principle is one of end-user performance (latency and throughput), system scalability (the number of peers participating in supporting an application), and application scalability (the complexity of the application itself in terms of its resource use, the number and sophistication of the transactions it initiates, etc.).

To support these goals, Plethora relies on a novel suite of algorithmic techniques and data models. At the heart of Plethora is an overlay routing technique that exploits existing Internet infrastructure to optimize locality, overlay link dilation (stretch), and congestion (on physical links). The Plethora routing core organizes peers into a local overlay, based on autonomous systems (ASs) and associated proximity, and a global overlay. The local overlay serves to localize requests by providing a scalable and adaptive caching mechanism. We show that this two-level routing infrastructure is capable of significantly better performance, both in terms of end-user latencies as well as global resource utilization. Built on top of this routing core is a hierarchical versioning read-write storage network, which provides a transparent and distributed storage pool. Data written to the pool is stored among peers in the network. The routing mechanism uses the local overlay to efficiently retrieve data that is geographically proximate; data far removed from the source of a query is routed using the global overlay, and cached among nodes in the local overlay. Versioning protocols similar to those used in versioning databases [19, 24] or filesystems [32] are used to implement a consistency model with well-defined behavior. Informally, updates to data are recorded using a hierarchical versioning scheme similar to those used in version control systems [36]. The Plethora infrastructure supports a distributed version tree along with a number of policies for conflict resolution. Applications have the flexibility to choose specific models based on their required access semantics. Finally, robustness and availability are incorporated using novel erasure codes that minimize overhead of decoding blocks and reconstituting codes in distributed environments.

The rest of the paper is organized as follows: we begin with an overview of the Plethora architecture in Section 2. We discuss related projects in Section 3. We then describe three key components: the Plethora routing core, object read-write semantics, and robustness and security.

2 Plethora Architecture and Design

The Plethora architecture is a novel design that combines features of peer-to-peer (P2P) systems, distributed versioning and database systems, and Internet-aware routing and caching strategies. At the heart of Plethora's design is a two-level network overlay. A global overlay spans all nodes in the network and is organized much like other P2P systems, which define a collection of peers that cooperatively provide access to resources such as storage. A local overlay, which provides caching mechanisms, lies on top of this global overlay. Nodes belong to both local and global overlays and

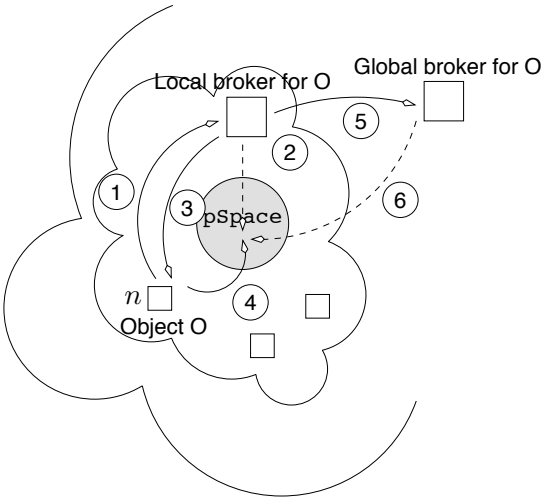
are organized into local overlays based on proximity information, specifically, the Autonomous Systems (ASs) to which they belong. Algorithms for organizing nodes into local overlays are described in greater detail in Section 4.2. Data accesses are first attempted in the local overlay, and forwarded to the global overlay when no cached copy is found. Specifically, if a Plethora node n requests a globally shared piece of data, the data is routed using a distributed (or consistent) hashing scheme to a node n_l in n's local overlay. If n_l does not have the requested data, a query is sent to the global overlay. This two-tiered search mechanism distinguishes Plethora from other P2P schemes in obvious ways. Most notably, it leverages the physical structure and organization of the Internet to provide efficient routing and lookup of shared data.

Concurrent Access Semantics in Plethora.

Plethora is a global object repository upon which classical applications such as filesystems, collaborative environments, publishing, etc. can be built. To support such repositories, appropriate concurrent access semantics must be built on top of the Plethora routing core. These semantics must facilitate desirable characteristics of client performance, global resource utilization, system scalability, and application scalability. We start our discussion by presenting the Plethora concurrent access model.

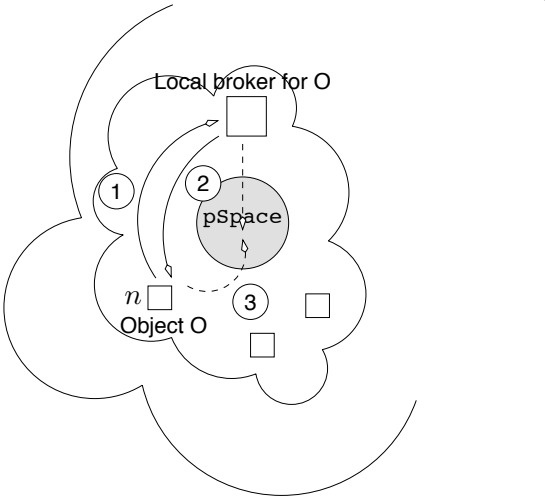
To be a member of a Plethora community, a node n defines two namespaces. The first, called `cSpace` is a local client namespace in which objects such as files are created and modified by applications that execute on the node. The second, called `pSpace` is the space used by Plethora to support shared access semantics. Based on this model, we describe the logical operation of the Plethora object sharing mechanism for the following basic tasks: (i) introducing an object into the network, (ii) acquiring an object from the network, and (iii) updating objects in the network. Plethora supports these tasks using local and global object brokers. These brokers are determined by applying appropriate hash functions to the object handle – a local hash function, whose range is the set of nodes in the local overlay and a global hash function, whose range is the set of all nodes in the Plethora community. Note that these ranges can be easily affected by using appropriate routing table entries in two distinct set of tables without changing the hash function. A local object broker is responsible for keeping track of the shared objects (that hash to it) within the local overlay. A global object broker keeps track of shared objects (that hash to it) in the entire network.

Consider a local object O in node n's `cSpace`, which needs to be initiated into the network. Node n communicates with a broker for the object in the local overlay. The broker copies the object into node n's `pSpace`, generates a local reference to this copy and also communicates this reference to the global object broker along with appropriate metadata. At this point, object O is visible to the other nodes. This process is illustrated in Figure 1(a). To acquire object O from



1. Node n notifies local broker to initiate Object O.
2. Local broker creates reference for O in `pSpace`.
3. Local broker notifies n of this reference.
4. Node n copies O to this location in `pSpace`.
5. Reference in `pSpace` sent to global broker.
6. Global broker records this reference.

(a) Initiating object O into Plethora.



1. Node n requests local broker for Object O.
2. Local broker sends reference to O in `pSpace`.
3. Node n creates reference, copies into `cSpace`.

(b) Searching for object O in Plethora (the case illustrated is one in which object is found in local overlay).

Figure 1: Protocols for object initiation and lookup. The protocol for updation is similar to initiation augmented by aggregation of commits.

the network, a node n first communicates with a local broker for the object. If the object O is cached in the local overlay, a reference to the cached version is returned and a copy subsequently communicated to node d's `pSpace`. This is illustrated in Figure 1(b). If no cached version exists, a search is initiated by the local broker over the global overlay and a copy is fetched by the broker into node n's `pSpace`.

To update a shared object O, it must first be copied into a node's `cSpace`. To make these changes visible to the rest of the network, the node must communicate its intention to have these changes committed to O's local broker. Object O's broker copies this object into node n's `pSpace`. A new version is recorded at the local broker and the existence of this version is propagated to Object O's global broker. In the event that multiple commits arrive at the local broker for the same object, the broker may choose to either ignore all conflicts among these versions, commit each set of updates as a new version, or attempt more sophisticated conflict resolution and commit only a single resolved version. Yet another option is for the local broker to rely on timestamps to discard all updates other than the most recent set. Plethora provides hooks for each of these policies to be implemented at the local broker. The global broker may apply similar strategies to resolve conflicts between multiple local brokers for the same object.

Several observations can be made about the suitability of Plethora's access semantics for wide-area storage systems. First, updates to objects occur entirely locally once the object is copied into the nodes `cSpace`. The applications determine when these changes need to be made visible to the rest of the network. Locality of data access in typical applications implies that such a scheme is ideally suited from the point of view of minimizing update traffic. Just as a node aggregates several object updates into a single request to commit to the local broker, the local broker may aggregate several such requests before propagating the version to the global broker. This hierarchical aggregation is critical for scalability to large number of nodes given constraints on available bandwidth.

Plethora's approach to object management and access is distinguished from other proposed approaches in several important respects. Most notably, the notion of local and global overlay implies that an object has meaning both from the perspective of a local overlay that caches its local neighborhood value, and from the perspective of the local object broker that records its global value. In other words, all nodes within a specific overlay see a consistent view of an object's state; this view may be globally inconsistent for a period of time, and resolved only when state changes are propagated to the global broker. Plethora's hierarchical overlay and routing scheme influences object lookup and access by permitting efficient manipulation of shared objects.

Another novel component of Plethora's software architecture is its use of a versioning system to deal with object conflict resolution and commitment. By supporting versioning semantics as

opposed to a copy/ invalidate protocol, Plethora allows concurrent updates to a shared object, and places the burden of proper conflict resolution on the application. In other words, two clients that commit updates to a shared object may see two versions of the object preserved by the object server. They may choose to reconcile these versions by identifying and remedying conflicts in much the same way as version control systems such as CVS or RCS do, or they may choose to simply perform subsequent modifications based on their version or some other version accessible from the local broker. Our primary goal in these design decisions is to provide scalability by taking advantage of the local overlay structure. Our versioning system provides a semantically clean characterization of object modification without requiring global updates on each commit.

Within this logical Plethora framework, a number of critical design decisions further influence system performance. For example, in actual systems, `pSpace` is distributed across the nodes in a local overlay. There are no explicit bindings between segments of `pSpaces`, the data they hold, and their physical location. In particular, the `pSpace` of a node n can be used to hold all data initiated into the network by node n. Similarly, on updates, committed versions may be copied into local segments of the `pSpace`, thereby generating no network overhead. A number of such design decisions underly an efficient realization of Plethora.

3 Plethora and its Comparison to Related Work

There has been a long history of research in the area of distributed file systems and storage. Plethora's design is inspired by many of these past efforts. Existing systems based on the client-server architecture like AFS [16], NFS [35], xFS [5], Sprite LFS [27] and Coda [17] do not meet our goals of scalability, availability, and network performance. Farsite [3] is a distributed file system that operates over untrusted environments using randomized replicated storage; the secrecy of file contents is ensured using cryptographic techniques, and Byzantine agreement protocols are used to ensure file and directory integrity. The goals of these systems are only partially aligned with ours. Distributed file systems like Coda or AFS are geared towards a client-server storage model. Decentralized systems like Farsite focus on completely untrusted local-area environments (such as a collection of desktops in a corporate campus). Neither consider routing and location services for geographically-dispersed peers, or consistency semantics when multiple copies of data are cached at distributed sites.

A number of researchers have addressed the problem of providing scalable solutions [33, 28, 38] in a P2P environment. Systems such as Chord [33], Pastry [28], and Tapestry [38] provide a simple primitive for name resolution: given a file name, return the IP addresses of the nodes that currently have reference to the file. To support this primitive, these systems rely on a *distributed*

hash table (DHT) abstraction [34, 30, 38], and provide an upper bound on hop-count of $O(\log n)$, where n is the total number of nodes in the network. This upper bound is achieved using a small amount ($O(\log n)$) of routing information per node. Other systems such as CAN [23] support similar primitives, but have different upper bounds on hop-count subject to varying constraints on per-node routing information.

Storage systems that exploit the organization and routing properties of the DHT systems described above have been recently proposed. The Cooperative File System (CFS) [10] is a peer-to-peer read-only storage system built on top of Chord. CFS distributes blocks of data among the nodes participating in the Chord network. Blocks are generally cached to provide better availability and to decrease network latencies. Like CFS, PAST [29] is a read-only peer-to-peer storage system. PAST is built on top of Pastry and it explores the hierarchical routing of Pastry to cache data. Both CFS and PAST use a single overlay network and rely solely on the proximity heuristics implemented in their underlying routing substrates to achieve locality when caching data. Plethora, on the other hand, with a two tier organization tries to group nodes that are physically close in local overlays. The local overlays are used as much as possible as caching domains. Another significant difference is that Plethora allows updates in the data stored in the network using a hierarchical versioning semantics for shared data.

Oceanstore [18, 25], built on top of Tapestry, is a related peer-to-peer storage system. Like Plethora, Oceanstore uses a versioning system to allow updates in data objects. Oceanstore relies on a Byzantine agreement algorithm, proposed by Castro and Liskov [9], to serialize updates in the data objects and to create new versions. Each object has a well defined set of nodes, comprising the inner ring, which serve as primary replicas and which are responsible for maintaining information about the latest version. Objects can also be replicated in secondary replicas. The secondary replicas of a data object are organized in a multicast tree. The inner ring is responsible for pushing updates down the multicast tree to keep the replicas as consistent as possible. Data objects are broken into multiple blocks. Blocks are further encoded using erasure coding to increase redundancy. Once a block is encoded, it is never changed and can be stored in any participating peer of the network. The use of erasure coding improves availability, but imposes an extra overhead. Another potential performance bottleneck is the use of Byzantine agreement protocol since nodes in the inner ring must cooperate closely to provide the guarantees offered by the system.

Pangaea [31] is a wide-area file system that shares many of our goals including high availability, low access latencies, and locality in cached data. As in Plethora, nodes in Pangaea are organized in regions, where a region represents nodes that are physically close in the Internet. To keep nodes grouped in regions, at least one node inside each region must keep information about all regions in the network. As indicated in [31], the maintenance of this information constitutes a major

bottleneck and generates a significant amount of traffic in the network, since nodes must probe each other to collect delay information. Plethora, on the other hand, exploits the organization of the Internet into autonomous systems and uses a global DHT system to manage the different regions. Specifically, the information about which nodes are present in each region is stored in the global overlay and not at a single node. Nodes of a particular autonomous system are grouped in a local overlay. Plethora uses the autonomous system number, a 16-bit identifier, to construct the local overlays. Pangaea also differs from Plethora in the way it handles data consistency. Nodes that contain replicas are connected in a graph that is used to propagate updates. Updates are pushed to replicas and conflicts are resolved after they happen. Users see the changes made by another users only at some unspecified future time. The system provides, therefore, only "eventual" consistency.

4 The Plethora Routing Core

Central to the performance of Plethora is an efficient routing core, designed to: (i) optimize locality of data access by efficient replication, (ii) minimize overlay link dilation (stretch) and network congestion, and (iii) support suitable object access semantics and robustness mechanisms. These objectives are achieved in part by organizing the network into two distinct overlays – a local overlay comprising nodes in close network proximity, and a global overlay used to provide universal location services. This is in contrast to many systems that attempt to build a single overlay that minimizes stretch. This two-level overlay provides several important benefits:

- The local overlay supports efficient caching and the global overlay facilitates efficient object location. It is well-known that there is significant spatio-temporal locality in database and file access [7, 22, 4] and for well-structured parallel and distributed programs in general. By manifesting this observation using the network proximity of the local overlay, we obtain excellent local hit ratios and consequently improve end-user and network performance (in terms of reduced congestion).

- Due to the two-level structure, protocols for joining and maintaining the network are considerably simplified. This is because node joins and departures can be handled largely locally.

- The global overlay provides critical support for building local overlays. Specifically, it provides an incoming node information regarding the local overlay it should join. It also keeps track of the state of the local overlays, issuing split and merge directives as needed.

- The local and global overlays can both be constructed using conventional hash-based techniques, and can thus provide efficient (polylogarathmic time) location services. Furthermore,

critical operations such as merging and splitting local overlays are facilitated by the nature of conventional overlay networks.

We begin with a description of the two-level Plethora routing core and summarize various node and network protocols.

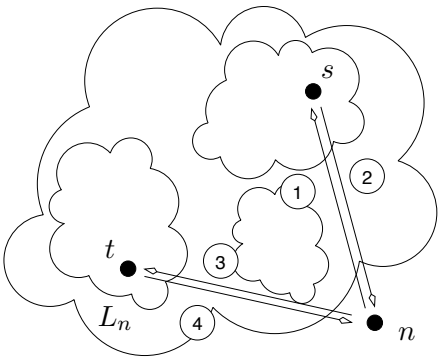
4.1 Locality Mechanisms in Plethora.

Locality of data access is at the core of scalability of Plethora. This has been recognized by many other projects as well, which use various mechanisms for enhancing locality in overlays. Commonly used methods can be classified as either state-based or static. State-based approaches rely on network state information (latencies, hop-count) for situating an incoming peer in an overlay. These methods include location using landmarks and topological inference. In contrast, static approaches rely on known information from the Internet infrastructure to build local overlays. Plethora bases its design on static information for locality. This is motivated by the availability of standardized information, such as IP domains and Autonomous System (AS) membership, across diverse platforms. Using this information efficiently, however, poses considerable challenges.

With the eventual goal of an AS-based node aggregation in mind, we investigated, in the Gnutella trace, the prefix length of CIDR (Classless Inter Domain Routing) blocks in these systems. The trace contains 3,264 different ASs, which constitute a representative sample of the Internet. The average prefix length of the AS's CIDR blocks was a little over 19 bits. This implies that local overlays based on single ASs constitute very small networks, and are unlikely to yield significant improvements from caching. Plethora therefore aims to build local overlays as aggregates of geographically proximate ASs. For each AS, we determine a ranking sequence of all other ASs close to it (this ranking sequence can be computed using static and/or dynamic state information such as delay or number of hops). This sequence determines the aggregation process. Aggregation is controlled by a root node in the global overlay that is responsible for this AS. The population of a local overlay is controlled by merging and splitting local overlays based on this AS proximity information. The precise algorithms for these operations are described in Section 4.4.

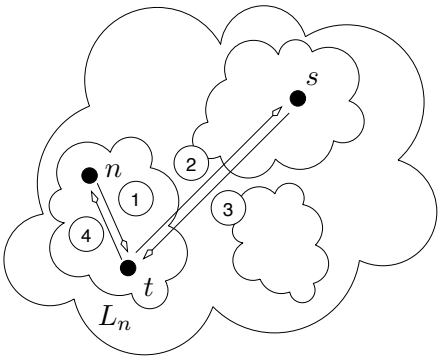
4.2 Building the Plethora Routing Core.

The key to building the Plethora routing core is to use the global overlay to maintain state information for local overlay. Specifically, an incoming node n is mapped to a node s in the global overlay. This node s uses the `whois` database to determine the AS to which n belongs and maps the incoming node to a local overlay L_n (several ASs might comprise a local overlay depending on peer population). Node s responds to incoming node n's request with the id of the local overlay



1. Node n sends a message to $s = h(n)$.
2. Node s sends global routing information to n along with local id. L_n, which n must join (this is determined by Node n's AS).
3. Node n sends a message to $t = h'(n)$ in L_n indicating its desire to join local overlay.
4. Node t sends local routing information to n.

(a) Entering the two–level Plethora routing core.



1. Node n sends a message to $t = h'(n)$ requesting data item d in local overlay.
2. If Node t has a copy of data item d, steps 2 and 3 are skipped. In step 2, Node t sends a request to Node $s = h(d)$ in global overlay requesting data item d.
3. Node s sends a copy of data item d to Node t.
4. Node t caches a copy of data item d in local overlay L_n and forwards a copy to Node n.

(a) Data access in Plethora.

Figure 2: Registering peers and locating data objects in Plethora.

L_n to which it should join. In addition, node s also sends node n the global routing tables. These correspond to steps 1 and 2 in Figure 2(a). At this point, node n has joined the global overlay and knows that it must join local overlay L_n (note that in this framework, it is possible to use additional state information at node s to determine L_n). Node n then sends a message to a selected node t in local overlay L_n (this node can be selected using a hashcode whose range is in L_n) indicating its desire to join the local overlay. Node t responds to this request with local routing information for overlay L_n. This is illustrated in steps 3 and 4 in Figure 2(a). In this manner, an incoming node joins the global location and local caching network.

4.3 Search in Plethora.

The process by which objects are located in Plethora is illustrated in Figure 2(b). An object is first searched in the local overlay using a conventional hash-based query routing scheme. If the object is found, it is fetched from within the local overlay. Otherwise, a query is generated on the global overlay. The object is fetched and cached in the local overlay and forwarded to the requesting

node. While current versions of the Plethora core use a simple LRU replacement policy with fixed buffer sizes, the specific caching mechanism is flexible w.r.t. implementation. The performance of this two level scheme is predicated on two critical factors: local overlays must have a high degree of network proximity for meaningful performance gains, and a significant fraction of all accesses must be satisfied from the local overlay (hit ratio). In extensive experiments using access traces as well as synthetic distributions, we have shown that for realistic network sizes, we can achieve hit ratios in excess of 70%. At these hit ratios, we observe over 50% improvement in peer performance with respect to a Pastry peer. With larger buffers and correspondingly larger peer density in local overlays, these savings are expected to grow considerably.

4.4 Plethora Network Maintenance.

An important task within the Plethora routing core is to ensure that local overlays do not become too large or too small. In both cases, benefits from localization are diluted, and the performance tends to that of a single global overlay (with some overhead). For this reason, Plethora supports two directives – `pSplit` and `pMerge`. A `pSplit` operation splits a single local overlay into two. This is simply implemented by dropping selected bits (paths) from the routing infrastructure (akin to splitting a single hypercube into two subcubes). It is important that all peers within the same AS drop the same routing paths. A `pMerge` operation conversely merges two local overlays into a single overlay. This is affected by pairing up peers in the two local overlays, having them exchange local routing tables and merging the two with different routing prefixes (akin to merging two subcubes into a hypercube of one larger dimension). Efficient algorithms for `pSplit` and `pMerge` have been developed and their performance characterized within the Plethora core.

4.5 Plethora Routing Performance

To critically examine the performance of the Plethora routing core, we have developed a network simulator that implements the routing schemes of the global and local overlays, implements the algorithms for merging and splitting local overlays, and emulates the underlying network. The topology of the underlying network is generated using the Georgia Tech. transit-stub network topology model (GT-ITM) [**?**]. All experiments are performed on a 32-processor Origin 2000 with 16GB of memory, running IRIX64.

The underlying network topology we use in our experiment contains 10 transit domains, with an average of 10 nodes per transit domain. Each transit node has an average of 10 stub domains attached. There are a total of 1,000 stub domains, with an average of 10 nodes per stub domain. A LAN with 10 hosts is connected to each stub node, resulting in a total of 100,000 hosts. The total

Overlay Nodes	10,000
Network Nodes	110,100
Cache Size per Node	5MB
Distinct Objects	500,254
Mean File Size	5,144 bytes
Max Local Overlay Sizes	200; 300; 400; 500; 1,000; 2,000
Max Delay (D)	30ms; 40ms; 50ms; 100ms; 200ms
α	0.70; 0.75; 0.80; 0.85; 0.90

Table 1: Parameters of the simulation experiments.

number of nodes in the underlying network topology is, therefore, 110,100. Each stub domain is considered an autonomous systems. The link delays are selected randomly in the following way: the delay of an edge between two transit domains is in the interval [20-80]ms; the delay of a transit-transit edge in the same transit domain is in the interval [3-23]ms; the delay of a transit-stub edge is in the interval [2-7]ms; the delay of a stub-stub edge is in the interval [1-4]ms; and the delay of an edge connecting a host to a stub node is fixed to 1ms.

The evaluation of a caching scheme requires appropriate dimensioning of the storage available for caching at each node, and a realistic workload. Since there are no publicly available traces that contain file sizes for existing peer-to-peer systems, we use web proxy logs for the distribution of file sizes in the network. The same approach was used to validate PAST in [29]. We use a set of web proxy logs from NLANR[1] corresponding to eight consecutive days in February 2003. The trace contains references to 500,258 unique URLs, with mean file size 5,144 bytes, median file size 1,663 bytes, largest file size 15,002,466 bytes, and smallest file size 17 bytes. The total storage requirement of the files in the trace is 2.4GBytes.

The main performance measuraments that we investigate are the performance gains in response delay and number of packets in the underlying network for queries in the two-level overlay compared with a single Pastry overlay. The performance gain is defined as: $g = \frac{m_1 - m_2}{m_1}$, where m_1 is the measurement (average delay or lookup messages) in the network without cache, and m_2 is the measurement in the network with cache. The source nodes of the queries are chosen randomly and uniformly, and the objects are accessed according to a Zipf-like distribution, with the ranks of the objects being determined by their position in the original NLANR trace. For the global overlay the Pastry parameters are: $b = 4$, and leaf set size $l = 32$. These parameters are also used in the single Pastry overlay. We measure the impact of the cache hit ratio, the maximum delay used to construct

[1] http://www.ircache.nlanr.net/

$\alpha = 0.70$					
Size \ Delay	30ms	40ms	50ms	100ms	200ms
200	51.70%	51.38%	51.24%	52.98%	56.64%
300	57.04%	58.40%	60.50%	61.36%	64.33%
400	59.31%	66.03%	66.17%	69.29%	68.88%
500	60.75%	68.26%	68.28%	72.65%	78.17%
1000	60.75%	72.15%	75.64%	86.03%	89.22%
2000	60.75%	72.15%	77.72%	90.01%	94.90%

$\alpha = 0.90$					
Size \ Delay	30ms	40ms	50ms	100ms	200ms
200	71.69%	71.55%	71.50%	72.53%	74.59%
300	74.72%	75.59%	76.82%	77.33%	78.89%
400	75.92%	79.65%	79.83%	81.59%	81.34%
500	76.68%	80.81%	80.94%	83.34%	86.20%
1000	76.68%	82.84%	84.78%	90.38%	92.18%
2000	76.68%	82.84%	86.00%	92.79%	95.88%

Figure 3: Cache hit ratios for $\alpha = 0.70$ and $\alpha = 0.90$.

local overlays D, and the maximum number of nodes in a local overlay in the performance gains. The cache hit ratio is the ratio between the number of queries responded in a local overlay by the total number of queries. The parameters of the simulation experiments are summarized in table 1.

The cache hit ratio is a function of the α value in the Zipf distribution and the maximum number of nodes in a local overlay. Figure 3 illustrates the cache hit ratios obtained with $\alpha = 0.70$ and $\alpha = 0.90$ and the different maximum local overlay sizes. The values in the tables correspond to the minimum and maximum ratios obtained for the parameters, other values of α produce cache hit ratios in those intervals. At these cache hig ratios, the reduction in end-user access latency exceeds 50%.

The number of local overlays is a function of the maximum number of nodes in a local overlay and the maximum delay allowed among ASs (D). Figure 4 show the number of local overlays when we vary these two parameters. The flat line after 500 when $D = 30ms$ is because the delay is limiting the number of nodes in the local overlays. For $D = 30ms$ the maximum number of nodes is never bigger than 481 nodes.

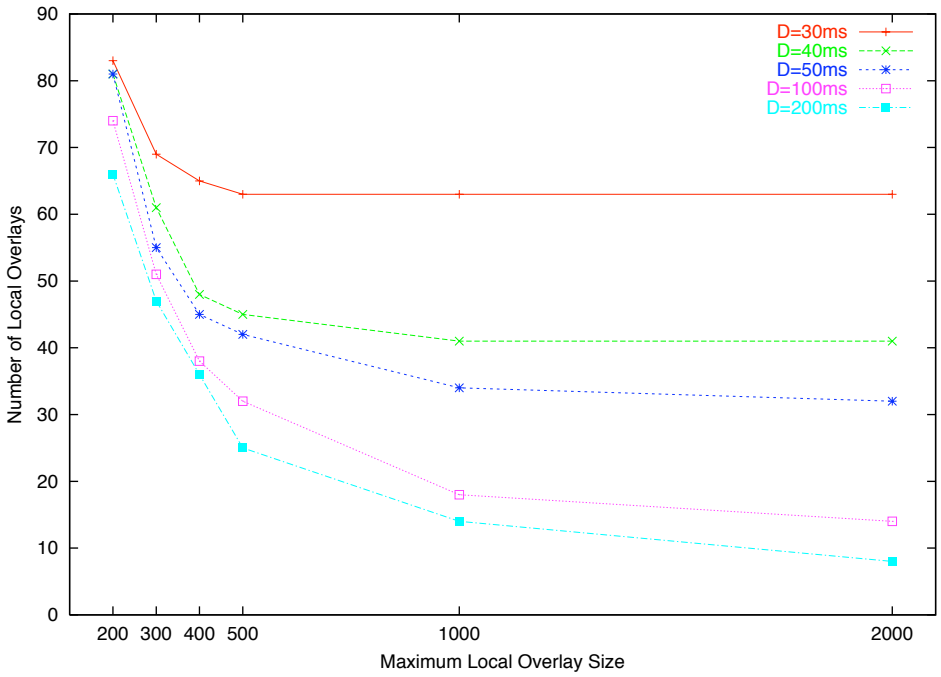


Figure 4: Number of local overlays.

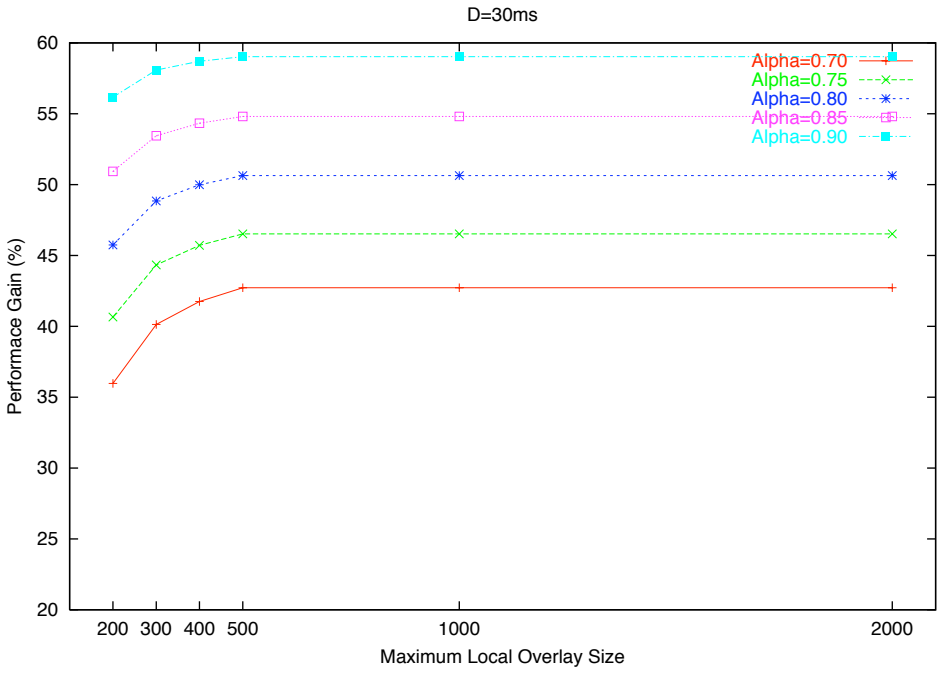


Figure 5: Performance gain in response delay for D equal to 30ms.

5 Concluding Remarks

This paper desrcibes the overall design of Plethora – a wide area read-write object repository. It puts Plethora in the context of related efforts, outlines its novel features, and describes the Plethora routing core in greater detail. Other critical aspects of Plethora, such as erasure coding, mutual exclusion, etc. are described in forthcoming manyscripts.

References

[1] The gnutella homepage. URL: www.gnutella.wego.com.

[2] The napster homepage. URL: www.napster.com.

[3] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In *5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

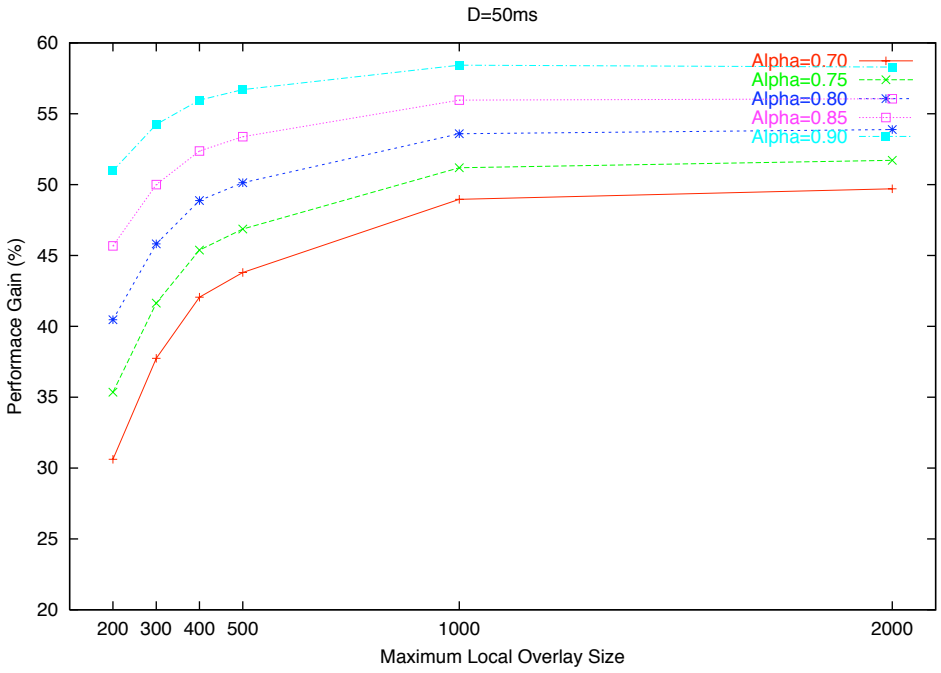


Figure 6: Performance gain in response delay for D equal to 50ms.

[4] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a Modern Processor: Where Does Time Go? In *Proc. of the 25th Intl. Conference on Very Large Databases*, pages 266–277, Sept. 1999.

[5] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless Network File Systems. In *Proceedings of the 15th Symposium on Operating System Principles. ACM*, pages 109–126, Copper Mountain Resort, Colorado, December 1995.

[6] J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Flexibility, Manageability, and Performance in a Grid Storage Appliance. In *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, July.

[7] P. Cao, E. Felten, and K. Li. Application-controlled file caching policies. In *USENIX Summer 1994 Technical Conference*, pages 171–182, June 1994.

[8] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting Network Proximity in Peer-to-Peer Overalay Networks, Technical Report MSR-TR-2002-82, 2002.

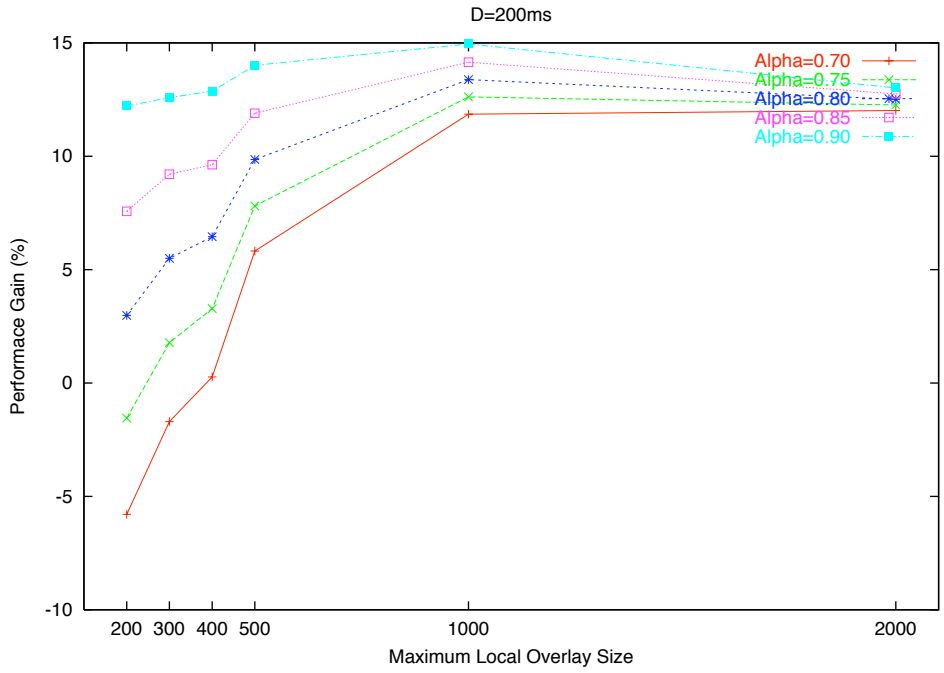


Figure 7: Performance gain in response delay for D equal to 200ms.

[9] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, February 1999.

[10] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-Area Cooperative Storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Lake Louise, Canada, October 2001.

[11] R. Ferreira, A. Grama, and S. Jagannathan. Building locality in peer-to-peer networks using autonomous systems. Technical report, Purdue University, 2003.

[12] R. Ferreira, A. Grama, and S. Jagannathan. An ip address based caching scheme for peer-to-peer networks. In *Proceedings of GLOBECOM*, 2003. (submitted).

[13] I. Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–25, 2001.

[14] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High-Performance Computing*, 11(2):115–128, 1997.

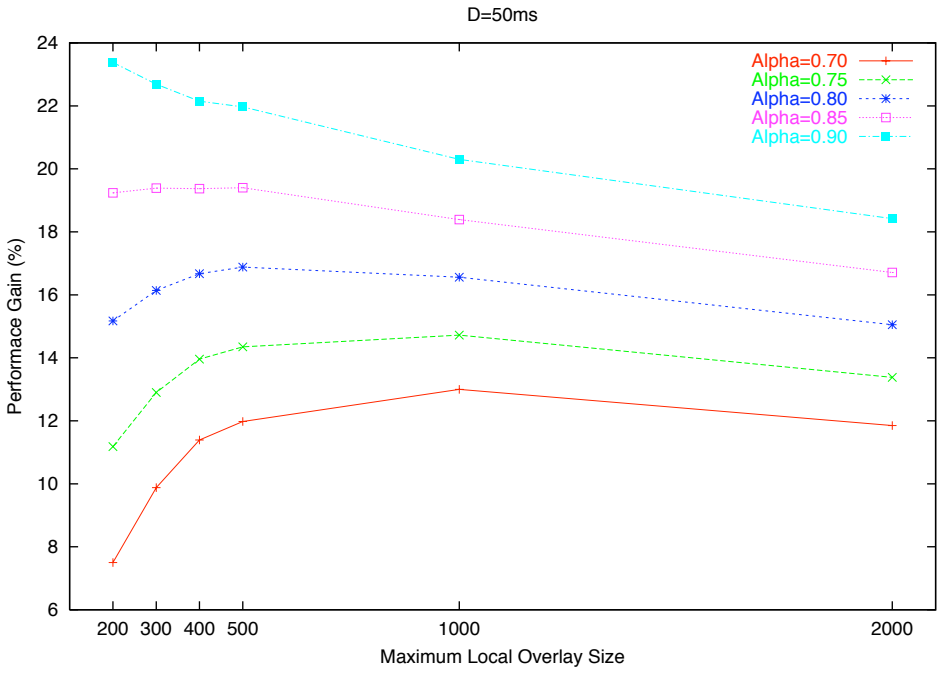


Figure 8: Performance gain in the number of packets in the underlying network D equal to 50ms.

[15] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan-Kaufmann, 1999.

[16] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM Transations on Computer Systems*, 6(1):51–81, Februrary 1988.

[17] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *13th Symposium on Operating Systems Principles*, Pacific Grove, CA, October 1991.

[18] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Cambridge, MA, November 2000.

[19] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *TODS*, 6(2):213–226, 1981.

[20] A. Muthitacharoen, R. Morris, T. Gil, and B. Chen. Ivy: A Read/Write Peer-to-Peer File System. In *5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[21] S. Quinlan and S. Dorward. Venti: A New Approach to Archival Storage. In *First USENIX conference on File and Storage Technologies*, Monterey,CA, July 2002.

[22] K. K. Ramakrishnan, P. Biswas, and R. Karedla. Analysis of file I/O traces in commercial computing environments. In *Proc. 1992 ACM SIGMETRICS and PERFORMANCE '92 Int'l. Conf. on Measurement and Modeling of Computer Systems*, page 78, Newport, Rhode Island, USA, June 1-5 1992. Published as Proc. 1992 ACM SIGMETRICS and PERFORMANCE '92 Int'l. Conf. on Measurement and Modeling of Computer Systems, volume 20, number 1.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[24] D. P. Reed. Implementing atomic actions on decentralized data. *ACM Transactions on Computer Systems (TOCS)*, 1(1):3–23, 1983.

[25] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The OceanStore Prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, San Antonio, TX, March 2003.

[26] S. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In *Proceedings of IEEE Infocom*, New Yourk, NY, June 2002.

[27] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[28] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 247–254, San Diego, CA, August 2001.

[29] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Lake Louise, Canada, October 2001.

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

[31] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming Aggressive Replication in the Pangaea Wide-Area File System. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[32] D. Santry, M. Feeley, N. Hutchinson, A. Veitch, R. Carton, and J. Ofir. Deciding When to Forget in the Elephant File System. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, Charleston, SC, December 1999.

[33] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160, San Diego, CA, August 2001.

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

[35] Sun Microsystems, Inc. NFS: Network File System Protocol Specification, RFC 1094, IETF, March 1989.

[36] W. F. Tichy. RCS - a system for version control. *Software - Practice and Experience*, 15(7):637–654, 1985.

[37] D. Zeinalipour-Yazti and T. Folias. Quantitative Analysis of the Gnutella Network Traffic, Technical Report, Department of Computer Science, University of California at Riverside, June 2002.

[38] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.