Towards a Smart, Self-scaling Cooperative Web Cache

Tomáš Černý[1], Petr Praus[2], Slávka Jaroměřská[2],
Luboš Matl[1], and Michael J. Donahoo[2]

[1] Department of Computer Science and Engineering,
Czech Technical University, Charles Square 13, 121 35 Prague 2, CZ
{tomas.cerny,matllubo}@fel.cvut.cz
[2] Department of Computer Science,
Baylor University, P.O. Box 97356, 76798-7356 Waco, TX, USA
{petr_praus,slavka_jaromerska,jeff_donahoo}@baylor.edu

Abstract. The traditional client/server architecture for web service delivery fails to naturally scale. This results in growing costs to the service provider for powerful hardware or extensive use of Content Distribution Networks. A P2P overlay network provides inherent scalability with multiple benefits to both clients and servers. In this paper, we provide analysis, design and prototype implementation of Cooperative Web Cache, which allows us to scale web service delivery and cope with demand spikes by employing clients in content replication. To demonstrate performance capabilities, we provide a prototype emulation for both client and server.

1 Introduction

Expectations on web service delivery rapidly grow every year. What we experienced in the past in a standalone, rich user interface application can be seen today online and on demand. As this trend continues, we expect online version of many applications such as office tools, remote desktops, user-friendly presentations, etc. This direction requires a network that can scale naturally and can provide soon feedback on client gestures and navigation.

The contemporary model used for web services follows the client-server architecture. This model works well until we reach hardware or bandwidth bottlenecks. In order to provide better scalability we must invest in powerful server hardware, load balancing, CDN services [1], and/or application rewrite to allow Cloud [2]. CDN distributes resources to multiple places around the world and allows the end-user to use the closest host for communication. Cloud computing allows to scale the service quickly. As the demand grows, the service might be replicated to multiple computers and handle larger load. Unfortunately the cloud involves extra charges for such ability.

In this paper, we argue that there exists a more natural way to deal with growing interest in a service. From the CDN and the cloud approaches, we see three aspects that should be addressed. First, the service should be as close as

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 443–455, 2012.

possible to the client. Second, if the interest grows, there should exist multiple replicas of such service. Finally, the existing service should not experience any changes and should not be recompiled. One possible approach to address the above mentioned aspects is to involve decentralized model to communicate with services. Peer-to-peer (P2P) overlay network has qualities that may satisfy our needs. Such network can be built by involved clients communicating with each other. This way the client requesting a service may cache the service results (similar as CDN) and those results might be provided to others. Such an overlay network may grow large, in that case we want two clients to have the best conditions for their communication, such as the best bandwidth or the lowest latency. Two distinct communication types exists, such as the client-to-server and client-to-client. The network itself should be self-scalable and resilient to client departure. Similar to clouds the client provides hardware resources as contribution to others. Such mechanism does not require any changes neither to the service nor to the client. Our prototype Cooperative Web Cache (CWC) provides the above mentioned functionality and properties. It allows us to evaluate real-world case study and to identify advantages and disadvantages to both client and server sides, and to identify challenges and further research.

This paper is organized as follow: As next analysis and design details are provided (Section 2). Case study and evaluation is in Section 3. Related work is discussed in Section 4. Paper is concluded in Section 5.

2 Analysis and Design of the CWC

Client-server model puts the entire burden on the service provider, who needs to invest into new hardware or CDN. CWC relocates most of the burden to clients who reuse the service results and share it with others. As popularity of a service grows more clients are participating in serving and in result the service scales more naturally with the size of clients. Furthermore, the distribution of participating clients may form clusters, so that a client in a given cluster should communicate within the cluster. The aim of CWC is to decrease server load and to speedup service delivery. The original service should stay unchanged and the whole process should be transparent to both sides. It can be seen as HTTP web browser extension or a proxy.

The CWC should be based on peer collaboration and expect various peer capabilities. CWC overlay network must sustain clients arrivals and departures and must be self-organizable. When clouds of clients with certain service results exist the requesting client should be directed to the nearest client. The nearest client in the overlay represents a location with the highest communication bandwidth and the lowest latency. In order to select the serving client an anycast request should be sent to the network. Although anycast is not supported by all networks the application level anycast can be used.

Regards web applications the service is to provide a page that consists of various resources (html, css, js, etc.). These resources can be divided on static and dynamic. Static resources often distributed on CDN are browser cacheble

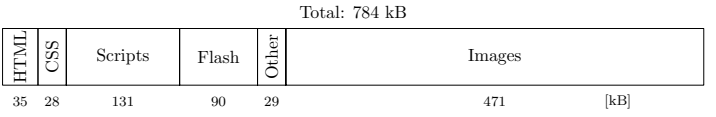

Fig. 1. Average web page content

and can be reused over the time (until changed by the host). Dynamic resources change often based on context, GET or POST request parameters [3] or user rights. Dynamic resources cannot be cached by browser and the replication would require a deeper knowledge of the system, although technologies such as AJAX [4] or AHAH (Asynchronous HTML and HTTP) can be seen as a successful approach with incremental changes to a temporarily cached dynamic resource. From the above, only static service results can be replicated. In order to impact the overall page load time by our approach the page must have static resources. Evaluation of Alexa Top 500 web sites [5] shows that 90% of web page resources are static in average case. This is in line with Google statistics as shown in Fig. 1.

The life-cycle of expected behavior shown in Fig. 2 is described from the client perspective. A client browser sends a request to a server (a1,b1,c1). The request is inspected to determine whether it provides a cacheable result (CWC). If not, the request is sent to the server (c2), otherwise the anycast request for the service result is sent to CWC cloud (a2,b2). If the result was not found (a3) the request is forwarded (a4) and fetched from the server (a5), otherwise is fetched from the cloud (b3). In both cases the client registers for the CWC group (a6) of clients having the result. In case the group does not exist then a new one is created (a6). The CWC group allows other clients to request the service result.

The life-cycle can be summarize as follows:

1. Client sends a request to a server
2. The request is inspected for the result
   (a) [Not cacheable] Request forwarded to the Server ⇒ step 3.
   (b) [Cacheable] Anycast for service result to CWC cloud
      i. [Result found] fetch from CWC Peer
      ii. [Result no found] fetch from Server
      iii. Join (create) CWC Group with service result
3. Done

The *basic* life-cycle above can be further extended with multiple strategies. A web page that consists multiple sources (service results) is most likely either cached in the network with all its sources or none of those will be registered in the network. We an approach based on this assumption *cache miss avoidance*. This way the CWC proxy can predict whether to send a request to the CWC cloud or directly to the server. As next, an index of expected service results can be built per a web page (as in [6]) and registered in the CWC cloud, this way the

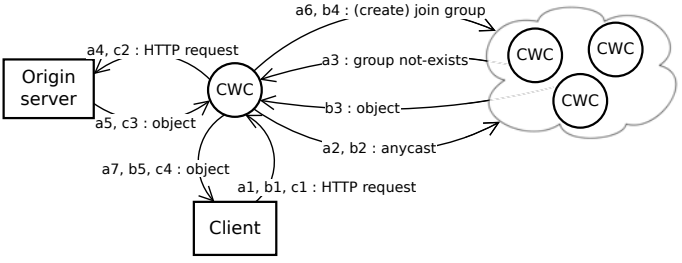

Fig. 2. Cooperative Web Cache request life-cycle

CWC proxy knows what services to call before the first service is requested and result parsed. This approach is referred as speculative *preloading*. Furthermore, a similar principle as in Bittorent [7] can be applied. The large service result can be divided into multiple chunks and requested from different locations in parallel, referred as *chunking*.

In order to verify validity of service result received from the CWC network multiple approaches can be applied. From the most optimistic approach a simple time-stamp will exist, once the result is stale based on the timeout the whole service result group is invalidated and must be rebuilt. The most pessimist approach is to always verify the result hash towards the server (as in HTTP), in case of failure the result group is invalidated. A reasonable approach is to employ probability where the client verifies the service result with probability p. In case of failure the group is invalidated and service requested from the server. The setting of p then reflects the level of optimism placed on the cache.

The CWC design [8] must be flexible towards modification and extension. For this reason it should be build on multiple layers to support maintenance and allow plugins for extension. The top layer should communicate with web browser as a proxy or web browser plugin. The bottom layer should contain a cache which can be either local or distributed, furthermore there can exist an addition extension or new services available on CWC. The middle layer should provide lookup services and mediate communication between the layers.

CWC prototype implementation consists from multiple parts. The distributed cache is build on P2P Framework developed at Rice University called Pastry [9]. This frameworks provides self-scalable P2P overlay network with operations like join, create or query. Its network is robust towards often joining and departing nodes and is tree-based towards its groups. A Tree allows to invalidated the whole group by contacting the root. Pastry itself does not provide neither anycast nor multicast capability necessary for optimal client-to-client communication. Pastry extension, Scribe [10] [11], provides both services in the application layer and guarantees that a message is delivered with logarithmic complexity regards the size of participants. The prototype is implemented in Java technology.

3 Case Study

In our case study we look at both client and server perspectives. In order to provide real-world study we must know capabilities of a server and a client that should be emulated. It is necessary to know the upload and download bandwidth capacity and latency of both.

From our measurement [5] for client-to-server, the median time for establishing a TCP connection (round-trip time) with the Alexa Top 500 websites through a backbone connection located in Prague, Czech Republic was 114 milliseconds. The latency for client-to-client is chosen from the above measurement to be 20 milliseconds. The presumption is that most traffic will be between geographically clustered nodes that are selected by anycast with distance metrics. End-user speeds vary significantly based on locations of such measurements or type of connectivity. The value for the U.S. household is based on Speed Matters report [12] and presented in Table 1. Average bandwidth across all providers in Prague and bandwidth average of state-wide provider UPC are taken from speed measuring website rychlost.cz on January 24th, 2011. Average bandwidth values for servers are much higher, commercial offers in data centers provide 100/100 Mbit/s for upload and download.

Our testbed environment [13] is based on a central unit directing distributed emulators via a test script. Each emulator node can be distributed on multiple machines where multiple nodes can be hosted by the same machine as well. Each emulated node has a specific network setting via latency and bandwidth [14].

3.1 Client Evaluation in Homogeneous Network

The evaluation in a homogeneous network applies identical client communication bandwidth and latency between all clients. The scenario is that a group of clients has the intention to download a web page. The page has 100 resources with consistence provided in Fig. 1. The first client requests and downloads the page, once done, a second client requests and downloads the page and so on. The client-to-client communication has latency 20 ms and bandwidth 2.2/9.3 Mbps for upload/download, which reflects the network properties in Prague. Server bandwidth capability is 100/100 Mbps for upload/download and the client-to-server latency is 114 ms.

Table 1. Typical download/upload bandwidth condditions

Type	Download [Mbps]	Upload	#Tests
Server	100	100	–
USA	3	0.6	–
Sweden	22.2	4.5	–
Japan	18.0	7.0	–
Prague, CZ	9.3	2.2	6467
UPC ISP, CZ	10.5	1.3	13195

Table 2. Massive download from server

Number of clients	1	20	30	40
Download time [ms]	6311	6860	6939	7297

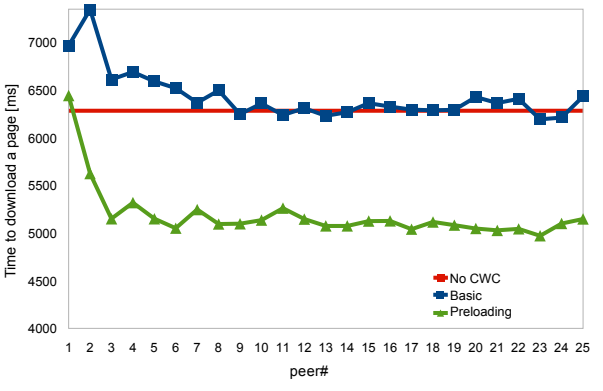

Fig. 3. CWC strategy evaluation

We measure the time it takes to a client to download the web page. An average download time of a single client downloading from the server with *no CWC* is **6.3** seconds (10 measurements). The time gets worse when multiple client load from the server as shown in Table 2.

The *basic*, *cache miss avoidance* and *preloading* CWC strategies described in Section 2 are compared with standard communication (*no CWC*). The results from the measurement are available in Fig. 3. Axis x,y of graphs represent the number of a client/peer and a total page download time respectively. For each measurement the first client requesting the page was fully served by the server, it is because the page resources were not stored in the CWC network yet. An average time for downloading a page from a server for the first client of the *basic* strategy is **6.9** seconds. The time difference of 0.6 seconds is caused by searches for files not available in the CWC network. When the *cache miss avoidance* strategy is employed, the average download time for the first client drops to **6.4** seconds, representing 7.3% improvement. The *preloading* strategy achieves similar results for the first peer. When a second client requests the page, the resources are fetched from both server and the first client. The page load time for *basic* strategy has worse load time, which is caused by lower first client upload compared to the server. Increasing the number of clients in the same CWC group allows a client to request service results from multiple locations, which positively impacts the overall load time. The saturation of the load time is in the range of 6-8 clients. In such case the load time drops to **6.3** seconds for both *basic* and *cache miss avoidance* strategies, respective **5.2** seconds for *preloading* strategy. An inspection of client participation shows that clients are

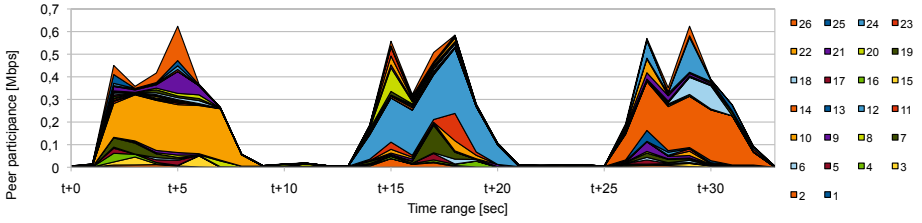

Fig. 4. Example peer participation in upload

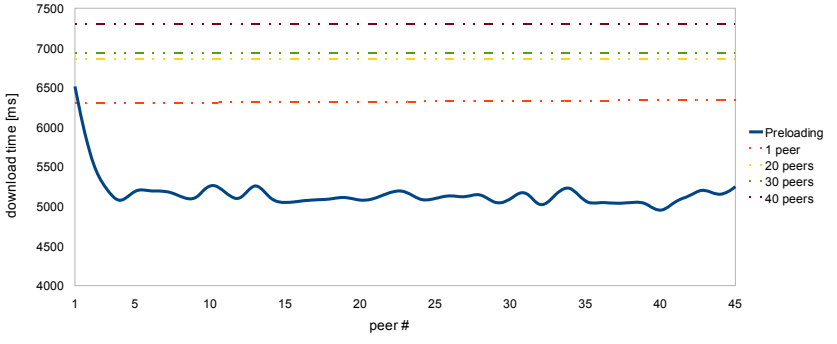

Fig. 5. Massive server download - compared to CWC with preloading strategy

not using the whole download capacity for *basic* strategy. However, when using *preloading* the download capacity was brought close to the limit distributed to all group members, although, no fairness mechanism was applied. In Fig. 4 is shown an example client participation in upload to a given client. The figure shows three consequent web page downloads in a CWC network consisting of 26 CWC clients with shared cacheable web page resources.

We also evaluate multiple clients downloading the same web page from a server at the same time. Fig. 5 shows that the *preloading* strategy is significantly faster than standard server download (no CWC).

3.2 Client Evaluation in Heterogeneous Network

This evaluation considers clients with different network capabilities. The scenario is the same as in the homogeneous evaluation. Clients has network conditions as in Prague, UPC, and Sweden listed in Table 1. The client-to-client latencies for all locations are specified in Table 3. Initial web page load times for evaluated locations (without CWC) are shown also in Table 3. Our evaluation with CWC network provides results in Fig. 6. The first requesting client is from Sweden and its load time is **4.3** seconds. Later load times for Swedish clients converge to **2.9** seconds. Prague load times converge to **5.6** seconds. Compare to homogeneous

Table 3. Location settings for peers

Location	Download	Upload	Latency [ms]				Web page load time
	[Mbps]	[Mbps]	Sweden	Prague UPC	Prague	Server	[ms]
Sweden	22200	4500	20	40	30	114	4303
Prague UPC	10500	1300	40	60	50	134	9497
Prague	9300	2200	30	50	40	124	6311

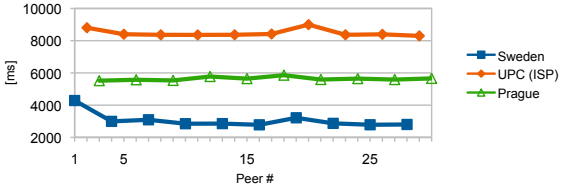

Fig. 6. Progressive heterogenous environment

network the result reflects increased delay that aggregates with number of resources. Prague UPC load times converge to **8.4** seconds.

3.3 Server Load Evaluation

From the server point of view the load can be seen by the number of hits or by downloaded data over a duration of the test. We compare requests going straight to the server with the use of CWC and without. Fig. 7 shows data downloaded from the server during the test. The green curve suggests that once the data is stored in the cooperative web cache the server load declines by 90%. This figure coincides with the ratio of dynamic content on a web page discussed above. Fig. 8 showing the number of requests to the server supports this idea. It should be noted the amount of downloaded data during the first five seconds roughly corresponds to the size of the page and its resources. After the first client downloaded some resources he provides them to other requesting clients.

3.4 Server Delay Dependency

Motivation for this test is discovering the boundary of server distance with which it is advantageous (performance-wise) to use CWC. For each bandwidth setup from Table 1 we compared download times of a client downloading directly from the server and a CWC client downloading from a network of 10 CWC clients. Fig. 9 shows that for Prague-like bandwidth it is useful to download page cached in a CWC network from server-to-client distances greater than 170 milliseconds. The same applies to UPC ISP and USA bandwidths with RTT 250 ms and

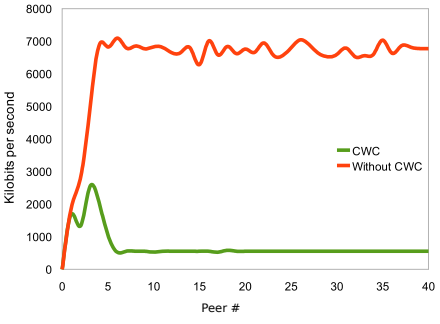

Fig. 7. Server load - data flow

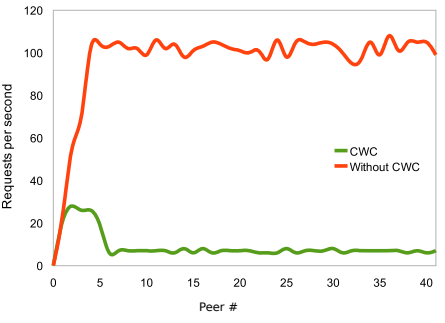

Fig. 8. Server load - requests

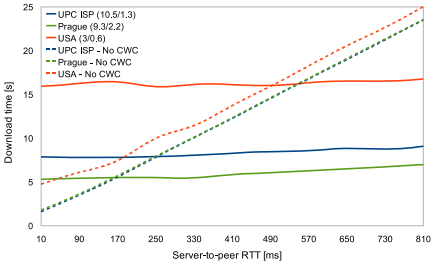

Fig. 9. Server delay - dependency between client-to-server RTT and speed

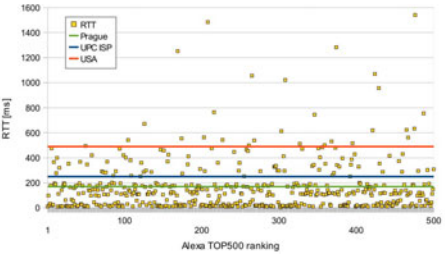

Fig. 10. Websites with improvable RTT values by different CWC setups

490 ms respectively. Fig. 10 shows RTT values from Alexa TOP500 mentioned above. The three horizontal lines represent efficiency boundaries for different setups. Access to websites above the lines could be improved by using a CWC network. The percentage of improvable websites by Prague, UPC ISP and USA bandwidth setups is 34%, 22% and 6.4% respectively.

3.5 Resilience to Abrupt Departure

Simulated departure of 5% and 10% clients from a network of 50 clients with all static sources in their caches was made. All peers left at the very same moment (no TCP FIN packet has been sent). An instant after the departure a new client joined the CWC network and downloaded cached page. The measured page download time is show in Table 4 (ten measurements). The referential download times were measured in a healthy CWC network with the same number of clients - for 5% and 10% departure 48 and 45 clients respectively.

Table 4. Resilience to abrupt departure of clients

	Network	Time [ms]	Deviation [ms]
5% DEPARTURE	Healthy	5124	96
	After departure	5435	349
10% DEPARTURE	Healthy	5109	97
	After departure	5523	414

4 Related Work

Survey of web caching [15] (section 4.1.2) contains early (1999) thoughts how a distributed caching of web content might look like and focuses on exchanging content between institution and national level caches. Distributed web caches are topics of multiple researches, for example Yingwu [16] shows their potentials and benefits. Squirrel [17] tries to engage clients in web content distribution and also builds on Pastry. It is similar to our proposal with multiple aspects, however, it aims only for corporate LAN networks and lacks anycast and multicast capabilities. Squirrel node, hosting a popular file, is bound to be eventually overloaded because all traffic for that given object is routed through a "home node". This also applies to its "directory mode" where the home node keeps reference to a certain number of it's latest clients and redirects new clients' requests to them just holding off the inevitable overload with redirect responses. Scribe [10], [11] implements anycast and multicast on top of Pastry [9] and allows us to implement efficient invalidation/update mechanism (multicast) and better scaling of popular files using the anycast. Dalesa [18] aims to provide web content caching for LAN networks with a working prototype. Kache [19] is focuses on lookup performance, its authors present a method reducing the number of necessary hops between nodes to one. They do this by using $O(\sqrt{n})$ space on each node (where n is the number of nodes) and a probabilistic approach to content retrieval. A node has a certain (very low) probability it will not be able to fulfill a request for the content it previously advertised.

P2P networks have unique security considerations, because peers are more powerful than clients. They issue their own node IDs, act as routers, relay messages and issues with trust arise. Uniform random distribution of node IDs is fundamental operational presumption of Pastry and all similar networks. If an attacker can *choose node ID*, she can surround a victim node, isolating it from the rest of the network, partition the network into smaller pieces or become a root key holder. This issue could be solved by centralizing node ID issuing into the hands of trusted certificate authority. Legitimate peers would refuse to communicate with peer not able to produce a signed combination of node ID and timestamp. A malicious node might also tamper with relayed messages, posing as the closest node. Furthermore one client can act with many identities [20]. Castro et al. describe possible solutions [21], but this area remains largely unexplored and addressing these concerns will be crucial for a practical use.

Multiple applications and research topics in this area are distributed web caches/proxies that specialize solely on content caching and its redistribution known as Content distribution networks (CDN's). Survey on CDN's and its optimization proposals are provided in detail by [22] or [23]. Among the most known solutions are Akamai [1] or Squid [24]. Squid is a hierarchical web proxy cache consisting of multiple independent servers. The content is shared by simple multicast query mechanism. These CDN solutions are statically distributed based on service provider decision. The motivation behind our approach is natural cache population formed by clients and its relocation based on popularity. Our approach does not involve costs related with cache server maintenance (Squid) or costs for given service (Akamai). On the other hand this approach does not face neither security issues nor content invalidation.

Cloud Computing [2] supports application scaling, it provides the illusion of infinite computing resources available on demand, eliminates an up-front commitment by cloud users and has the ability to pay for use of computing resources on a short-term basis as needed. The disadvantage comes with the implementation and extended costs. Our approach pushes the responsibility of the server-cloud to a user-cloud with no charge or changes in the implementation and could complement the cloud-computing in order to decrease charges for service.

5 Conclusion

In this paper we suggest that Internet web services could take advantage of peer-to-peer overlay network that is formed by clients with similar interest in the given time frame. We have presented our prototype Cooperative Web Cache and its simulations which are relatively close to the real-world conditions and prove our concept. CWC provides benefits to end-users who may experience improved download times for relatively common cases, benefits to service providers involve significant savings and better capability to survive peak loads. Although the beforementioned results are promising, we are aware that CWC concept is missing some key properties, such as proper security and content invalidation, which are subjects to further research. The invalidation scheme may impact performance as well as security. Security for decentralized approaches is more complicated and may require addition centralized mechanism issuing certificates or Facebook-like friendships. In order to provide chunking strategy for CWC the underlying network must provide manycast service.

Future work includes extensive emulation environment. We were able to emulate a network of at most about a fifty nodes with our hardware. Using more computers (for example on PlanetLab) would better reflect viability of our proposal in practical terms. Introducing jitter or packet-loss into our environment with heterogeneous network would provide more real-world performance results. Efficiency of CWC relies on a large user population and therefore it should be implemented in the most user-friendly way as a web browser plug-in.

References

1. Nygren, E., Sitaraman, R.K., Sun, J.: The akamai network: a platform for high-performance internet applications. SIGOPS Oper. Syst. Rev. 44, 2–19 (2010)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical report. Berkeley (2010)
3. Stevens, W.R.: TCP/IP illustrated: the protocols, vol. 1. Addison-Wesley Longman Publishing Co., Inc., Boston (1993)
4. Ullman, C., Dykes, L.: Beginning Ajax. Wrox (2007)
5. Cerny, T., Jaromerska, S., Praus, P., Matl, L., Donahoo, J.: Cooperative web cache. In: 18th International Conference on Systems, Signals and Image Processing, pp. 85–88. IEEE (2011)
6. Swen, B.: Outline of initial design of the structured hypertext transfer protocol. J. Comput. Sci. Technol. 18, 287–298 (2003)
7. Cohen, B.: Incentives Build Robustness in BitTorrent (2003)
8. Matl, L.: System for source distribution to support web application load time (cz). Master's thesis. Czech Technical University (2011), `https://dip.felk.cvut.cz/browse/pdfcache/matllubo_2011bach.pdf`
9. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Liu, H. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
10. Druschel, P., Kermarrec, A.-M., Rowstron, A.: Scalable Application-Level Anycast for Highly Dynamic Groups. In: Stiller, B., Carle, G., Karsten, M., Reichl, P. (eds.) NGC 2003 and ICQT 2003. LNCS, vol. 2816, pp. 47–57. Springer, Heidelberg (2003)
11. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications 20, 1489–1499 (2002)
12. Speed matters, internet speeds report (2010), `http://www.speedmatters.org/2010report`
13. Jaromerska, S.: Environment for peer-to-peer application simulation with application on cooperative web cache. Master's thesis. Czech Technical University (2011), `https://dip.felk.cvut.cz/browse/pdfcache/jaromsla_2011bach.pdf`
14. Praus, P.: Framework for network management to support simulation of varying network conditions. Master's thesis. Czech Technical University (2011), `https://dip.felk.cvut.cz/browse/pdfcache/prauspet_2011bach.pdf`
15. Wang, J.: A survey of web caching schemes for the internet. ACM SIGCOMM Computer Communication Review 29, 36–46 (1999)
16. Zhu, Y.: Exploiting client caches: An approach to building large web caches. In: Proceedings of the 2003 International Conference on Parallel Processing, ICPP 2003 (2002)
17. Iyer, S., Rowstron, A., Druschel, P.: Squirrel: A decentralized peer-to-peer web cache. In: Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing, pp. 213–222. ACM (2002)
18. Dalesa: The Peer-to-Peer Web Cache, `http://www.dalesa.lk/`
19. Linga, P., Gupta, I., Birman, K.: Kache: Peer-to-Peer Web Caching Using Kelips (2004)

20. Douceur, J.R.: The Sybil Attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)
21. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.: Secure routing for structured peer-to-peer overlay networks. ACM SIGOPS Operating Systems Review 36, 299–314 (2002)
22. Ball, N., Pietzuch, P.: Distributed content delivery using load-aware network coordinates. In: Proceedings of the 2008 ACM CoNEXT Conference, CoNEXT 2008, pp. 77:1–77:6. ACM, New York (2008)
23. Bakiras, S., Loukopoulos, T., Papadias, D., Ahmad, I.: Adaptive schemes for distributed web caching. J. Parallel Distrib. Comput. 65, 1483–1496 (2005)
24. Spare, I.: Deploying the squid proxy server on linux. Linux J. (2001)