Towards fine-grained adaptivity in web caches

Olivier Aubert, Antoine Beugnard
ENST de Bretagne, Brest, France
Olivier.Aubert@enst-bretagne.fr

February 1999

Abstract

Web cache complexity is growing nearly as fast as the WWW is expanding. As the size increases, the architecture becomes more complex and the number of parameters involved in tuning is growing.

Tuning a Web cache involves a great deal of expertise, which is shared by many people around the world. But not every web cache administrator has the necessary knowledge to tune a cache and obtain good performances.

Based on this observation, we would like to model and develop a web cache which could tune itself, by taking into account its traffic and other parameters. We present here a model for a framework allowing various strategies to be dynamically changed during the server runtime[1].

Keywords: WWW, cache, adaptivity, design pattern, strategy

[1]This work is sponsored by the city of Brest, the *Region Bretagne*, FRED and FEDER.

1 Introduction

Previous experience with large system [17] has shown that their configuration should be as automatic as possible, thus allowing the system to respond to environmental changes rapidly.

Web caches, which are commonly grouped into large cooperating clusters, are no exception to this rule. They are tuned by operators today, but the increasing complexity will soon prevent them reacting to changes in network traffic or system load.

That is why we would like to investigate the field of modular web caches that have the ability to configure themselves. This self-configuration implies some adaptivity.

The adaptivity concept is already used in many domains. It relies on the idea that a system should adapt to its environment. If the external conditions change, the system should evolve accordingly. The underlying assumption is that a system – in the sense of an entity interacting with an external environment – cannot be considered independently of its environment.

Typical examples include graphical interface adaptivity and hypertext adaptivity [9], OS adaptivity [8] or routing adaptivity.

We have briefly introduced the adaptivity concept. In this article, we explain how it can be applied to web caches in section 2. Our model is presented in section 3: we will present the global model, the information and statistics model, then the various strategies isolated so far in section 4. We will mention similar work in section 5. Finally, a provisional timeline and future research ideas will be given in the conclusion.

2 Application to web caches

Adaptivity can naturally be applied to web caches. It can be implemented with different granularities, however. We mention these, ranging from coarser to finer, and describe in greater detail the fine-grained adaptivity that we have chosen to study.

The coarser granularity is used for instance in distributed web caches to organize the distribution of the data in the whole system dynamically. The AWC project [5] is cited as an example in section 5. It does not have a great influence on the inner working of each node of the distributed system, apart from the implementation of the distribution algorithm. The fine-grain approach considers that each node of the distributed system can benefit from an independent adaptivity of certain parameters or strategies. The authors of [2] have for instance shown the importance of the low-level details involved in a cache. We will here focus on the fine-grain approach, but the two granularities are of course not exclusive.

The idea of a fine-grained adaptive web cache stems from many sources. Amongst these are our previous interest in the adaptivity concept, and also numerous articles in the web caching domain which describe or study various algorithms. Many different algorithms have been proposed for instance for document replacement [13, 11] or consistency [14, 10, 12, 3] policies.

The study of a web cache configuration file, using Squid[18] for instance, reveals a significant number of tuning parameters: `cache_mem`, `maximum_object_size`,

`refresh_pattern`, `quick_abort`, `delay_pool`, timeouts, etc.

This observation, as well as the reading of the `squid-users` mailing-list in which many messages are dedicated to tuning questions, has led us to think that a fine-grained tuning of web caches (as opposed to the configuration of a group of caches, described later in section 5) can offer a significant improvement in performance.

The W3C (*World-Wide-Web Consortium*[15]) published a *Replication and Caching Position Statement*[17], in which they list a number of requirements for caches. One of them is *Limited Human Intervention*. It states explicitly that

> A human must dedicate resources (storage, processing, bandwidth) to the use of the system at some point.

That is especially true for the *routing or optimization configurations in large scale systems*, which are too complicated to be manually reconfigured as fast as the system evolves.

> For a large scale system, most replication and caching decisions must be completely automatic, though local policy decisions may be imposed over such an infrastructure.

Thus our system must not only be tunable, but it must be able to tune itself. Therefore the main aspects of our project involve:

- *introspection capability* of the caches to access the information necessary to take tuning decisions.
- *isolated politcies/strategies*, designed with the Strategy Design Pattern [6] in mind. A Strategy Design Pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable.
- decision rules.

3 Model

One of the goals of the project is to elaborate the model of a framework designed to integrate various strategies. We elaborated a preliminary version of the model, which we intend to prototype in order to refine it later. We briefly explain here our development choices, then describe the context of our global model. Finally, we give an overview of our information and statistical model.

We have chosen not to develop the entire prototype from scratch, in order to concentrate on the specific aspects of web caches. Thus we have chosen to use an existing product, Jigsaw.

Jigsaw [16] is the experimentation platform of the W3 Consortium [15] in which they evaluate the new protocols and standards which are proposed. It features a modular architecture, and has the additional benefit of being up-to-date with respect to the latest standards such as HTTP1.1 [4].

We consider a distributed cache system as a set of interconnected nodes. Each node can constitute a potential access point for the rest of the network. The access method is not strictly defined here, so it could be anything from a standard proxy access to a specialized access method to provide a transparent

caching system. All the nodes cooperate to increase storage size, as well as computation capabilities. From the client's point of view, the whole system is just one entity. Our model concerns only one individual node, which will feature, among other strategies, many strategies defining the relationship between this node and its neighbours. We will define some of these strategies below.

One node of the system is composed of a control entity called Cache, which processes information – Reference, DocumentCluster – according to the strategies described in section 4. Figure 1 represents a simplification of our model. The strategies appearing in it are described in section 4.

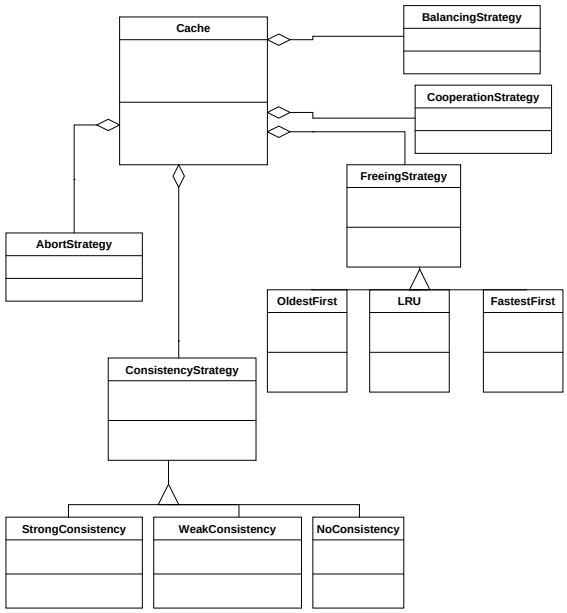

Figure 1: Global model

Information model

The information (see figure 2) appears as DocumentClusters: WWW requests typically involve approximately fixed sets of documents – for instance the inline images for an HTML document. It seems interesting to take this locality factor into account, in order to optimize the placement and the expiration method of the data.

The cache will mainly process DocumentClusters, while having the possibility to handle individual References. This is a modified *Composite* design pattern [6], that represents part-whole hierarchies of objects.

Our ultimate goal is to make our cache system *self-adaptive*. In order to achieve this, we have to gather statistical information which will allow us to determine the strategies that should be used.

Every event occurring in a node of our system has to be recorded. We especially record information about the documents that have been stored (and may have been purged since) in our cache. For the moment we have defined the following events:

DownloadEvent download of a document, i.e. its transfer from an external

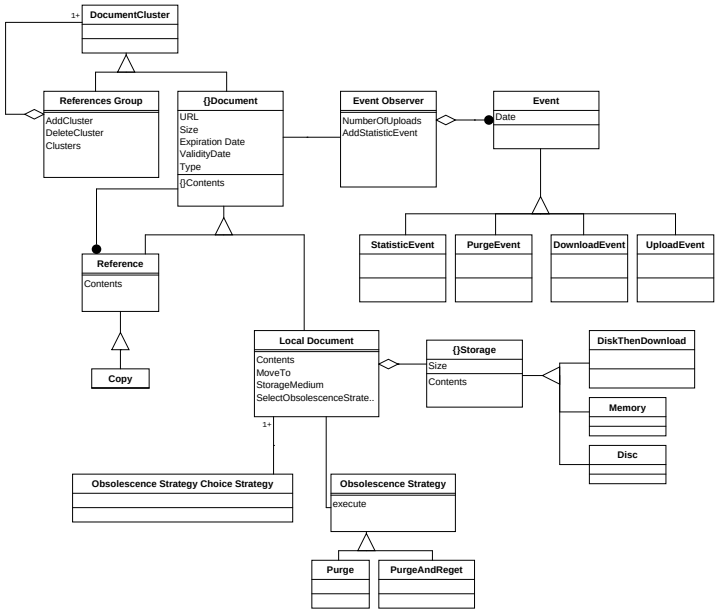

Figure 2: Information model

entity into the node.

UploadEvent upload of a document to a client.

PurgeEvent purge of the document from the node. Note that once a document is purged, we continue to keep track of it in our event log.

Each event contains useful parameters, such as the name of the document, the date, and other state information.

This information will accumulate very quickly, so we have to implement the ability to summarize a number of events into a StatisticsEvent.

4 Strategies

Figures 1 and 2 both feature strategies that do not apply to the same category of elements. We classify them in two groups: global strategies and local strategies.

Global strategies apply to a defined set of nodes. We have to think of a mechanism that will ensure that the global strategies are synchronized over a set of nodes.

CooperationStrategy This strategy allows us to establish the cooperation model used by the nodes. It is not necessarily global to the entire distributed web cache, but it has to be coherent between two cooperating nodes.

Existing cooperation strategies are ICP, CARP, Digests, WCCP, etc.

BalancingStrategy This will decide on which node the information retrieved is to be stored, and migrate the information from one node to another,

in order to balance the load or bring the data closer to the client. Its intersection with the CooperationStrategy is not empty, so we may have to merge them in the future.

ConsistencyStrategy The consistency of documents can be achieved in various ways. Many algorithms [14, 10, 12, 3] have been proposed, implementing weak consistency or strong consistency.

Local strategies can vary from one system node to another. They allow us to adapt our distributed system configuration internally. We can distinguish two degrees of locality: local to the node or local to a document.

Strategies local to a document can vary from one document to another. They take into account parameters such as size, download times, etc.

StorageStrategy decides on which storage unit the document is to be stored. The initial decision is made on the basis of known factors (document type, document size) but it can change as statistical information about the document is gathered (especially access frequency). We can envision many storage methods:

- onDisk only on disk.
- inMemory only in memory (small documents frequently requested).
- MemoryThenDisk for swappable documents. We store the document first in memory, and later swap it to disk.
- DiskThenNetwork for big documents that are located on fast servers: we can afford to store only the beginning of a file, and fetch the rest directly from the original server.

FetchingStrategy This strategy determines the method used to fetch, or refetch, a document. Statistical data that we have accumulated, especially the speed of the connection to the original site, as well as topological information about the other nodes in the system, can lead us to choose for instance to directly fetch the document rather than to use a cooperation method.

Strategies local to a node take into account a wider range of parameters.

FreeingStrategy The most commonly used FreeingStrategy is based on the LRU *(Least Recently Used)* algorithm, but many articles [20, 21, 19, 1] have proposed other methods.

AbortStrategy This is a mapping from the `quick_abort` configuration directive of Squid. It defines the way that the cache handles client-initated aborts. This parameter has a great impact on bandwidth consumption, so an adaptive strategy should be able to observe the bandwidth used, and adapt the AbortStrategy accordingly.

The framework that we plan to build should give us the ability to change a strategy. These changes can happen at different times:

- *cold change*, i.e. the classical definition in a configuration file, taken into account when restarting the server.

- *hot change*, during server runtime.

In order to provide true adaptivity, we must have the ability to modify a strategy during server runtime. This constraint introduces interesting problems.

First, one common feature of strategies - considered here as algorithms - is that they often need invariants. So how do we handle the invariant change during a strategy switch? What should we do in the case of conflicting invariants?

Moreover, as our model is local to a single node of the system, should we allow many *different strategies in cooperating nodes*? We will have to classify the extent of a strategy, which can be either local to the node, or global to the entire system.

Finally, we will have to study the *cost of a switch in strategy*. Our aim being to optimize performances, we have to take the switch time and the frequency of switches into account.

5 Similar work

We have noticed two similar projects. They both present adaptivity as an important feature, which permits greater performances and scalability.

The Adaptive Web Caching project [5] is an ongoing project at UCLA and LBNL. It aims at building a self-configuring Web caching system consisting of overlapping multicast groups of Web caches.

They propose to "develop self-organizing algorithms and protocols" that allow cache groups to dynamically adjust themselves according to changing conditions in network topology, traffic load, and user demand [5].

This project uses adaptivity at a greater level of granularity, thus not involving any low-level tuning of the individual nodes of the whole system.

The JAWS project [7] is closer to our project, but in another class of products, Web servers. Web caches and Web servers are of course not that different, but there are some features related to web caching such as distribution or document expiration that justify the development of two different systems.

The JAWS project has a very complete approach, including adaptive protocols and threads of control. Our goal is to be less intrusive, and use existing functions or protocols as far as possible.

6 Conclusion

This is work in progress. So far, a first version of the model has been designed and is being implemented, in order to have an experiment platform and determine the performances and shortcomings of our model.

One of the constraints of our project was to work with existing protocols and infrastructure, while keeping the ability to evolve with new standards. That is why we chose Jigsaw as our development platform. Its endorsement by the W3C and its modular structure seem to respect these constraints.

At the time of writing, there is still no working prototype. An implementation is being developed, which will follow these steps:

1. Implementation of the general framework, which can be divided into two parts: strategy handling and event collecting.

2. Implementation of at least two types of strategies: Storage and Freeing seeming to be good candidates.

3. Manual adaptivity, i.e. manual tuning of the cache in order to study the problem of switch in strategy, and the impact on performance.

4. Implementation of self adaptivity, which will require us to elaborate algorithms, heuristics or other methods to extract relevant pieces of information from statistical information.

Using strategies has brought other directions of research to our notice:

- *strategy combination/conflicts.* How do combinations interact, how can we specify the application range of a strategy, in order to ensure that there are no conflicts?

- *switch in strategy.* Switch time is a critical phase. How do we keep the invariant of the strategy and what is the state of our system during switch time?

References

[1] Jean-Chrysostome Bolot and Philipp Hoschka. Performance engineering of the world wide web: Application to dimensioning and cache design. In *Fifth International World Wide Web Conference*, May 1996,

URL: http://www5conf.inria.fr/fich_html/papers/P44/Overview.html.

[2] Ramon Caceres, Fred Douglis, Anja Feldmann, Gideon Glass, and Michael Rabinovich. Web proxy caching: The devil is in the details. In *1998 Workshop on Internet Server Performance*, 1998,

URL: http://www.cs.wisc.edu/~cao/WISP98/html-versions/anja/proxim_wisp/index.html.

[3] Adam Dingle and Tomas Partl. Web cache coherence. In *Fifth International WWW Conference*, 1996,

URL: http://sun3.ms.mff.cuni.cz/~dingle/webcoherence.html.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, January 1997.

[5] Sally Floyd. Adaptive web caching. In *2nd International Web caching workshop*, 1996,

URL: http://ircache.nlanr.net/Cache/Workshop97/Papers/Floyd/floyd.ps.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements Of Reusable Object-Oriented Software.* Addison Wesley, 1995.

[7] James Hu. The jaws adaptive web server. Technical report, Washington University of St Louis, 1998,

URL: http://www.cs.wustl.edu/~jxh/research/.

[8] E. Douglas Jensen. Custom end-to-end qos for distributed object systems,
URL: http://www.realtime-os.com/lucent/.

[9] Michael Jungmann and Thomas Paradies. Adaptive hypertext in complex information spaces. Technical report, Technical University of Ilmenau, 1997,
URL: http://nero.prakinf.tu-ilmenau.de/~jungmann/ws-ht97-html/.

[10] Chengjie Liu and Pei Cao. Maintaining strong cache consistency in the world-wide web. In *Proceedings of ICDCS'97*, pages 12–21, May 1997,
URL: http://www.cs.wisc.edu/~cao/papers/icache.html.

[11] P. Lorezetti, L. Rizzo, and L. Vicisano. Replacement policies for a proxy cache. Technical report, Universit di Pisa, 1996,
URL: http://www.iet.unipi.it/~luigi/caching.ps.gz.

[12] Mesaac Makpangou and Eric Brenguier. Relais : un protocole de maintien de cohrence de caches web cooprants. In *Proceedings of the NoTeRe colloquium*, 1997.

[13] Luigi Rizzo. Replacement policies for a proxy cache. Technical report, UCL-CS, 1998,
URL: http://www.iet.unipi.it/~luigi/lrv98.ps.gz.

[14] Margo Seltzer and James Gwertzman. World-wide web cache consistency. Technical report, Harvard University, 1996,
URL: http://www.eecs.harvard.edu/~vino/web/usenix.196/.

[15] Consortium W3,
URL: http://www.w3.org/.

[16] W3C. Jigsaw - web server platform,
URL: http://www.w3.org/Jigsaw/.

[17] W3C. Propagation, caching and replication on the web,
URL: http://www.w3.org/Propagation/.

[18] Duane Wessels. Squid. Technical report, NLANR, 1998,
URL: http://squid.nlanr.net/.

[19] Stephen Williams, Marc Abrams, Charles Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for world wide web documents. In *ACM SIGCOMM 96*, pages 293–305, August 1996,
URL: http://ei.cs.vt.edu/~succeed/96sigcomm/.

[20] Roland P. Wooster and Marc Abrams. Proxy caching that estimates page load delays. In *WWW6 Proceedings*, pages 325–334, April 1997,
URL: http://www.cs.vt.edu/~chitra/docs/www6r/.

[21] Roland Peter Wooster. *Optimizing Response Time, Rather than Hit Rates, of WWW Proxy Caches*. PhD thesis, Faculty of the Virginia Tech, December 1996,

URL: http://scholar.lib.vt.edu/theses/available/etd-34131420119653540/.