

A Survey on Peer-to-Peer Web Hosting

Fabian van der Werf



Delft University of Technology

A Survey on Peer-to-Peer Web Hosting

Research Task in Computer Science

Parallel and Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Fabian van der Werf

31st January 2007

Abstract

This report is an overview and analysis of current peer-to-peer web hosting systems and techniques. The systems are analyzed with respect to three different aspects. First, the content and storage mechanisms of the web hosting systems are explained. Most systems are either structured or unstructured networks, while there are a few systems with a hybrid network. Both types, structured and unstructured, can provide keyword search and replica management, for unstructured networks this is done automatically. Structured networks have to manage these features actively, but have the advantage of bounded lookups. Second, the issues of decentralization, scalability, and resilience are discussed. Generally, peer-homogeneous systems provide the best scalability and resilience, because these systems are self-scaling, and a node failure causes only the resources donated by that node to be no longer available. Finally, the goals and environments for which the systems are designed are discussed. Most systems have as a goal either server alleviation or latency reduction. Additionally, some systems provide facilities for untrusted environments, for dealing with high churn, for handling DHCP, or for firewalls.

Preface

This report describes my literature study, which is a part of my M.Sc. in Computer Science at Delft University of Technology. This study is an overview and comparison of peer-to-peer web hosting systems and is performed in the context of the I-Share project. The I-Share project performs research in the area of sharing resources in virtual communities for storage, communications, and processing of multimedia data.

I would like to thank my advisors dr. ir. D.H.J Epema and dr. ir. J.A. Pouwelse for their guidance and support.

Fabian van der Werf

Delft, The Netherlands
31st January 2007

Contents

Preface	v
1 Introduction	1
2 Overview of P2P Web Hosting Systems	3
3 Content Storage and Retrieval	5
3.1 Structured Networks	5
3.1.1 Squirrel	6
3.1.2 Kache	6
3.1.3 Backslash	7
3.1.4 Riptide	7
3.1.5 CoralCDN	9
3.1.6 OverCite	9
3.1.7 LOID	10
3.1.8 Evaluation	10
3.2 Unstructured Networks	10
3.2.1 PROOFS	11
3.2.2 BuddyWeb	11
3.2.3 Tuxedo	12
3.2.4 Internet Archive	12
3.2.5 Evaluation	13
3.3 Hybrid Networks	13
3.3.1 YouServ/YouSearch	13
3.3.2 OLP	14
3.3.3 Evaluation	15
3.4 Globule	15
3.5 Subnet Broadcast	15
3.6 Evaluation	16
4 Decentralization, Scalability and Resilience	17
4.1 Central Components	17
4.1.1 YouServ/YouSearch	17
4.1.2 OLP	18

4.2	Partially Decentralized	18
4.2.1	BuddyWeb	18
4.2.2	Riptide	19
4.2.3	CoralCDN	19
4.3	Fully Decentralized	20
4.3.1	Structured Networks	20
4.3.2	Unstructured Networks	20
4.3.3	Subnet Broadcast	21
4.3.4	Globule	21
4.4	Conclusion	21
5	Focus of Use	23
5.1	Goal	23
5.1.1	Server Alleviation	23
5.1.2	Latency Reduction	25
5.1.3	Archive Systems	26
5.2	Environments	26
5.2.1	High Churn Environments	27
5.2.2	Untrusted Environments	27
5.2.3	Firewalls and DHCP	28
5.3	Evaluation	29
6	Summary	31

Chapter 1

Introduction

Since the introduction of the world wide web, the architecture for web document delivery has not changed much. Most web content is still delivered through the traditional server-client architecture. Heavily loaded sites employ Content Distribution Networks (CDN) to increase performance. However, CDNs still rely on the server-client architecture, with replicated servers placed at tactically selected locations to distribute load among these servers.

In the past few years, peer-to-peer technologies have gained popularity in resource sharing. The main focus of peer-to-peer technologies has been on the distribution of large files, e.g., music and video. Obviously, peer-to-peer technology can also be used for smaller files such as web pages. This report is a survey on current peer-to-peer web hosting systems and techniques.

The systems and techniques discussed are analyzed with respect to three aspects. First, content storage and retrieval is explained. This aspect comprises how and where content is stored, and how other peers retrieve the content stored by the system. Second, the degree of decentralization is discussed. Ideally, all nodes in a peer-to-peer system are equivalent, which improves scalability and resilience. However, some peer-to-peer systems still rely on a central component. Finally, the focus of use of the systems is discussed. Peer-to-peer web hosting networks can have different goals such as latency reduction and server alleviation. This aspect also includes specific environments for which a system may be designed.

The rest of this report is organized as follows. Chapter 2 gives a brief overview of the systems and techniques that are discussed throughout the rest of this report. The following chapters discuss the three aspects mentioned above. Chapter 3 describes the storage and retrieval mechanisms, Chapter 4 focuses on the level of decentralization, and Chapter 5 discusses the focus of use. Finally, a summary of peer-to-peer web hosting is given.

Chapter 2

Overview of P2P Web Hosting Systems

In Table 2.1 an alphabetical overview is presented of the peer-to-peer web hosting systems that are discussed in this survey. The first column gives the names of these systems, which are used to refer to these systems in this report, the second column gives a short description for each system, and the third column contains references to the articles covering the systems in detail. The last entry in the table has no name, because it is not about a system or technique, but an analysis.

Name	Description	Ref.
Backslash	A system built on a DHT to handle flash crowds.	[22]
BuddyWeb	A collaborative caching system with mobile agents and dynamic reconfiguration.	[27]
CoralCDN	A world wide caching system that exploits locality.	[4]
Globule	A platform for self-replicating objects with a trust model for CDNs.	[15, 16]
Internet Archive	A concept of using the caches of nodes as an archive of the Internet.	[11]
Kache	A web caching system with a constant load on the nodes and the network.	[10]
LOID	A technique to store large objects in DHTs.	[8]
OLP	A caching system that tries to estimate the lifetime of objects located at peers to improve redirection.	[19]

Table 2.1: Overview of the peer-to-peer web hosting systems discussed in this report. (Continues on next page)

Name	Description	Ref.
OverCite	A peer-to-peer version of CiteSeer built on a DHT.	[25]
PROOFS	A peer-to-peer system for handling flash crowds with a randomized network.	[23]
Riptide	A secure caching system built on top of Tapestry.	[3]
Squirrel	A caching system built on top of Pastry.	[6]
Subnet Broadcast	An analysis of P2P caching design space, and a caching system using broadcasts.	[12]
Tuxedo	A caching system to improve latency.	[21]
YouServ/ YouSearch	A network of personal web servers with keyword searches and replicas.	[7, 1]
	An analysis on the scale of cooperative caching.	[28]

Table 2.1: Overview of the peer-to-peer web hosting systems discussed in this report. (Continued)

Chapter 3

Content Storage and Retrieval

The core of every peer-to-peer system that enables sharing of content are the storage and retrieval mechanisms. These mechanisms define what content is stored by what node and how a node can retrieve content that is stored by other nodes. This chapter presents the storage and retrieval methods used by the systems listed in Chapter 2.

Most peer-to-peer networks can be classified into one of two categories: structured networks and unstructured networks. Structured networks provide routing facilities and give guarantees on actually finding an object. Unstructured networks do not have such facilities, and finding an object involves querying neighbors. Finally, two hybrid systems, YouServ/YouSearch and Globule, are discussed which cannot be categorized in one of the two previously mentioned categories.

3.1 Structured Networks

This chapter discusses the peer-to-peer web hosting systems that have a structured network. Most of these systems are built on top of a *Distributed Hash Table* (DHT) [17, 24, 18, 29], which distributes the storage of key/value pairs across multiple nodes. Nodes and keys are mapped to a common namespace, and each node is responsible for storing the key/value pairs for keys that are located closely to the node in the namespace. The node storing a key/value pair is called the *home node* for that key/value pair. The usual approach for building a storage system is by using the object's name as a key and the object itself or a pointer to its location as corresponding value.

The following systems are based on DHTs and their storage and retrieval algorithms are described in the following sections: Squirrel, Kache, Backslash, Riptide, CoralCDN, OverCite and LOID

3.1.1 Squirrel

Squirrel [6] is a web cache sharing system that uses Pastry [18] to find object locations. Pastry is a DHT which uses a 128-bit namespace, objects and nodes are assigned an unique identifier in this namespace, and objects are mapped to the node whose identifier is closest the identifier of the object. Routing of messages is performed in a similar manner as CIDR; a message in transit is forwarded to a node such that the length of the matching prefix of the destination and the location of the message increases. The number of required hops for routing a message is bounded by $O(\log N)$, with N being the number of nodes in the system.

A user's browser is configured to use Squirrel as proxy server. Whenever Squirrel receives a request, Squirrel determines the home node according to Pastry. Squirrel has two schemes: home store scheme and directory scheme. With the home store scheme, the home node is responsible for having a cached copy of the objects that are mapped onto the home node. If the home node has not a cached copy, it acquires a fresh copy from the origin web server.

The directory scheme is somewhat more complicated. The home node keeps a directory of nodes for each object that is mapped onto the node. The directory contains pointers to nodes that have recently accessed the object, and therefore these nodes are likely to have a copy in their local cache. When an object is requested, the home selects randomly a node, whose copy is still fresh, from its directory. The home node updates its directory by adding the requesting node.

3.1.2 Kache

Kelips [5] is a DHT that allows increased efficiency and stability through increased memory usage and communication overhead. Kelips divides the peers into k so-called *affinity groups*, and objects are mapped onto an affinity group by hashing the object name.

Each peer maintains three tables: Affinity Group View, Contacts, and Resource Tuples. The Affinity Group View table is a set of other nodes located in the same affinity group. The Contacts table contains nodes located in other affinity groups, for each affinity group a small set of nodes is kept. Because a node has at least one contact for each affinity group, lookups are bounded by $O(1)$. For all entries in the Affinity Group View table and Contacts table, a round-trip time estimate is recorded. The Resource Tuples table stores filenames and the node storing the file, and only files that are mapped onto a node's affinity group are stored in this table.

Information dissemination occurs through gossiping: at a fixed interval, a node selects a small set of nodes to which information is multicast. The probability that a node is chosen for a gossip multicast decreases as the round-trip time for that node increases. Each entry in the tables is associated with a heartbeat. If the heartbeat of an entry is not updated for a fixed time-out period, the entry is deleted. Linga et al. introduce a web caching system [10] based on Kelips. For each entry in the Resources table, a directory is kept with hosts that have the object cached.

To retrieve an object, the node with smallest round-trip time is selected. When a node has successfully fetched an object, it adds an entry in the Resources table to indicate that it stores a copy of the object, and this entry is shared with other nodes through gossiping.

3.1.3 Backslash

Backslash [22] is a system that addresses flash crowds built on top of CAN [17]. However, it can be implemented using any other DHT. In the normal state, Backslash is not active and all nodes serve their own content. Whenever a web server is perceiving difficulties to handle incoming requests, it will divert subsequent requests through the use of URL-rewriting. If the server becomes even more loaded, all requests are redirected through DNS or HTTP redirection.

URL-rewriting consists of changing the URLs of embedded objects, such as images, to divert requests for these embedded objects to other nodes. Backslash implements two types of URL-rewriting: DNS-based rewrites and HTTP-based rewrites. With the DNS-based rewriting, each Backslash runs a simple DNS server, and prefixes the hostname of an embedded URL with the hash of the original URL. A request for an object is preceded by a DNS lookup for the hostname, the DNS server can divert requests to other Backslash nodes according to the CAN overlay network. The HTTP-based rewrite replaces the host of embedded URLs directly with the IP address of a Backslash node.

Periodically, each Backslash node injects its content into the DHT. This content is stored in the so-called replica storage and is guaranteed to be in place. Additionally to this replica storage, each node has temporary cache storage which is used to cache objects opportunistically in order to improve performance for subsequent requests for the same object.

3.1.4 Riptide

Riptide is a global caching system built on OceanStore [9], which is a global storage system.

OceanStore OceanStore is a global persistent storage system. In OceanStore, objects may always be cached anywhere. OceanStore has two methods for retrieving objects, a fast probabilistic routing method, and a slower reliable method.

The probabilistic routing is implemented using *attenuated* Bloom filters. An attenuated Bloom filter is an array of normal Bloom filters; the first Bloom filter is a summary of the objects that are stored by the local node; the i th Bloom filter is the union of all Bloom filters of the nodes that can be reached in i hops. A node keeps an attenuated Bloom filter for each neighbor. with these Bloom filters a node is able to select the outgoing edge with the maximum probability of encountering the requested object.

The slower reliable method uses Tapestry [29]. Tapestry assigns objects and nodes IDs from a common namespace independent of their location. A message is routed incrementally to its destination by increasing the matching suffix of the destination and the next location of the message. Tapestry also provides fault-tolerance for routing and location. Fault-tolerant routing is provided through redundancy of the routing table. For each entry in this routing table, two backup values are maintained in addition to the primary value. Whenever a neighbor fails, the backup entries are used to deliver messages. Location fault-tolerance is provided by assigning multiple home nodes for each object, the multiple home nodes of an object are identified by adding so-called salt values to the object ID.

Updating objects is difficult, because there may exist multiple replicas of the object. To ensure all replicas of the object are consistently updated, OceanStore organizes the replicas in a multicast tree. This multicast tree is self-maintaining, in that when a new replica is created, the routing algorithms discussed above are used to find other replicas. Updates executed on a replica are forwarded to all other replicas using the dissemination tree.

Riptide The Riptide system consists of three types of agents. The first type of agent is a browser proxy, which resides on the machine of the user, and fetches web content on behalf of the browser. If the browser proxy regards a requested object as cacheable, it submits a request to the OceanStore system. If the object is found, it is passed on to the browser, otherwise, the object is requested directly from the origin server.

The second type of agent is a gateway, which is a service that is responsible for inserting new content and updating stale cache copies. Gateways retrieve their content directly from the origin servers. If an organization wants its web site to be available through Riptide, it may deploy a gateway that pushes the web site content into OceanStore.

The third type of agent is a cache manager, which managers control the number of cache replicas and their locations. Browser proxies may give hints to a cache manager to improve the performance of the system. For example, a browser proxy may contact a cache manager to indicate that it was unable to fetch an object, the cache manager may then request a gateway to fetch the object and insert it into the underlying OceanStore. Note that, although users do not inject data, they can still contribute storage space as long as they participate in the underlying OceanStore system.

Riptide also uses OceanStore to provide push caching. Cached copies are tied together by OceanStore with a dissemination tree. When a cached object becomes stale, it may be updated, and through the dissemination tree all replicas of the object are also updated. With push caching the cached objects are always fresh.

3.1.5 CoralCDN

oralCDN is a network of HTTP proxies and DNS servers providing caching. The usage of CoralCDN is very simple; by appending `nyud.net:8090` to the host-name of an URL, the request is relayed to CoralCDN (e.g., `http://www.cnn.com.nyud.net:8090`). CoralCDN uses a hierarchical indexing system, Coral, which is explained next.

Coral Coral organizes peers in a hierarchical organization of clusters, and the actual implementation uses a three-level hierarchy, each peer is a member of one cluster at each level. A cluster is characterized by a *diameter*, the maximum desired round-trip time. Coral uses 20 ms, 60 ms and ∞ as diameters for respectively the lower level, the middle level and the upper level. This hierarchy enables to exploit locality of objects. When a node wants to locate an object, it starts searching for the object in the lower-level cluster. If the object cannot be found in the lower-level, the search continues in the middle-level cluster, and if necessary, the upper-level cluster is searched for the object.

Each cluster forms a distinct DHT. Routing in the clusters is based on Kademia [13]: at each hop the matching prefix of the node and the destination is increased. Key/value pairs may be stored at nodes which are not numerically closest to the key, due to the sloppy storage technique that Coral employs in favor of load balancing. When a node receives many requests to store key/value pairs for a certain key, it may deny store requests; another node which is also close to the key is then requested to store the key/value pair.

CoralCDN The network of CoralCDN consists of Coral DNS Servers and Coral HTTP Proxies. When a node looks up a web site which ends in `nyud.net:8090`, a Coral DNS Server receives a request to resolve the hostname, and returns a closely located Coral HTTP Proxy by using the hierarchical clustering network. Subsequently, the HTTP Proxy is requested for the URL, and if the proxy cannot find a cached copy in its local cache, it searches for a copy in the Coral network. If the URL is not available in Coral, the URL is requested from the origin server, and it inserts a new value in the Coral network indicating that the proxy has a copy of the URL.

3.1.6 OverCite

OverCite is a distributed version of CiteSeer, which is a digital library of computer and information science papers. CiteSeer crawls web sites for papers, and adds these papers to the library. OverCite uses a few tables to store documents and metadata, which are stored in a DHT. The DHT network is fully connected, which bounds lookups by one. Among the tables is a list of URLs that have to be crawled, and periodically, a node chooses randomly a URL from this list to crawl. If a node

encounters a document, it stores the document and metadata (e.g., citations) in the appropriate tables.

A found document is also inserted into an inverted index to allow efficient keyword lookups. The inverted index is divided into k partitions, and each node maintains a copy of one partition. With n nodes, there are n/k copies of each partition. A document is indexed in only one partition, but in all copies of that partition. If a node inserts a document in the inverted index, it also notifies the other nodes maintaining the same partition.

To serve web requests, each OverCite node also runs a web server, and requests from web clients are distributed over the nodes with a round-robin DNS server. A query is propagated to exactly one node in each partition to ensure completeness of the result. The web server collects the results from each node, and returns them to the client.

3.1.7 LOID

LOID [8] distributes the storage of large objects in DHT over multiple nodes. This technique works in general for DHT and not only for peer-to-peer web hosting systems. If the size of an object exceeds a fixed threshold, the file is divided into several smaller blocks and distributed over multiple nodes. The home node stores a header file with administration data such as URL name and pointers to the blocks. The home nodes for the blocks are determined by hashing the blocks.

The main reason why this technique is interesting for caching systems is because it has the potential of improving *byte* hit ratios. Traditionally, web cache replacement algorithms try to maximize hit ratios, and therefore purge large files to free space for many small files, because this yields higher hit ratios. However, in terms of byte hit ratios, a single hit for a large file of for example 10 megabytes is equivalent to 1000 hits for smaller files of 10 kilobytes.

3.1.8 Evaluation

The peer-to-peer system of Backslash is only used if a node is having difficulties to serve its requests, which are then diverted to other nodes according to the DHT. Squirrel and Kache are both thin layers on top of a simple DHT without extra features, CoralCDN and Riptide are more complex with replicas and locality. It is possible to provide keywords searching in DHT-based systems, as OverCite shows; however, this comes at the cost of maintaining an inverted index.

3.2 Unstructured Networks

Unstructured networks do not have fixed object location and routing mechanisms like DHTs. Each node in the network can have an arbitrary number of neighbors, although most implementations limit the number of neighbors. When a node decides to retrieve an object it will query its neighbors for the object, who in turn

will query their neighbors for the object, and so on. This continues until the object has been found or the maximum number of propagations has been reached. This search method is known as *blind search* [26]. The unstructured systems discussed in this report are PROOFS, BuddyWeb, Tuxedo, and Internet Archive.

3.2.1 PROOFS

PROOFS [23] consists of two protocols. The first protocol creates and maintains the network, and consists of peers exchanging a part of their neighbor set. The second protocol operates on top of the first protocol and facilitates object retrieval. Each peer has a set of neighbors, which is upper bounded by C . Peers continuously exchange a subset of their neighbor set, which is called a *shuffle* operation. A shuffle is performed as follows. First, a peer selects a random subset from its neighbor set. From this subset, the peer chooses a neighbor P with which it wants to do a shuffle. Then P is removed from the subset and the subset is sent to P . Peer P can either accept or reject the shuffle operation. If P accepts the subset, it will create an equally sized subset from its own neighbor set and sends it back to the initiating peer.

To retrieve an object, a peer creates a query and sends it to (a subset of) its neighbors. Each query carries an object description, a TTL value, a fanout value and a return address. The TTL value indicates how often the query should be forwarded as long as the object is not found. The fanout value indicates the number of neighbors a query should be forwarded to. When a peer receives a query, it first checks if it has a copy of the object stored locally. If the object is found, it is sent to the return address. If it is not found, the TTL value of the query is decremented. If the TTL value is below zero, the query is dropped. Otherwise, the query is forwarded to f neighbors, with f the fanout value.

3.2.2 BuddyWeb

BuddyWeb [27] is a peer-to-peer web cache sharing system that is built on top of BestPeer [14].

BestPeer BestPeer is a general peer-to-peer platform consisting of many peers and fewer LIGLO servers. If a peer wishes to enter the network, it has to register with a LIGLO server. Unlike most other networks, a registration does not end when the peer disconnects. When a peer re-enters the network, it has to contact the same LIGLO server with which it is registered.

BestPeer uses mobile agents for retrieving objects. For each query, a mobile agent is created and clones are sent to all neighbors, where they digest a peer's content, and report back. The process of cloning of agents continues until the pre-determined TTL value drops to zero.

In order to improve performance, BestPeer provides dynamic reconfiguration. A peer should keep the nodes from which it benefits most in its neighbor set. When a

peer initiates a request and receives matching results from peers that are not in its neighbor set, it may decide to add these peers to its neighbor set.

BuddyWeb BuddyWeb acts as a local proxy for the web browser. Whenever the browser submits a URL, BuddyWeb will create a query that is sent in the underlying BestPeer network. The result is a set of peers holding a copy of the object. The local proxy contacts one of these peers and requests the object, which is passed on to the browser.

In addition to the dynamic reconfiguration provided by BestPeer, BuddyWeb also facilitates similarity-based reconfiguration. Every peer's interest is stored by the peer's LIGLO server as a word list. Periodically, the LIGLO servers compute vector spaces for each peer that represent its word list. When a peer comes online and logs onto its LIGLO server, it also receives the vector spaces of all other peers. The peer computes similarities with other peers, and peers with a higher similarity are ranked higher.

3.2.3 Tuxedo

Tuxedo [21] is a peer-to-peer caching system using the CONCA [20] proxy cache architecture. Besides simple caching, CONCA provides transcoded cached objects, e.g., translated documents, resized images, etc.

Each Tuxedo node maintains two tables, a neighbor table and a server table. For each entry in both tables, contact information as well as latency and bandwidth is stored. When a node decides to retrieve an object, it only queries its neighbors that are able to provide performance improvement. Most web documents are relatively small and neighbors are selected by comparing their latency with the origin server latency. For large objects such as video clips, the selection of nodes is based on bandwidth instead of the latency. Searches are not forwarded to other neighbors, this is equivalent to a blind search with a TTL value set to one.

Because only direct neighbors are queried for results, maintaining the neighbor table becomes important. A Tuxedo node should keep the nodes from which it can benefit close. Furthermore, it is important that the bandwidth and latency values are representative. Each time a peer requests a neighbor, the neighbor sends a neighbor table along with the response. The receiving node inserts the new information in its own table and uses preliminary values for latency and bandwidth. Periodically, a node probes its neighbors to update the neighbor table with fresh values.

3.2.4 Internet Archive

Mantratzis and Orgun propose to use the aggregate caches of all nodes as a distributed Internet archive [11]. However, this is more a concept than an actual implementation. Objects with a short lifetime remain available through this archive. Whenever an object is no longer available on a website, it may still be retriev-

able from the caches of other nodes. The system would use a simple unstructured network and blind search for retrieving objects similar to Gnutella.

3.2.5 Evaluation

Tuxedo, Internet Archive, PROOFS and BuddyWeb simply use an unstructured network with blind search to retrieve objects. Tuxedo limits the time spent searching an object by setting the TTL to one. PROOFS makes its networks fault-tolerant by continuously letting nodes exchange their neighbor set. BuddyWeb uses for its blind search not a simple keyword query but mobile agents, these mobile agents provide extensibility. Creating a new type of search is done by creating a new kind of agent; other nodes do not have to be notified of this new type of agent.

3.3 Hybrid Networks

Hybrid networks use a central component to find which peers have the requested object. The discussed hybrid systems are YouServ/YouSearch and OLP.

3.3.1 YouServ/YouSearch

YouServ [7] is a collection of personal web servers tied together by a central component. YouSearch complements YouServ by enabling keyword search of the content published by the YouServ system.

YouServ YouServ is a system that allows users to publish content using their own machine as a web server. The YouServ system consists of four components: Browsers, Peers, a Dynamic DNS (DDNS) server, and a YouServ coordinator. Users who wish to access content served by YouServ do not need any modification, because a standard browser suffices. Peers publish content through a lightweight web server that runs on the peer's machine. The DDNS server and YouServ coordinator provide facilities to achieve improved availability and accessibility of content that is published by peers.

Each YouServ peer has its own subdomain name through which the published content is accessible with a standard browser. Whenever a peer comes online, it contacts the YouServ coordinator to indicate that it is online. The YouServ coordinator updates the peer's IP registered at the DDNS server such that the peer is accessible by its domain name.

Content may be unavailable for two reasons. First, a peer may be offline, and therefore its content is not available. YouServ provides facilities to overcome this. Two peers can make agreements to serve each other's content when one of the peers is offline. Whenever a peer goes offline, the coordinator checks whether the other peer is online and is willing to serve a replica. If the other peer accepts to serve a replica, the coordinator updates the DDNS server such that requests are redirected to the replica server.

Second, a peer may be unable to accept incoming connections, due to a firewall, a proxy, or NAT, and consequently the content is not accessible. YouServ deals with this by making each peer act as a proxy for other peers who are unable to accept incoming connections. When a peer registers with the coordinator, the coordinator checks whether it can connect with the peer. If the peer does not receive a connection, it requests the coordinator for a proxy peer. The peer receives contact information for the proxy peer and establishes a connection. The coordinator sets the DDNS record for the peer to point to the proxy peer. Upon an incoming request, the proxy checks at which host the request is directed, and if necessary, the request is forwarded.

YouSearch Periodically, a peer runs an indexing process to extract keywords from the published content. The process starts by scanning for new files and files that have been modified since the last scan. The process keeps an inverted index, and new and changed files are added to the inverted index. After all files are scanned and indexed, the process creates a Bloom filter [2] for all the keywords for which the node has an object. The Bloom filter is uploaded to the YouServ coordinator, and the coordinator aggregates the Bloom filters of all peers into a structure that maps bit positions in a Bloom filter to a set of peers.

When the coordinator receives a search query, it returns the peers that are likely (false positives may occur due to Bloom filters) to have content matching the keyword. The peer then contacts all peers in the result set and requests for objects concerning the keywords. A requested peer can easily lookup objects concerning keywords using the inverted index.

3.3.2 OLP

OLP [19] is a peer-to-peer web hosting system. The origin web server of a web site maintains a redirection directory for each object, which contains pointers to other nodes who have downloaded the object. When a client requests an object, the server returns several peers from the redirection directory. The client first tries to retrieve the object from peers returned by the server. If this fails, the object is directly served by the web server.

When a server has multiple peers for the same object, a selection of peers is needed. OLP tries to predict the object's lifetime at each peer. The lifetime in a peer-to-peer network depends on the peer's cache replacement policy and the peer's participation behavior. The minimum of the average active period and the average object replacement time is used as prediction of an object's lifetime at a certain peer. If the time when an object was cached and the time the peer joined the network are known, then a prediction of the end of the object's lifetime can be calculated. The web server bases its neighbor selection process on this prediction.

3.3.3 Evaluation

Both YouServ/YouSearch and OLP show similarity in the way requests are handled. A central component receives incoming requests and diverts it to the specific peer which stores the actual content. OLP and YouServ/YouSearch differ in which peer serves what the content; in YouServ/YouSearch the same peers always serve the same content, while in OLP any peer that recently requested the object may serve the request. Additionally, YouServ/YouSearch provides keyword search through a central inverted index.

3.4 Globule

Globule [15] is a platform that provides replication of web documents in a network of web servers. Significant performance improvements can be achieved if each served object has its own replication strategy compared to the traditional one-for-all approach. Globule takes the former approach and considers the replication policy as a part of the object. An object determines its best replication strategy by using trace-based simulations. Globule also supports replication of dynamically generated objects. This requires that the code as well as the data that is needed by the code, e.g., a database, is replicated.

When an object notices that a replica at a remote location would improve performance, the server starts by identifying suitable hosting servers that are able and willing to serve a replica. When placing a replica, the cooperating servers make agreements on resource consumption like storage and bandwidth. The master server of an object is responsible for redirecting requests to a suitable replica, which it does this through DNS redirection.

3.5 Subnet Broadcast

According to Mao et al., superior performance in cooperative web caching can be achieved by maintaining a shared cache among peers that are located geographically close [12]. This method is based on the assumption that nodes that are located geographically close have similar interests, i.e., a subnet. Querying other nodes can be implemented efficiently by using broadcast facilities of the underlying network. Instead of using the traditional URL-based caching, Mao et al. employ *content*-based caching. With content-based caching, a document is divided into several components, each component has its own cacheability. In traditional caching a document is entirely marked as either cacheable or uncacheable, while in content-based caching some components may be cached. This also allows dynamic web pages to be cached partially.

3.6 Evaluation

Due to the pull model of unstructured networks, replicas are created automatically. For structured systems replicas have to be managed actively; Riptide and CoralCDN perform such active replica management. Riptide and CoralCDN use these replicas to exploit locality by trying to return the closest replica, upon object requests.

Structured systems map the object name to its location without considering the content; therefore, keyword searches in structured networks require the additional overhead of maintaining an inverted index, like OverCite does. Not all unstructured systems provide keyword searching. However, keyword search in unstructured networks is straightforward: instead of just comparing an incoming query with the object's name, a node compares the query with the object's metadata such as keywords. Like structured systems, hybrid systems also require an additional inverted index to provide keyword search.

Chapter 4

Decentralization, Scalability and Resilience

Decentralization is important in peer-to-peer systems because it greatly influences scalability and resilience. For example, central components limit scalability and resilience. Three levels of decentralization can be identified. This chapter categorizes the systems listed in Chapter 2 based on their level of decentralization. The scalability and resilience of the systems are also discussed.

The first level consists of the networks relying on one or more central components providing vital functionality. Such central components may perform indexing or routing. Systems in the second level are partially decentralized. These systems do not have any central components, but not all peers fulfil the same role. Finally, the last level consists of systems with homogeneous peers, i.e., all peers are equal.

4.1 Central Components

OLP and YouServ/YouSearch rely on central components in the network. Obviously, a central component limits the scalability significantly. A growth of users in the network must be matched by replacing the central component with a more powerful one. Furthermore, if the central component is vital to the system, it is a single point of failure.

Some systems discussed in this report are peer-to-peer cache sharing systems [6, 21, 27, 3]. Although a cache sharing system may be fully decentralized, when content is not available in any peer's cache these systems rely on the origin web server. Most web servers use the traditional server/client model, and therefore reintroducing a single point of failure.

4.1.1 YouServ/YouSearch

The YouServ/YouSearch consists of two central components: YouServ Coordinator and the DDNS server. Both components form a single point of failure. When the

YouServ coordinator fails, the system becomes static in node participation. Node joins and proper node leaves are no longer possible. Node joins involves contacting the coordinator which contacts the DDNS server, thus the node wanting to join is not reachable by their hostname. When a node leaves, the YouServ coordinator normally tries to activate a replica. Thus also the replica functionality fails whenever the coordinator fails. Furthermore, the coordinator handles keywords searches. The other central component is the DDNS server. The DDNS server is responsible for translating YouServ hostnames into IP addresses. When the DDNS server fails, hostnames cannot be translated into IP addresses. Consequently, content served by YouServ nodes is only accessible through IP addresses. It is unlikely that users know IP addresses of YouServ nodes, and content is practically inaccessible. Without the DDNS server the YouServ/YouSearch system becomes useless. This makes the DDNS the most vulnerable single point of failure of YouServ/YouSearch.

4.1.2 OLP

In OLP the web server redirects requests to peers for which it estimates that they are live and have a cached copy of the requested object. Structured and unstructured systems do not rely on a central component to route requests, and nodes in these system send their requests to their neighbors. OLP is compared to these systems not very resilient, because whenever the web server fails, its content is no longer accessible in spite of live nodes with cached copies. As with the traditional server/client model, the server is still a single point of failure. Scaling OLP requires replacing the web server with a more powerful one, to be able to divert the request load to other peers. Furthermore, note that the central component needs sufficient storage to store a copy of all the objects in the system.

4.2 Partially Decentralized

This section focuses on partially decentralized systems. These systems do not have a central component, therefore these systems do not have a single point of failure, and are thus more resilient. Scaling such systems can be done in two ways. First, replace current components with more powerful ones, similar to scaling of centralized systems, and second, add more of the same components. Tasks are distributed among components, consequently, adding more components decreases the load on each component. The systems in this category are BuddyWeb, Riptide, and CoralCDN, their resilience and scalability are discussed in this section.

4.2.1 BuddyWeb

The underlying network of BuddyWeb, BestPeer, assigns a BestPeer Identifier (BPID) to each node. These BPIDs are assigned by Location Independent Global Names Lookup (LIGLO) servers, and provide translation from BPIDs to IP addresses. When a node comes online, it contact its LIGLO server which will update

the IP address for the BPID. A BPID is a pair that consists of the IP address of the peer's LIGLO server and a unique node ID assigned by the LIGLO. Because a BPID also includes the IP address of the LIGLO server, the node ID assigned by the LIGLO server has to be unique only within the LIGLO server and not among all LIGLO servers. However, this imposes that a peer always connects to the same LIGLO server.

Distributed LIGLO servers enhance resilience, because whenever a LIGLO server fails, only the nodes registered by that server are no longer accessible, other nodes are still available. However, peers cannot switch between LIGLO servers, therefore from the peer's point of view, the LIGLO server with which it is registered is a single point of failure.

Increasing the number of LIGLO servers causes a decrease of the performance of the similarity-based reconfiguration algorithm. Each peer announces its interests to its LIGLO server as a word list, and LIGLO servers compute similarity vectors from these word lists. Peers receive from their LIGLO server the vectors of all other peers, and compute similarities with each other. Thus each server holds the vectors of all peers in the network, adding more LIGLO servers will not improve this situation. Thus the similarity-based reconfiguration limits scalability, LIGLO servers have to be equipped with more resources to hold all the peer interest vectors as the number of peers grows.

4.2.2 Riptide

As discussed in Chapter 3 the Riptide system consists of three types of agents: gateways, cache managers and browser proxies. The role of injecting information into the system is explicitly separated from the role of the users, who retrieve data from the system. Consequently, the system does not scale automatically as the number of users increases, because scaling requires increasing the number of gateways and cache managers. The underlying storage system, OceanStore, scales along if gateways, cache managers, and users also participate in OceanStore.

Riptide has no single point of failures. Gateways, cache managers, and users are spread throughout the system. The routing overlay routes messages to the nearest replica or service provider. This means that when a cache manager or gateways fails, messages are automatically routed to the next nearest cache manager or gateway. This is also true for any OceanStore node contributing storage.

4.2.3 CoralCDN

The CoralCDN system consists of two type of nodes: proxy caches and users. Users are not involved in the peer-to-peer network: users use the system but do not contribute any resources to it. An increase of the number of users requires adding proxy caches to the system.

Failure of a node causes that the cache of that node is no longer available, however the system continues to function properly. If an object is not available in CoralCDN

(e.g. due to node failure), it is fetched from the origin web server.

4.3 Fully Decentralized

This section discusses fully peer-homogeneous networks. Such networks have the potential of being self-scaling.

4.3.1 Structured Networks

Structured networks built on DHTs (see Section 3.1) that are fully decentralized scale automatically as new nodes join. New nodes are added to the peer-to-peer network and messages are automatically routed to these new nodes. When a node fails, the objects for which the node is responsible are automatically mapped onto other nodes. For the Squirrel system this is not a problem, because it is a caching system. Squirrel will fetch unavailable objects from the origin server. For other systems this may pose a problem, because the lost content cannot be served by other nodes if there are no replicas.

The Kelips DHT used in Kache is more robust against node failures, because all nodes in the same affinity group have a copy of the objects mapped on the group. Objects become unavailable, if all nodes in the same group fail simultaneously.

Lookups in Kelips are bounded by $O(1)$, thus scaling does not influence lookup time. However, the resources necessary to maintain neighbor tables increases as the number of nodes increases. This problem is even bigger for OverCite, because OverCite nodes keep a full routing table, consequently the routing table grows proportionally with the number of nodes.

For Backslash a failing node has a considerable impact. When a node fails, the content that it served is neither available via the node nor via nodes serving replicas, because the failing node is also responsible for redirecting requests to replica servers. Furthermore, the failing node also served replica objects for other nodes, consequently redirections performed by these other node to the failing node will fail.

4.3.2 Unstructured Networks

The unstructured networks that are fully decentralized are Tuxedo, PROOFS, and Internet Archive; these systems show similar scalability. The resources needed by each node are constant; the size neighbor tables are limited, and only objects interesting to the node are stored. A node failure causes only that the resources of that node are not available anymore, however replicas may exist on other nodes. All three systems are caching systems, therefore Section 4.1 also applies to these systems.

4.3.3 Subnet Broadcast

The subnet broadcasting system introduced by Mao et al. is fully peer-homogenic. However, because the system relies for its performance on facilities provided by the underlying network, it scales not very good. This system uses broadcasts to communicate with other peers, and this kind of communication is cheap within a subnet. However, broadcasting over the Internet requires the node to set up point-to-point connections to each node in the network, and to send the message over these all connections. Consequently, each node needs to maintain a table with contact information of every other node in the network. This table may become very large, if there are many nodes in the network.

4.3.4 Globule

Globule does not define the network and communication to finding other peers that are able and willing to share resources. Therefore, it is impossible to state anything about the scalability of such a network. However, the trust model that Globule employs can be discussed [16].

For scalability, it suffices to know two properties of the trust evaluation algorithm, the trust model is covered in Section 5.2.2. First, calculating trusts requires all-to-all broadcast. Second, in advance, it is unclear which recommendations influence which trust values.

In a network with point-to-point communication and n nodes (e.g., the Internet), an all-to-all broadcast requires $n(n - 1)$ messages. The amount of traffic generated by the trust model grows quadratically with the number of nodes. With regard to communication, Globule's trust model does not scale well.

The second property forces a peer to calculate trust values for peers in which it may not be interested, because the order of evaluation of peer trust values is fixed by the algorithm. In the worst case, trust values for all peers have to be calculated. As the number of nodes increases, computing trust values becomes more computationally intensive. Furthermore, if a recommendation value changes, then trust values for all peers have to be recalculated.

4.4 Conclusion

Not surprisingly, fully decentralized scale best and are most resilient. The hybrid systems, YouServ/YouSearch and OLP, employ central components, which are single points of failure. In the discussed systems, the central components provide such important functionality, that failure results in total failure of the system. Furthermore, caching systems rely on the origin web server, whenever a requested object is not available in the network. However, these central components are not vital to the system.

Partially decentralized system provide the same resilience compared with fully decentralized networks. Partially decentralized networks do not scale as well as fully

decentralized systems, because scaling requires adding components to the network. Because BuddyWeb users are bound to a LIGLO server, a LIGLO server is for its users a single point of failure. CoralCDN and Riptide do not have such vulnerabilities and are therefore more resilient.

The structured systems that are fully decentralized are not all self-scaling. The routing tables maintained by Kelips and OverCite increase with the number of users, however Kelips and OverCite have good resilience due to the replicas. Subnet Broadcast has the same problem when the underlying network does not provide broadcasts. The structured systems that are self-scaling are Squirrel and Backslash. Fully decentralized unstructured systems are self-scaling and are also resilient.

Chapter 5

Focus of Use

Systems are designed with a certain goal and environment in mind, and this is also true for peer-to-peer web hosting systems. This chapter discusses what the systems try to accomplish, and whether they are successful. Furthermore, the environments for which the systems are designed and how the systems are adapted to these environments are considered.

5.1 Goal

The discussed systems are aimed at one of three goals. First, systems aimed at server alleviation serve web requests to take away load from the origin web server. Second, systems may aim at latency reduction through cooperative caching. Peer-to-peer caching systems aggregate the caches of all peers into a single cache storage. The peer-to-peer cache storage is much larger than the cache storage of a single peer, therefore peers can achieve higher hit ratios together. Finally, the aggregate of caches can be used as Internet archive. Objects that are no longer available from their origin server may still be available from the cache of a peer, furthermore, versioning of objects may be supported. These goals are discussed in this section.

5.1.1 Server Alleviation

Loads on server are often not constant and show peaks. To be able to handle these peaks, servers have to be overprovisioned. Overprovisioning is costly and inefficient, because most of the time the resources of the server are not used to their full extent. Peer-to-peer web hosting systems designed for server alleviation take away the need to overprovision servers by distributing load over nodes. Systems that are designed for server alleviation are PROOFS, CoralCDN, Backslash, YouServ/YouSearch, Globule, and OverCite. Ideally, load is distributed uniformly over all nodes, and this section discusses how well the previously mentioned systems achieve this.

PROOFS and CoralCDN fully take away the load from the server; if the document is available from the network, thus the origin server is not used at all. The overlay network of PROOFS randomizes continuously, therefore requests are distributed with a uniform probability over the participating nodes. CoralCDN and Globule exploit locality, and requests are routed to the closest cache with a copy of the requested object. With CoralCDN and Globule, flash crowds originating from a single geographic location can still put a heavy load on the system. Besides locality, this is also due to the fact that users are not part of the peer-to-peer network. Otherwise, the nodes at the geographical location would serve each other. PROOFS and CoralCDN provide good server alleviation by spreading the load evenly over the nodes.

Backslash does not perform as well as PROOFS and CoralCDN, due to the strict object-location mapping created by the DHT. Whenever a Backslash node is overloaded all, requests are redirected to another Backslash node. Backslash uses a DHT, and therefore, requests for the same object are always redirected to the same node. In the presence of a flash crowd, requests are redirected to the same Backslash node. The load is moved from one node to another, and the only improvement is that embedded objects are served by different nodes. The underlying DHT creates the strict mapping from objects to node, and is thus the root of this problem.

YouServ/YouSearch lets users host their own content with their own workstations, which removes the need for a central web server at which users put their documents. However, load is not distributed evenly across nodes. If a node serves a highly popular object, it will fail much faster at handling incoming requests than a traditional server, because a server has generally much more processing power than an ordinary workstation (i.e., the node). YouServ/YouSearch only works if the number of users is limited or additional replica management is performed. The current replica management of YouServ/YouSearch is not sufficient for this, because it only activates a replica when the original serving node intentionally goes offline, and it only activates one replica.

OverCite distributes web requests uniformly over nodes through a round-robin DNS. A query is on its turn distributed over nodes through the distributed inverted index. Of each partition there exists multiple copies (if there are enough nodes), therefore retrieval of an object is spread over subset of all nodes. As well serving web requests as handling queries is distributed over multiple nodes.

Evaluation PROOFS, CoralCDN, and OverCite distribute the load well over the nodes. Globule and CoralCDN are still susceptible to flash crowds originating from a single geographic location. Backslash performs poorly, because the same objects are always served by the same nodes, thus flash crowds are not handled well. YouServ/YouSearch has the same issue.

5.1.2 Latency Reduction

Caching is a common technique to reduce latency. The storage space reserved for caching greatly influences the effectiveness of caching, because more storage space obviously results in a higher hit ratio and less latency. By aggregating the cache storage of all peers, the total cache storage is much larger and thus a higher hit ratio is attainable.

Cost reduction is for most systems not a goal in itself, but it is an additional advantage of peer-to-peer web caches. In peer-to-peer caching systems a higher hit ratio can be achieved, therefore systems deployed in a corporate LAN can decrease the traffic to the Internet. Less traffic results in less costs.

Before discussing the latency reductive systems, we point out the conclusion of the analysis by Wolman et al. on cooperative caching [28]. Wolman et al. analyzed two traces on the potential of cooperative caching. They conclude that, cooperative caching improves hit ratios for smaller populations, and that for larger populations, the hit ratios do not improve significantly. Furthermore, they conclude that cooperative caching in a specialized environment (i.e. users have similar interests) does not perform better than cooperative caching in a general environment.

Systems The following latency reductive systems are discussed: Tuxedo, BuddyWeb, Squirrel, Kache, OLP, and Subnet Broadcast. Tuxedo strongly focuses on latency reduction; if a Tuxedo node estimates that a requested object cannot be delivered quicker by the peer-to-peer network than by the origin server, then the node fetches the object directly from the server, and does not bother to use the peer-to-peer network. A search for an object is not propagated to other neighbors, thus latency is completely determined by the latency of the connections to the direct neighbors and not the rest of the network. These strict rules causes that the benefit of the network is less compared to other systems, because a Tuxedo node will more often contact the origin server.

The maximum number of hops for Kache is two. For BuddyWeb, the number of hops necessary to find an object cannot be predicted. Therefore, BuddyWeb may be faster if the latency of the network is an order of magnitude smaller than the latency of the origin server. However, it is unclear how often a search is propagated, therefore no guarantees can be given on the latency. For Squirrel, the number of required hops is bounded by $O(\log n)$.

Systems that require multiple hops are only successful at reducing latency if they are deployed in a limited network, such that the latency of the network links are a magnitude smaller than the latency of the server. Otherwise, the server will be faster.

OLP will never achieve a lower latency than the web server. A request is first send to the web server and then redirected to another node. Consequently, the total latency is always greater than the latency of the web server.

The latency incurred by an subnet broadcast in a subnet is an order of magnitude smaller than the latency for a server on the Internet. Requests handled by subnet

broadcast have a low latency. However, as discussed in Section 4.3.3 this limits the scalability. For Internet level applications application multicast is necessary, and this does not have the low latency of an subnet broadcast. Subnet Broadcast is only successful within a subnet.

Evaluation Tuxedo is the only system that strictly controls the incurred latency and will ensure that latency is actually reduced. However, the latency of the links that connects peers is likely to be higher than the links that connects servers. In such a situation, the Tuxedo system is not much of a use, because most requests will be send directly to the origin server. This is also true for Kache and Subnet Broadcasting, because lookups in these systems require a single hop in the overlay network. OLP will never be able to improve latency. The systems are only useful when deployed in a LAN or subnet, otherwise the origin server will in most cases be faster.

Wolman et al. conclude that a large population does not improve performance the hit ratio significantly over a smaller population, therefore, peer-to-peer caches should be limited to a smaller network. Also taking into account that the discussed systems are unable to improve latency if deployed at a global scale, peer-to-peer caching systems perform best in a subnet or LAN.

5.1.3 Archive Systems

The web hosting systems that can be used as an archive are Riptide and Internet Archive. The versioning and archiving functionality of Riptide is provided by OceanStore. Internet Archive is actually not a working system, but it is a concept. It raises the issue of object versions, but it does not resolve this issue. Internet Archive uses a Gnutella-like network overlay. Questions like how does a peer know the version of an object and whether it is the latest version, are not answered.

5.2 Environments

Peer-to-peer networks can be deployed in many different environments, varying from a limited set of stable nodes in a LAN to millions of dynamic nodes spread across the Internet. Most peer-to-peer network are designed to operate at a global scale and are fault-tolerant in order to absorb node failures. This section focuses on systems designed for such particular environments.

Three different specific environments can be identified in the systems. First, networks with a high churn rate, second, environments in which nodes cannot trust each other, and finally, facilities may be provided for nodes that use DHCP or are connected to the Internet through a firewall.

5.2.1 High Churn Environments

This section discusses the systems that have specifically been designed for environments with a high churn rate. In such an environment peers often connect and disconnect, and sessions may be short. Kache, OLP, and Tuxedo are systems that are designed to operate in a high churn environment, and their mechanisms to handle churn are discussed in this section.

Kache Kache focuses on one specific consequence of churn, and that is the increased load on peers and the network that churn may cause. Churn may even block operations on the DHT, and thus cause denial of service of peer-to-peer systems. Kache prevents a denial of service by keeping the load on the hosts and the network constant.

Information coming from nodes joining and leaving is not passed immediately to other nodes, but is deferred to a later point in time. At a fixed interval, each node sends a limited amount of information to one neighbor. This rule imposes a limit on the amount of data a peer receives and sends, and thus also limits the load on peers and the network. See Section 3.1.2 for more details. However, the limitation of load does not come for free, because the cost is dissemination latency. It takes more time before information has reached each peer in the network, this latency depends on the algorithm that is used to select the peer to which information is sent. Dissemination latency within an affinity group is $O(\log n)$ under uniform target selection and $O(\log^2 n)$ under spatial target selection. Spatial target selection gives close nodes a higher probability of being selected compared to nodes that are more distant.

OLP OLP considers that nodes may have left the network, and that redirection to these nodes will be unsuccessful. A OLP server estimates whether a peer is still participating in the network, before possibly returning the peer as a candidate from which a cached copy can be fetched. For details on how OLP accomplishes this see Section 3.3.2.

Tuxedo In Tuxedo, a node maintains for each peer in its neighbor table a reputation value. When a node tries to contact a neighbor and the neighbor does not respond, the reputation value for that neighbor is decreased. If the neighbor is live and responds, then the reputation value is increased. Whenever the peer considers which neighbor to contact for an object, the reputation value is taken into account.

5.2.2 Untrusted Environments

One major issue in peer-to-peer is trust. In a global network peers often do not know each other, which raises the issue whether data received from another peer is correct and complete. Globule and Riptide are the only systems addressing this issue.

Globule One part of Globule is the agreements between cooperating hosts on storage space and the bandwidth that is donated by one host to the other. It is difficult, if not impossible, to verify if a host is acting according to its part of the agreement. Of course, it is easy to check whether a node is hosting a certain object or not, but verifying whether a node is giving the agreed bandwidth is not straightforward. Therefore, Globule introduces a trust model based on recommendations [16].

Each peer has recommendations, trust values, for a few other peers in the network. The trust values for the rest of the network are deduced using two rules. Rule 1 enforces trust decay; if node A has a trust of $T(A, B)$ in node B , and node B has a trust of $T(B, C)$ in node C , then the trust of node A in C through B , $T_B(A, C)$, is a function of $T(A, B)$ and $T(B, C)$ of which the result is less than $T(A, B)$ and $T(B, C)$.

Rule 2 aggregates parallel trust values. Consider node A that has a trust in D of $T_B(A, D)$ and $T_C(A, D)$ via respectively B and C calculated according rule 1. The aggregate trust, $T(A, D)$, is a value larger than both $T_B(A, D)$ and $T_C(A, D)$. The exact functions to compute the trust values can be found in [16]. Using these two rules, trust values for each peer in the network can be calculated. A peer may use these trust value to decide whether it will cooperate with another peer.

This trust model has two drawbacks. First, the trust model is static. Recommendations cannot be adjusted according to previous cooperations, because it is impossible to determine whether a cooperation was successful. This is due to the fact that it cannot be verified whether a node is respecting the agreement. Second, the trust model scales poorly, Section 4.3.4 elaborates upon the scalability of Globule's trust model.

Riptide Riptide is in two aspects designed for untrusted environments. First, the underlying storage system, OceanStore, is designed for an untrusted infrastructure. OceanStore assumes that nodes cannot be trusted with the contents of the files they store, therefore every file is encrypted. Reading a file is impossible because it is encrypted, and replacing a file's content requires a private key which the node does not have. Second, in Riptide, users are not trusted and cannot insert data in the caching system. Gateways are the only agents that can add objects to OceanStore, but this causes that the system is not very scalable, see Section 4.2.2.

5.2.3 Firewalls and DHCP

BuddyWeb provides facilities to handle peers using DHCP. In addition to DHCP handling, YouServ/YouSearch also provides facilities for peers that are unable to accept incoming connections. These are discussed in this section.

Firewalls Due to security reasons and the shortage of IP addresses, many nodes connect to the Internet through a NAT or a firewall. Consequently, many nodes are unable to accept incoming connections, and other nodes are unable to contact these

firewalled nodes to request resources. YouServ/YouSearch provides facilities for peers located behind a firewall. A firewalled peer can relay its incoming connections through another peer that is able to receive incoming connections. Each peer acts as a proxy server for up to four other peers. The firewalled peer establishes a connection with the proxy peer, and whenever the proxy peer receives a request for the firewalled peer, it forwards the request over that connection.

DHCP Handling DHCP is only necessary if a peer has to be identifiable across multiple sessions for which the IP address of the peer may be different. BuddyWeb and YouServ/YouSearch use an intermediate translation service to map peer IDs to IP addresses. YouServ/YouSearch uses Dynamic DNS for this purpose and BuddyWeb use their LIGLO servers. When a peer logs onto the network, the record in the DDNS server or LIGLO server is updated with the IP address of the peer.

5.3 Evaluation

Most system are focused on either server alleviation or latency reduction. Backslash and YouServ/YouSearch are still susceptible to flash crowds and do not distribute the load over the nodes. PROOFS, CoralCDN and OverCite are better at this. PROOFS and OverCite distribute the incoming requests randomly over the nodes. CoralCDN exploits locality, and may therefore fail if a flash crowds originates from a single geographical location.

Latency reductive systems will only be successful if the diameter of the network is limited. Tuxedo only queries direct neighbors to guarantee latency reduction, while in a LAN multiple hops can be done and still improve latency. Thus Tuxedo is not used to its fullest extent. BuddyWeb, Squirrel, and Kache always first try to fetch a requested object from the network and are thus using the network more effectively than Tuxedo. OLP will never be able to improve latency, because the origin web server, which performs the redirection, has to be contacted.

Kache, OLP, and Tuxedo are the systems that are focused on handling high churn rates. Kache focuses on the traffic that node joins and leaves generates, while OLP and Tuxedo focuses on the availability of a node for retrieving the object.

Globule and Riptide focus on untrusted environments. Globule use recommendations supplied by the user in combination with a inference mechanism to deduce trust values for all peers, however this system has its limitations. Riptide does not allow users to inject new information, because they cannot be trusted. Injecting new information is done by gateways. Finally, BuddyWeb and YouServ/YouSearch are able to handle DHCP through an intermediate translation system.

Chapter 6

Summary

The discussed peer-to-peer web hosting systems use one of three types of peer-to-peer networks: structured, unstructured, or hybrid networks. Most of the discussed web hosting systems have either a structured (i.e., DHT) or an unstructured network. The main differences between structured and unstructured networks are in the areas of keyword search, bounded lookups, and replica management. Unstructured systems naturally provide keyword searches through their architecture, while structured networks can provide keyword search, but at the cost of maintaining an inverted index. The advantage of structured networks is that object lookups are bounded and objects are always found, unless they are not available at all. In unstructured networks, such guarantees cannot be given. unstructured networks automatically create replicas due to their architecture. Some structured networks actively create replicas and exploit locality. However replica management does not come for free with structured networks. Replicas are only useful for static web pages, while dynamic pages cannot easily be replicated. Replicating dynamic pages requires replication of the code and other resources (e.g., databases) that are necessary for generating the dynamic page.

Some systems integrate with the standard browser by configuring the peer-to-peer network as proxy. The system can then evaluate whether to forward a request to the peer-to-peer system or not.

Hybrid systems can provide keyword search and replicas, but are less interesting due to central components. These central components causes hybrid systems to scale poorly and be less resilient. Fully decentralized systems are systems in which all peers are equal, in general, these systems scale best and are most resilient. In partially decentralized, systems there are no central components but not all peers are equal. These system are as resilient as fully decentralized systems, but are not self-scaling.

The discussed systems are aimed to accomplish one of the following goals: latency reduction, server alleviation, and archiving. A peer-to-peer cooperative caching application will only be successful if the diameter of the network is limited. The search for and retrieval of a document must take less time than the retrieval of the

document from the original server. To achieve this, peers have to be connected by low-latency and high-bandwidth links. Since caching systems have a limited size, scalability is less of an issue. Server alleviating systems try to handle flash crowds and to fulfil every request. These systems do this by distributing the load among peers. A simple DHT webhosting system is unable to accomplish this, because requests for certain documents are always mapped onto the same nodes. Unstructured systems automatically create multiple replicas of objects. In structured systems, replica management has to be done actively (e.g., CoralCDN and Riptide), while in unstructured systems replicas are created automatically due to the pull model of unstructured networks.

Bibliography

- [1] Mayank Bawa, Roberto J. Bayardo Jr., Sridhar Rajagopalan, and Eugene J. Shekita. Make it fresh, make it quick: searching a network of personal webservers. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, pages 577–586, New York, NY, USA, 2003. ACM Press.
- [2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- [3] Patrick R. Eaton. Caching the web with oceanstore. Technical Report UCB/CSD-02-1212, Berkeley, Berkeley, CA, USA, 2002.
- [4] Michael J. Freedman, Eric Freudenthal, and David Mazires. Democratizing Content Publication with Coral. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.
- [5] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. 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*, 2003.
- [6] Sitaram Iyer, Antony Rowstron, and Peter Druschel. Squirrel: a decentralized peer-to-peer web cache. In *PODC '02: Proceedings of the 21st Annual Symposium on Principles of Distributed Computing*, pages 213–222, New York, NY, USA, 2002. ACM Press.
- [7] Roberto J. Bayardo Jr., Rakesh Agrawal, Daniel Gruhl, and Amit Somani. Youserv: a web-hosting and content sharing tool for the masses. In *WWW '02: Proceedings of the 11th International Conference on World Wide Web*, pages 345–354, New York, NY, USA, 2002. ACM Press.
- [8] Kyungbaek Kim and Daeyeon Park. Efficient resource management for the p2p web caching. In *AICT-SAPIR-ELETE '05: Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop*, pages 382–387, Washington, DC, USA, 2005. IEEE Computer Society.
- [9] John Kubiawicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, New York, NY, USA, 2000. ACM Press.
- [10] Prakash Linga, Indranil Gupta, and Ken Birman. A churn-resistant peer-to-peer web caching system. In *SSRS '03: Proceedings of the 2003 ACM Workshop on Survivable and self-regenerative systems*, pages 1–10, New York, NY, USA, 2003. ACM Press.
- [11] Constantine Mantratzis and Mehmet Orgun. Towards a peer2peer world-wide-web for the broadband-enabled user community. In *NRBC '04: Proceedings of the 2004*

- ACM Workshop on Next-generation Residential Broadband Challenges*, pages 42–49, New York, NY, USA, 2004. ACM Press.
- [12] Yonggen Mao, Zhaoming Zhu, and Weisong Shi. Peer-to-peer web caching: Hype or reality? In *ICPADS '04: Proceedings of the 10th International Conference on Parallel and Distributed Systems*, pages 171–178, Washington, DC, USA, 2004. IEEE Computer Society.
 - [13] Petar Maymounkov and David Mazières. Kademia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the 1st International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.
 - [14] Wee Siong Ng, Beng Chin Ooi, and Kian-Lee Tan. Bestpeer: A self-configurable peer-to-peer system. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, page 272, Washington, DC, USA, 2002. IEEE Computer Society.
 - [15] Guillaume Pierre and Maarten van Steen. Globule: A platform for self-replicating web documents. In *PROMS 2001: Proceedings of the 6th International Conference on Protocols for Multimedia Systems*, pages 1–11, London, UK, 2001. Springer-Verlag.
 - [16] Guillaume Pierre and Maarten van Steen. A trust model for peer-to-peer content distribution networks. http://www.cs.vu.nl/~gpierre/publi/TMPTPCDN_draft.php, Nov. 2001.
 - [17] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-Addressable Network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.
 - [18] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.
 - [19] Young-Suk Ryu and Sung-Bong Yang. An Effective Peer-to-Peer Web Caching System under Dynamic Participation of Peers. *IEICE Transactions on Communications*, E88-B(4):1476–1483, 2005.
 - [20] W. Shi and V. Karamcheti. Conca: An architecture for consistent nomadic content access. In *Workshop on Cache, Coherence, and Consistency*, June 2001.
 - [21] W. Shi, K. Shah, Y. Mao, and V. Chaudhary. Tuxedo: A peer-to-peer caching system. In *PDPTA '03: Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications*, Washington, DC, USA, 2003. IEEE Computer Society.
 - [22] Tyron Stading, Petros Maniatis, and Mary Baker. Peer-to-peer caching schemes to address flash crowds. In *IPTPS '02: 1st International Peer To Peer Systems Workshop*, volume 2429, pages 203–213, Cambridge, MA, 2002. Springer Berlin / Heidelberg.
 - [23] Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A lightweight, robust p2p system to handle flash crowds. In *ICNP '02: Proceedings of the 10th IEEE International Conference on Network Protocols*, pages 226–235, Washington, DC, USA, 2002. IEEE Computer Society.
 - [24] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

- [25] Jeremy Stribling, Isaac G. Councill, Jinyang Li, M. Frans Kaashoek, David R. Karger, Robert Morris, and Scott Shenker. Overcite: A cooperative digital research library. In *International Workshop on Peer-to-Peer Systems (IPTPS '05)*, 2005.
- [26] Dimitrios Tsoumakos and Nick Roussopoulos. Analysis and comparison of p2p search methods. In *InfoScale '06: Proceedings of the 1st International Conference on Scalable Information Systems*, page 25, New York, NY, USA, 2006. ACM Press.
- [27] Xiaoyu Wang, Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. Buddyweb: A p2p-based collaborative web caching system. In *Revised Papers from the NETWORKING 2002 Workshops on Web Engineering and Peer-to-Peer Computing*, pages 247–251, London, UK, 2002. Springer-Verlag.
- [28] Alec Wolman, M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *SOSP '99: Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 16–31, New York, NY, USA, 1999. ACM Press.
- [29] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*. Technical Report UCB/CSD-01-1141, Berkeley, Berkeley, CA, USA, 2001.