Paper ID: 378

Student Paper: Self-organized Autonomous Web Proxies

Markus J. Kaiser
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Kowloon, Hong Kong
mjk@gmx.it

Kwok Ching Tsui
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Kowloon, Hong Kong
tsuikc@comp.hkbu.edu.hk

Jiming Liu
Department of Computer Science
Hong Kong Baptist University
Kowloon Tong, Kowloon, Hong Kong
jiming@comp.hkbu.edu.hk

ABSTRACT
With the increasing size of the Internet, proxy servers have emerged as a feasible way to reduce the overall network load and latency. More recently researchers focused on new ways to combine multiple cooperative proxies into one transparent proxy system to further increase the overall performance gain, but no work so far was really able to propose an ideal trade-off between content dissemination and clustering in a changing environment caching environment. This paper introduces a self-organizing approach to combine multiple autonomous proxies into one transparent proxy system One of the emerging attributes of a system of self-organizing autonomous proxies is a balance between content clustering and data dissemination. Our experimental results show that such a system outperforms conventional cooperative proxy infrastructures.

Categories and Subject Descriptors
I.2.11 [**Distributed Artificial Intelligence**]: *Coherence and coordination, intelligent agents, Languages and structures, Multiagent systems*

General Terms
Algorithms, Management, Measurement, Performance, Design, Reliability

Keywords
Proxy, Load Balancing, Data Clustering, Self-Organization

1. INTRODUCTION
Internet traffic is growing exponentially and increases the need for methods to minimize network latency. Proxy servers have been shown to be a well-suited approach to help improving the network performance. They are usually placed between the client and the origin server and act as an intercepting proxy cache [6]. Their advantages start with reduced bandwidth consumption, a reduction in latency due to data dissemination and a reduction of the load for remote origin server [1][10]. Recent approaches try to combine a set of multiple proxy servers into one cooperative proxy system with shared caches acting transparently as one proxy server. These cooperative proxies usually share the knowledge about their cached data and allow a faster document fetching through request forwarding [11], but non of this approaches was able to maintain a stable trade-off for ideal performance characteristics. A new approach based on the self-organizing nature of autonomous objects, as proposed in this paper, seems promising to expand beyond the limitations of existing cooperative approaches.

AAMAS-'02, Month 1-2, 2000, City, State.

1.1 Cooperative Proxy Systems
Existing cooperative proxy systems can be categorized into hierarchical & distributed proxy systems [2][12][17][6]. The hierarchical approach, first introduced by the Harvest Caching Project [24][25], is based on the Internet Caching Protocol (ICP) and a statically assigned hierarchy of proxy servers. A page not in the local cache of a proxy server is first requested from neighboring proxies on the same hierarchy level. If still unresolved the assigned root-proxy in the hierarchy will be queried and unresolved requests continue to climb the hierarchy and often lead to a bottleneck situation on the main-root server.

The distributed approach is usually based on a hashing algorithm like the Cache Array Routing Protocol (CARP) [16]. In a hashing-system each requested page is mapped to exactly one proxy in the proxy array and will be either resolved by the local cache or requested from the origin server. Hashing-based allocations can be widely seen as the most ideal way to find cached web pages, due to the fact that their location is pre-defined and the search algorithm requires no further overhead, but their major drawback is inflexibility and poor adaptability [15][21].

Additional work for distributed proxy systems, like Adaptive Web Caching [5] and CacheMesh [7], try to overcome specific performance bottlenecks. For example, in Adaptive Web Caching through dynamically created proxy groups combined with data multi-casting, while CacheMesh computes the routing protocol based on exchanged routing information. Both approaches can still be considered experimental and didn't reach the widespread acknowledgement like CARP or ICP. Yet other approaches like pre-fetching, reverse proxies and active proxies [11] can usually be seen as further improvements to speed-up the performance of a

general hierarchical or distributed infrastructure and go hand in hand with our proposed self-organizing approach based on intelligent autonomous proxies.

1.2 The Ideal Cooperative Proxy

The major challenges of a cooperative Proxy environment can be classified into the following areas: *content allocation*, *cache-usage*, *load balancing* and *reactivity towards a changing infrastructure*.

Content allocation describes the challenge of finding the location of a currently cached web page in a set of distributed proxy servers. An ideal cooperative proxy system knows exactly where to find requested data. Assuming that cached content is highly dependent on the current request pattern and that a cache is usually limited in size. Ideally each cache object is assigned to an exact set of locations, where it can easily be found and retrieved by other proxies. In hierarchical caches, proxies usually exchange cache summaries through inter-proxy protocols and learn about the content of neighboring proxies. If no knowledge is available a simple search process is initiated which occurs a high overhead. In hashing-based algorithms this problem is solved through a direct, and therefore ideal, assignment of cache objects to a specific proxy through, for example, a simple modulo calculation over the request URL. However, this approach is inflexible and adapts poorly to a changing infrastructure.

Cache-Usage tries to maximize the usage of all combined proxy caches in the cooperative proxy system, through minimizing the number of duplicate copies in neighboring proxies. An ideal cooperative system minimizes redundant data in neighboring proxies to maximize the overall cache-usage. It should be pointed out that we emphasize on avoidance in neighboring proxies, due to the fact that for the client its insignificantly from where to fetch the object from but its more important to have a realm of cached data. Hierarchical systems usually do not consider this issue. Hashing-based allocations solve this problem again ideally through an ideal mapping of cached objects onto the set of proxy servers.

Load Balancing describes the situation of ideal content dissemination, regarding the current request pattern, hot-spots or unused resources. It is usually true that a client requesting an cached object from a close proxy experiences a lower latency than a request for an object from a remote server. Hierarchical proxies, store copies of the requested objects on the path down the hierarchy. Subsequent requests for the same object have a high likelihood to be fulfilled locally. It should be pointed out that load balancing interferes with the idea of a maximized cache usage and also with the goals of content allocation. Maximized Cache usage, tries to minimize data dissemination whereas content allocations is simplified with low system reactivity towards the current request pattern.

The last challenge concerns adaptability regarding *changes in the underlying infrastructure*. A good distributed proxy system should be able to scale well with newly added resources, and more importantly should fail gracefully with the removal of resources. In real-life, proxies will be added to and removed from the network. In the current implementation of hierarchical and distributed approaches, each proxy usually comes with a predefined knowledge of other neighboring proxy servers. The configuration is mostly based on the individual decision of the system administrator and does not necessarily represent an ideal scenario regarding the current network situation and usually lacks flexibility in hot-spot situations and changes in the network traffic. An ideal cooperative proxy system should be able to react towards changes in the underlying proxy infrastructure; it scales well and dies gracefully.

1.3 Previous Work

A general cooperative proxy system can be divided into three layers: Clients, Proxies and Servers. In an ideal scenario with neighboring proxies, as shown in Figure 1, an incoming request for Server B, for example, will be forwarded to one specific proxy (ideal allocation, no unnecessary data dissemination), this proxy will acknowledge that it is dedicated for Server B and try to fulfill the request.

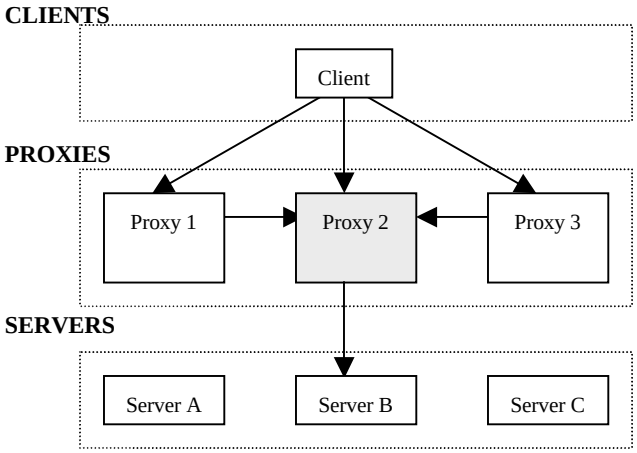

Figure 1: System Layers

In previous work by our group [8] we placed a central decision making entity between the client and all proxy servers (assuming we have a small system, e.g. institutional level) to promote ideal load balancing in respect to the proxies' individual performance characteristics. The work introduced a single point of failure and did not consider the cached data.

The logical next step, which we present in this paper, is to place a decision-making component into every proxy component (autonomous proxy). The new idea is based on the assumption that after a learning phase, the autonomous proxy will self-organize [23], [26], [3] themselves in such a way that similar unresolved requests will be forwarded to the same devoted and ideal proxy.

1.4 The proposed Approach

In real life a self-organizing proxy architecture based on autonomous proxies can be compared to a simple market buyer-seller environment where the buyer (client) acts as a "dumb" customer choosing always the same shop for all its requests (similar to pre-defined proxies in web-browsers). The maximization of the market depends completely on the sellers (autonomous proxies). Each shop has a limited local stock (Cache) and the goal to maximize the customer satisfaction. There are two ways in supplying good service, either by having the

requested item in the local stock or by knowing the most suitable way to supply the item. Additionally each proxy tries to attract more requests (not customer but other proxies) by specializing on a certain category of items (clustering). This decision is usually made, based on the current specialization of the shop and the incoming request pattern.

Self-organization in general, describes the ability of a system to maximize itself towards a specified goal. Based on the local decisions of a large number of autonomous objects emerging attributes will lead the autonomous society as a whole toward a pre-defined objective and will settle down in an ideal stable state. The previous scenario of seller-buyer objects in a dynamic market is a very fitting analogy for the goal to maximize the hit rate for proxy requests in a distributed autonomous proxy environment. The subsequent chapters describes design and attributes of a simple self-organizing autonomous proxy based on the stated scenario.

2. SELF-ORGANIZING AUTONOMOUS PROXIES (SOAP)

In this section we will describe in detail the components of the proposed approach and show the design of a self-organizing autonomous proxies (SOAP): cache, routing table, forwarding function, feedback function and selective caching.

2.1 Request Comparison

The biggest challenge for the proposed system lies in the assumption that we are able to compare and categorize incoming requests regarding their similarities. In an ideal situation, each incoming request is stored with its feedback values. This ideal approach would not only lead to an enormous amount of data, but it would also require a high number of requests per object to distinguish ideal from inappropriate decisions. Learning by feedback includes a learning period, and in our case the learning period can be presented in number of requests.

We need to define categories in such a way that we have a minimum number of requests to reduce the amount of feedback data and to acquire the necessary minimum number of requests per each category to make suitable assumptions about a chosen forwarding path. On the other side, the categories should be fine enough to allow a good data dissemination and to minimize overlapping data in neighboring proxy caches. In the context of hashing-algorithms, a simple modulo function, for example, can define similarity, over the requested URL. But such a simple categorization cannot be sufficient.

A more sufficient approach would be based on keywords distribution in the returned data, but this approach could not be simulated due to time limitations and shall be part of the future work. For the purpose of this paper we defined request comparison based on the main-domain of a given URL, e.g. “sport.com”.

2.2 Components

2.2.1 Cache Replacement Scheme

Each proxy comes with a local cache, a data space for storage and retrieval of transferred objects. In the case of a full cache, the caching of a newly arrived page is preceded by the removal of an existing page. We chose to simulate the Least Recently Used algorithm as one of the most common and well-suited algorithm for a proxy cache replacement scheme.

2.2.2 Routing Table

The routing table stores numerical values, used by the forwarding function, to select an ideal forwarding path of an unresolved request. The table contains one row for each known category and the columns represent the list of all known proxy objects (or in more realistic scenarios, a subset). In other words, for each category the proxy has multiple paths to choose from to fetch the requested object. Each table entry represents the accumulated average value for the last n requests forwarded through this path and is calculated by the request feedback function.

Table 1 : Example Weight Table

	DIRECT	PROXY 1	PROXY 2	PROXY 3
Search.com	1.0	0.03	4344	100
Sport.com	1.0	0.7	545	43
Riddles.com	1.0	3424	0.005	0.0002
Travel.com	1.0	50	0.45	435

2.2.3 Forwarding Function

The forwarding component is triggered when a request could not be resolved by the locally cached data and data in the routing table is used to find the most suitable path for the unresolved request (see Figure 2). After identifying the main-domain of the request, it looks up the specific row and calculates a weight for each entry in regard to the direct proxy-server communication. Table 1 shows an example table after the weight calculation. The weight for a direct proxy-server communication receives always the value 1.0 and represents a reference value for comparison. Proxy paths that are usually two times faster than a direct communication will receive the weight 2, and a path that is half as fast as the direct communication will receive the weight 0.5.

FOR I = 1 TO (PROXIES + 1)

IF (VALUE [i] == DIRECT)

$$weight[i] = 1.0$$

IF (VALUE [i] > DIRECT)

$$weight[i] = \frac{value[i] - rowMax}{rowDirect - rowMax}$$

ELSE

$$weight[i] = \left(\frac{value[i] - rowDirect}{rowMin - rowDirect} \bullet scale \right)^{stability}$$

DO

weight[i] : the computed weight for the i-th column
rowDirect: the plain Value for a direct forwarding
rowMin : the row plain value Minimum
rowMax : the row plain value Maximum
scale : a value greater 10
stability : value around 50

Figure 2: Forwarding Function

Note, that a high value in the core table always leads to a low value in the calculated weight table to avoid this path for future selections. All weights will be normalized and used as a probability value for each possible path for this category. A small random noise value is added to avoid the system to become stuck in local minima. The final path is randomly chosen based on the received probabilities.

2.2.4 Feedback Function

The feedback function is executed after a forwarding proxy received the returned data object (see Figure 3). It should be mentioned at this point that for simplification purposes we assume all are objects of equal size and future work should adapt the algorithm to objects with different sizes. The feedback function will use the received latency value to update the appropriate cell in the routing table. Basically, each value in the routing table is the average latency of the last n requests for this path. The formula is based on the simple weighted average calculation for the tracking of a non-stationary problem.

$$table_{(i,t)} = \frac{table_{(i,t)} \bullet memory + latency}{memory + 1}$$

latency : new returned latency for this request
memory : constant value, representing the number of last n values included in cell average
table (i,t) : represents table value in position (i,t)
i : request category, table row
t : chosen path, table column

Figure 3: Feedback Function

2.2.5 Selective Caching

The selective caching component decides if the data from a resolved request will be added to the local cache or discarded (see Figure 4a,b). As described earlier selective caching is introduced to promote efficient clustering based on the overall traffic pattern and not just on the recently observed requests. Selective Caching uses the current cache status for the requested category and calculates a ratio in regard to the category with the highest page assignments. The ratio is adapted with a small random noise, to allow categories with a low ratio to leave their minimum in case of a sudden change in focus on a certain category.

IF (# OF NEWPAGE.GETCACHED < MINIMUM)

$$Caching = Random(\max Noise)$$

ELSE

$$Caching = \frac{newPage.getCached() - \min Cached}{\max Cached - \min Cached}$$

$$Caching = Caching \pm Random(\max Noise)$$

END

Figure 4a: Selective Caching

Caching : probability value to cache the new objects
Minimum: a lower bound for categories close to zero
MinCached : Category with smallest # of cached objects
MaxCached : Category with greatest # of cached objects
NewPage : Category of the new object

Figure 4b: Selective Caching

2.3 Algorithm

The following algorithm describes the core steps in a simplified version without emphasize on the evaluation subroutines (see Figure 5). It should be clarified at this point, that all autonomous proxies are based on exactly the same algorithm.

```
WHILE (NOT END)
    WAIT FOR REQUEST
    CHECK LOCAL CACHE
    IF (PAGE EXISTS)
        UPDATE LEAST RECENTLY USED CACHE
        SEND DATA TO REQUESTER
    ELSE
        IF (HOPS > MAX HOPS)
                FORWARD TO ORIGIN SERVER
        ELSE
                DO FORWARDING FUNCTION
    RECEIVE DATA
    DO FEEDBACK FUNCTION
    DO SELECTIVE CACHING
    SEND DATA TO REQUESTER
DO
```

Figure 5: Algorithm

3. EXPERIMENTATION

In order to show the ability of the proposed algorithm to provide the objectives of an ideal proxy environment, we tested the simulated system first on its ability to load balance and cluster in reaction to experienced hot-spot situations. Additionally we tested the algorithms ability to adapt to changes in the underlying infrastructure. The gained results will show how a system of multiple self-organizing distributed autonomous proxies adapts to the request pattern in such a way that load balancing and data-clustering will emerge. Additionally the results will also proof that the system is well able to adapt to radical changes in the autonomous proxy infrastructure, like removal and addition of proxy resources.

3.1 Architecture

The infrastructure used for later simulations is similar to a common institutional proxy environment; with for example, up to 10 proxies. We assume that in this context, going to any proxy within the array of proxies, even by doing a maximum number of hops, is always faster than requesting the data from the origin server. The same applies to an overloaded proxy object. A local hit will always be faster than a remote request. In such a scenario, an remote server appears to be very far away and each server a very high, constant latency value.

We further assume full knowledge and full connectivity within the proxy layer, so that each autonomous proxy knows about every other autonomous proxy and they are capable of connecting to each other without network restrictions. Furthermore we assume that each proxy is equally able to connect to the 20 servers with the same latency value, e.g. Figure 6.

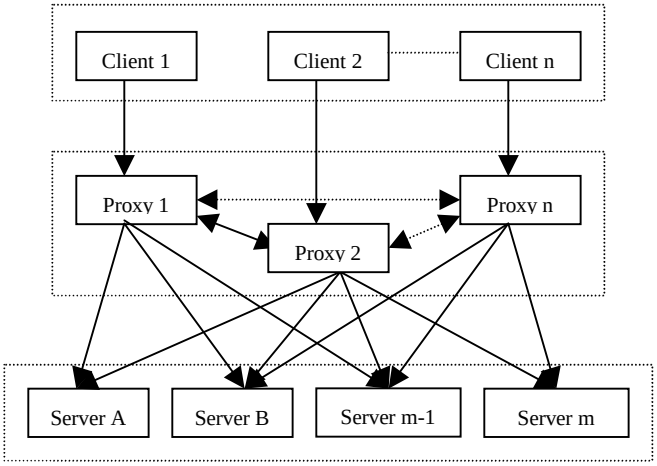

Figure 6: Simulated Scenario

3.2 ZIPF-Request Pattern

Resent research has shown that a power-law or ZIPF-law distribution is very suitable to describe the experienced request pattern on a proxy server [4]. In our simulation, clients inject requests for all existing servers based on a ZIPF-distribution.

In the simulation, we wanted to be able to artificially create hot spot situations. A hot spot is defined in such a way that the algorithm first chooses a subset of origin servers, which shall be part of the hot spot, and the ZIPF distribution will be placed upon this subset in such a way that chosen servers will be the most requested ones, and the remaining set of servers receive almost no requests. Future work with a real system shall broaden these limitations.

Preliminary simulations have shown that the outcome of the simulations are highly dependent on the request pattern. We simulated a uniform distribution and also a normally distributed pattern over all server pages and gained results with similar but scaled performance curves.

3.3 Distributed Hashing

All experimentations are comparing to a classical distributed hashing approach where an incoming request is always assigned to a specific request based on a simple modulo calculation. In our simulations for the hashing algorithm, each of the 10 autonomous proxies is pre-assigned responsible for two of the origin servers, e.g. proxy 1 caches data for server A & B, and so on. We will show, that this pre-assignment wont adapt to the request pattern, a server overload will occur for hot-spot situations with lower performance values than a self-organizing autonomous proxies approach. The second test shows the adaptability of the proposed approach to changes in the proxy infrastructure without comparison to the hashing algorithm, due to the fact that classical hashing approaches are not able to adapt to changes in the number of resources. Present research in the area of consistent hashing algorithms tries to overcome this limitation through a feedback function based on multi-casting groups [27].

3.4 Test 1: Dissemination & Clustering

The following tests show the ability of the system to adapt to an incoming hot-spot request pattern with focus on a specific subset of 20 servers. This kind of situation occurs, when a sudden hot-spot emerges and many clients are focusing on a specific topic (“America vs. Terrorism”) The same request pattern is also used for a classical hashing system and we should see a performance gain in hot-spot situations due to load-balancing and content dissemination while the hashing-based system should outperform the self-organizing approach in situations with almost no hot-spots, due to uncertainties and the need for an adaptation period.

Performance Gain. As seen from Figure 7, the self-organizing approach works much better than the plain hashing-approach when it comes to clear hot-spot situations. And as predicted the performance gain decreases with decreasing hot spots in comparison to a pre-defined hashing algorithm. This limitation will be considered in future work together with an adaptive URL-comparison and categorization.

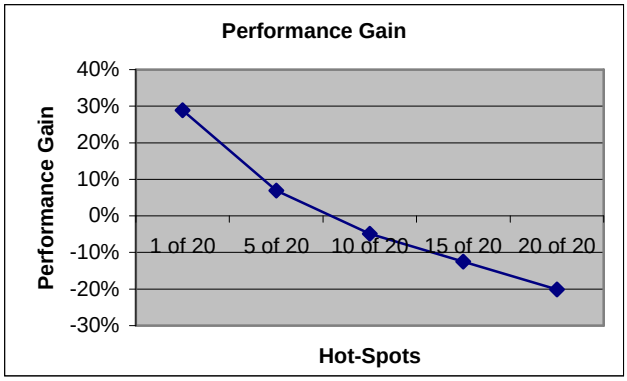

Figure 7: Performance gain compared to plain Hashing regarding Hot-Spots

Looking at *the average number of hop* request needs before it gets resolved, we can see that both approaches require close to the prediceted medium value of 2 hops. A hop is defined by a request moving from one host (client, proxy or server) to another host. Figure 8 shows, that the self-organizing autonomous proxies are able to build a stable infrastructure with clustered content minimizing unnecessary request forwarding.

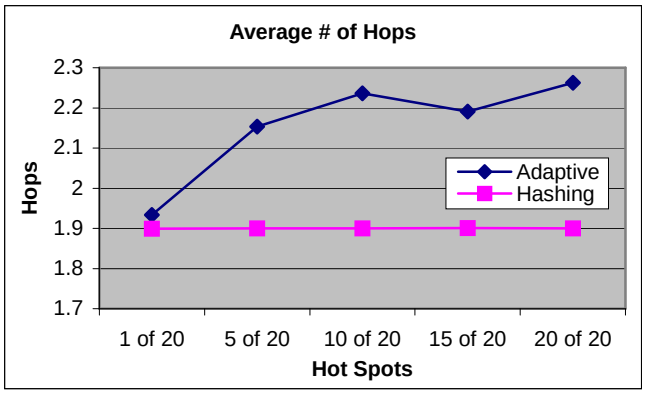

Figure 8: Average number of Hops in regard to the number of Hot-Spots

Figure 9, which displays the *average hit rate* for both SOAP & hashing, shows the same behavior as the previous diagrams. In hot-spot situations with a focus on a certain set of data, the adaptive approach performs better than the hashing approach, but with increasing request distribution over the whole number of sets the performance of the adaptive approach decreases.

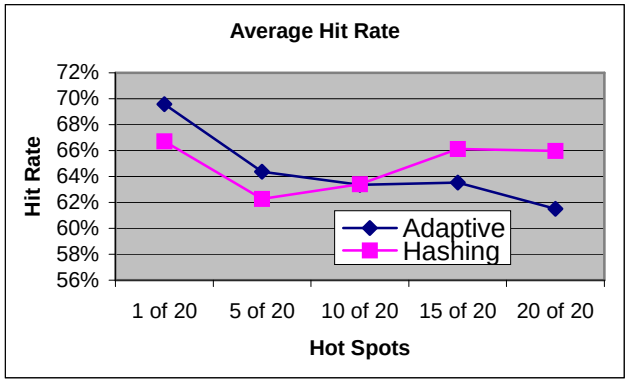

Figure 9: Average Hit Rates in regard to the number of Hot-Spots

The evaluation of *Content Clustering* shows that the number of data clusters is dependent on the number of requested hot spots. In our system with 10 autonomous proxies, clusters will emerge based on the current hot-spot distribution.

1 of 20: Only one type of content is requested and all proxy servers will fill their caches with the same category of data.

5 of 20: Clusters will emerge in such a way that always two of the proxies focus on the same type of data. Five combined clusters were created as a result.

10 of 20: The data for 10 hot spots are equally distributed over the 10 proxies allowing each one to focus on one specific category. The system becomes stable with 10 different clusters.

15 of 20 & 20 of 20: more random clustering are observed with some each proxy having a small percentage of all categories available.

3.5 Test 2: Changing Infrastructure

The following test scenario proofs that the system is able to react to changes in the proxy infrastructure. In the simulated example, the overall performance of an individual proxy server will be raised. Decreasing the proxy performance to a bare minimum is equal to removing this proxy out of the system, due to the fact that the proxy latency will increase immensely. As we can see in Figure 10, the remaining autonomous proxies are adapting to the newly created situation and even if they are not able to make up for the lost resources, they try to minimize the negative impact through further load balancing.

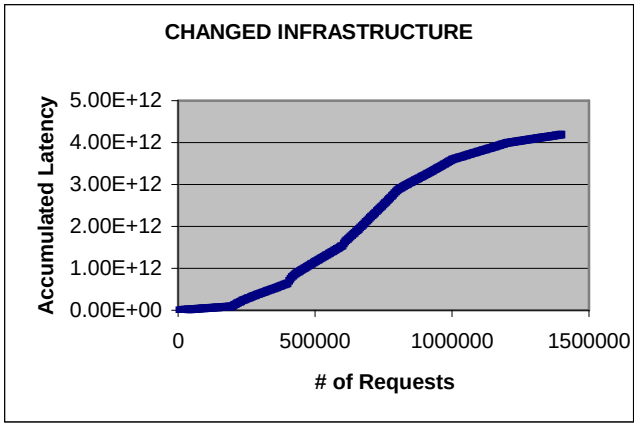

Figure 10: Changed Infrastructure, in the first half of the graph resource will be removed, in the second added

In the first half of the test, 0 to 750000, we removed proxy servers every 200000 requests, from 10, to 6, to 3 to 1 running proxy server. The figure shows clearly that the steepness of latency curve increases reaching its steepest at around 800000 requests. Afterwards, requests from 1000000 to 1500000, we added new resources to the simulated system, from 4 to 7 and to 9 proxy servers. Clearly visible in figure 10 the curve is slowly getting flatter. The steeper the curve, the more latency added per request. A flat curve stays for good performance. The graph shows explicitly that the self-organizing autonomous proxies are able to adapt to resource changes.

Adding new resources in a real life scenario unfortunately includes the introduction of a minimum inter-proxy communication (a topic which was not addressed by this work). Running proxies need to be notified about a newly added resource, and this can either happen statically through a system administrator or in a dynamic way through a simple broadcast or multi-cast protocol. A simply protocol for the exchange of proxy lists shall be considered as part of the future work.

3.6 Discussion

Overall we can say that our current self-organizing proxy system performs as well as a common hashing algorithm and is able to outperform it in hot-spot situations commonly found in Internet traffic.

The results show us that the system is not only able to find a stable balance between content dissemination and clustering but it is also capable of adapting gracefully towards the addition and removal of proxy resources. Additionally, we can see that the overall system performance is highly dependent on the incoming request pattern and that the comparison and categorization of requests is a key factor for future tests. Besides the fact that the adaptive system performs as well as a common hash function, we also gain all additional features that define an ideal distributed proxy system like, load balancing, simple allocation, good cache usage and reactivity to changes in the infrastructure. Overall, the results show that self-organizing autonomous proxies are a promising approach to compete with or even replace existing cooperative proxy caching systems.

4. CONCLUSION

Our work was able to show that self-organizing autonomous proxies are a highly suitable architecture for a cooperative proxy environment. Its adaptive attributes give the system the necessary flexibility to cope with emerging hot-spot situations in a general network environment and we have shown that already a simple algorithm is able to compete with conventional hierarchical and distributed proxy systems. Further investigations will give more insights into ways to improve this already promising approach.

5. FUTURE WORK

Future work will especially focus on new ways to compare and categorize request patterns. We will try to make the comparing algorithm more adaptive and we will additionally try to replace the static system parameters with mechanisms of self-discovery. Load-balancing attributes in correlation to a system of proxy servers with different performance characteristics need to be tested. Furthermore the system needs to be expanded in such a way that it also reacts to different object sizes and the idea of proxy migration needs further exploration. Overall the proposed approach is not only applicable for proxy caching systems, but a more general view on resource allocation in a dynamic infrastructure needs further consideration.

6. ACKNOWLEDGEMENT

This work is supported by Baptist University research grant FRG/01-02/I-03.

7. REFERENCES

[1] Ghaleb Abdulla, Analysis and Modeling of World Wide Web Traffic, Dissertation submitted to Virginia Polytechnic Institute and State University (May, 1998)

[2] Mohammed Salimullah Raunak, A Survey of Cooperative Caching, (December, 1999)

[3] I. Kassabalidis, M.A. El-Sharkawi, R.J.Markus II, P. Arabshahi, A.A. Gray, Swarm Intelligence for Routing in Communication Networks,

[4] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker, Web Caching and Zipf-like Distributions: Evidence and Implications, Technical Report 1371, Computer Sciences Dept, Univ. of Wisconsin-Madison, April 1998

[5] Scott Michel, Khoi Nguyen, Adam Rosenstein, Lixia Zhang, Adaptive Web Caching: Towards a New Global Caching Architecture, In Third International Caching Workshop, June 1998.

[6] Greg Barish, Katia Obraczka, World Wide Web Caching: Trends and Techniques, IEEE Communications Magazin, May 2000

[7] Zhen Wang, Jon Crowcroft, Cachemesh : A Distributed Cache System for World Wide Web, NLANR Web cache workshop, June 1997

[8] Kwok Ching Tsui, Jiming Liu, Hiu Lo Liu, Autonomy Oriented Load Balancing in Proxy Cache Servers, Web Intelligence: Research and Development, First Asia-Pacific Conference, WI 2001, p.115-124

[9] Ari Luotonen, Kevin Altis, World-Wide Web Proxies, First International Conference on the World-Wide Web, Elsevier Science B, 1994

[10] Bradley M. Duska, David Marwood, Michael J. Feeley, The Measured Access Characteristics of World-Wide-Web Client Proxy Caches, In USENIX Symposium on Internet Technology and Systems, Monterey, California, USA, December 1997. USENIX Association.

[11] Jia Wang, A survey of Web Caching Schemes for the Internet, ACM Computer Communication Review, 29(5):36--46, October 1999.

[12] Pablo Rodriguez, Christian Spanner, Ernst W. Biersack, Web Caching Architectures: Hierarchical and Distributed Caching. 4th International Caching Workshop, 1999

[13] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, Henry M. Levy, On the scale and performance of cooperative Web proxy caching, SOSP-17, 12/1999

[14] Krishnanand M. Kamath, Harpal Singh bassali, Rajendraprasad B. Hosamani, Topology aware algorithms for proxy placement in the Internet

[15] Keith W. Ross, Hash-Routing for Collections of Shared Web Caches, IEEE Network Magazine, 11, 7:37--44, Nov-Dec 1997

[16] J. Cohen, N. Phadnis, V. Valloppillil, K.W.Ross, Cache array routing protocol v.1.1, Sept. 1997, Internet Draft

[17] Sandra G. Dykes, Clinton L. Jeffery, Samir Das, Taxonomy and Design for Distributed Web Caching, Published in the Proceedings of the Hawaii International Conference on System Sciene, 1999

[18] Alec Wolman, Geoff Voelker, Nitin Sharma, Neal Cardwell, Molly Brown, Tashana Landray, Denise Pinnel, Anna Karlin, Henry Levy, Organization-Based Analysis of Web-Object Sharing and Caching

[19] Tony White, Routing with Swarm Intelligence, SCE Technical Report SCE-97-15

[20] Greg Barish, Katia Obraczka, World Wide Web Caching: Trends and Techniques, IEEE Communications Magazine Internet Technology Series, May 2000

[21] Xueyan Tang, Samuel T. Chanon, Optimal Hash Routing for Web Proxies

[22] R.B. Bunt, D.L. Eager, g.M. Oster, C.L. Williamson, Achieving load balance and effective caching in clustered web servers*Proceedings of the 4th International Web Caching Workshop*, January 1999

[23] Jiming Liu, Kwok Ching Tsui, Jianbing Wu, Introduction to autonomy oriented computation, In Proceedings of 1st International Workshop on Autonomy Oriented Computation, pages 1-11, 2001

[24] A. Chankhunthod et al., "A Hierarchical Internet Object Cache"",Proc. USENIX Tech. Conf., 1996

[25] K. Claffy and D. Wessels, "ICP and the Squid Web Cache", 1997

[26] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz, Swarm Intelligence: From Natural to Artificial Systems

[27] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lwin, Rina Panigrahy, Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web