TÉCNICO
LISBOA

# VFC for Wikis and Web Caching

**Carlos Manuel Santos Roque**

Dissertation submitted to obtain the Master Degree in
Information Systems and Computer Engineering

## Examination Committee

Chairperson: Prof. Ernesto José Marques Morgado
Supervisor: Prof. Luís Manuel Antunes Veiga
Co-Supervisor: Prof. Paulo Jorge Pires Ferreira
Member of the Committee: Prof. Ricardo Jorge Feliciano Lopes Pereira

**May 2013**

# Acknowledgements

I would like to thank my advisors Prof. Luís Manuel Antunes Veiga and Prof. Paulo Jorge Pires Ferreira for all their help along the course of this thesis, their comments contributed significantly to what this thesis and the paper you are reading paper are today.

I also would like to thank the numerous other people that helped me, not only during this thesis, but also along my graduation, as part of my success is also attributed to them.

Lisboa, June 24, 2013

Carlos Roque

To all that helped me along the way.

# Resumo

Nos sistemas de cache web e de replicação distribuída (wikis) atuais, existe a necessidade de minimizar a quantidade de dados transmitidos, porque no primeiro caso, existe um aumento no tamanho dos objetos web que podem ser armazenados em cache, enquanto que no segundo caso o crescente uso deste tipo de sistema, significa de que os dados ou páginas podem ser alterados e vistos por muitas pessoas ao mesmo tempo. O que significa de que as modificações numa página ou nos dados, tenham de ser propagados para várias pessoas, o que conduz ao aumento da utilização da largura de banda da rede, independentemente do nível de interesse de cada um nestas modificações.

Nesta tese, é descrito como os sistemas atuais de cache web e sistemas de wiki, funcionam e gerem a replicação e é oferecida uma alternativa que adota o algoritmo de Vector-Field Consistency (Consistência Vetorial) e a utilização de algumas preferências do utilizador ao ambiente da web e das wikis.

# Abstract

In today's caching and replicated distributed systems, there is the need to minimize the amount of data transmitted because in the first case, there is an increase in the size of web objects that can be cached, while in the second case, the continuous increase in usage of these systems, makes that a page can be edited and viewed simultaneously by several people. This entails that any modifications to data have to be propagated to a lot of people, and therefore increase the use of the network, regardless of the level of interest each one has on those modifications.

In this paper, we describe how the current web and wiki systems perform caching and manage replication, and offer an alternative approach by adopting Vector-Field Consistency and the use of some preferences of the user, to the web and wiki environment.

# Palavras Chave
# Keywords

## *Palavras Chave*

Cache Web

Consistência Vetorial

Wikis

Preferências do Utilizador

HTTP

Páginas Relacionadas

## *Keywords*

Web Cache

Vector-Field Consistency

Wikis

User Preferences

HTTP

Linked Pages

# Índice

# List of Figures

# List of Tables

# 1 Introduction

In today's Internet environment, there is an increasing number of users geographically dispersed and a large amount of those users uses the web, some of them exclusively, to do their daily work and to look for information or for recreational activities like social networking, online gaming, etc.

This poses many problems to the internet service providers that have almost unlimited requests for providing more bandwidth to their clients, placing the providers in a difficult situation since their network may not be ready for the increase of traffic and the redesign of that network may have prohibitive costs.

So in order to continue satisfying the demands of the users, many ISPs use one or more layers of cache servers in order to reduce the bandwidth required for some of the most common protocols (like HTTP), but due to the dynamic nature of the requirements needed for a cache, those caches have more and more problems to satisfy the client´s needs (that prefer a fast response above all),[1] mainly related to the number of unnecessary pages (frequently dynamic), cached and updated/invalidated and to the fact that many web pages have a lot of uninteresting things to the user, that are highly dependent on the user itself.

The first of these problems is related to the traffic from the caches to the client workstations and from the caches to the original web servers, since 40% of all requested objects from a client are not cacheable without resorting to special strategies(Rabinovich & Spatschek 2002), which makes the traditional caching systems more and more inefficient, given the amount of web content.

Also many objects cached are useless since even if 90% of all web accesses are made to the same set of servers, there is a significant percentage of those pages that are accessed only once(Abdulla 1998), which causes unnecessary traffic when a cache is refreshing pages that are not needed anymore and also an increase in storage use, which is problematic due to the increase in the number of clients of these cache systems.

Besides the general increase in web pages, there is also an increase in the interest and in the number of users of replicated distributed systems like wikis, being that some of the largest wikis have a daily number of users in the order of thousands [2], [3] with an equally high number of edits to their pages.[4]

This results in a problem similar to the one present in web cache, because there is the need of presenting the wiki users with the most updated information (ie. the most recent version

[1] http://www.websiteoptimization.com/speed/tweak/design-factors/
[2] http://www.alexa.com/siteinfo/wikipedia.org+wikia.com+orkut.com+live.com
[3] http://www.google.com/adplanner/static/top1000/
[4] http://s23.org/wikistats/index.php?sort=edits_desc

of a wiki page), while reducing the bandwidth required for transmitting those updates to the client.

This is allied with the fact that most wikis allow their users to maintain a set of preferred or favorite pages and that many wiki systems allow a user or set of users to be responsible for a given set of pages, according to their knowledge about a certain topic, requiring those users to keep an eye on the changes made their set of pages, in order to keep the wiki free of spam posts and as accurate as possible.

## 1.1 Shortcomings of current solutions

In the case of the web caches, most of the current solutions try to propose different algorithms for page replacement, i.e. different ways of specifying when a page should be removed from memory, but many pages are non-cacheable according to the most commonly used consistency method, because of several reasons, including misconfigured servers or web applications that do not use the full knowledge of their domain by providing useful information that the server cannot infer (in the case of strong ETags or last modification dates) and that could be useful, even in the case of dynamic websites like blogs and news sites.

Also since the most used consistency model does not really take precise attention into the user behavior, certain patterns like users with a working set of similar pages and interests are not taken into attention, which could have helped to prevent that pages outside that common working set have replaced pages belonging to that common working set.

This is the case of schools or small enterprises where employees or students have a similar working set of pages, that are related to enterprise related pages or supplier´s pages in the case of employees from an enterprise and from school or course related pages in the case of students.

## 1.2 Proposed Solution

In this paper we develop a system that adapts and extends the Vector-Field Consistency model to semantically enhance the caching of web pages and to show updates to pages interesting to a wiki user.

It is also the purpose of this work to provide a system that reduces the number of cached pages to the ones that are commonly used by a set of users and avoid the caching of unrelated pages or elements to those common to that working set.

In terms of wikis, the purpose of this paper, is to decrease the number of watched pages of a moderator or user, by allowing all of the related or linked pages to be also considered for updating purposes, avoiding the need for a user or moderator, to explicitly and exhaustively include those pages in his working set.

In the subsections below, we provide more detail on the objectives and enumerate the collected functional and non-functional requirements for both the web cache and wiki system.

### 1.2.1 Web Caching

As functional requirements we have the following:

- Usage of the importance of a web page to a user in order to determine if it should be cached and when the cached document is updated/validated;
- Usage of the distance, as the number of links from the bookmarked web pages to a given page, the usage frequency of a page and the time a page has been cached without being requested as a measure of page importance;

The non-functional requirements our cache system should strive to provide include:

**Fast Access** Our cache system should provide a fast access, not only globally but also to the set of pages frequently used by the users of our cache and to the web pages that are related to the ones they like and spend more time in;

**Adaptability** Our cache system, should adapt to new access patterns and not stick to a given set of frequently used pages indefinitely, but rather change at the same rate of the access patterns and users' tastes;

**Simplicity** By building upon Vector-Field Consistency, our cache system will inherit the simplicity of VFC, while providing a powerful web cache scheme.

### 1.2.2 Wiki Replication

On the wiki side, our objective is similar to the web cache objective, so our functional requirements are:

- Usage of the importance of a wiki page to a user in order to determine when the user needs to be notified of changes to that page;
- Usage of the distance, as the number of links, from the watched wiki pages to a given page, the maximum number of updates that a page can have before the user wants to know about them as a measure of wiki page importance;

As for the non-functional requirements we keep the Adaptability and the Simplicity.

### 1.2.3 Browser Extension

As for the browser extensions, whose use is to determine the preferences of the user in a page, our objective is to have an extension that:

- Provides a way for the user to control the consistency zones and specifying their consistency requirements, both for a cache and wiki;
- Propagates changes of the bookmarked pages to the web cache server;

## 1.3 Contribution and Goals

To both of these problems we propose a strategy based on VFC(Santos et al. 2007; Veiga et al. 2010), in order to use a semantic and client oriented approach to the problems related above, that not only takes into account commonly used semantic information such as distance between documents, [5] but also other useful information in order to know when caches should update their documents.

Using this algorithm, we also plan on exploiting the common working set of pages by users, so that our algorithm is particularly useful in the case of a group of users with similar interests, like academic or scholar users, library users, enterprise users and so on. Thus, allowing the common set of pages to be almost always stored in cache, while avoiding the poisoning of the cache with pages not related with that common working set.

In terms of wikis, this will greatly ease the work of a wiki moderator, allowing them to track a small but related set of pages relevant to the domain of their knowledge and be notified of changes not only to these set of pages, but also to the pages that are related to this page set.

For all of these problems we will use the bookmarks of a user and the watched pages of a wiki moderator as a starting point or sources, for Vector-Field Consistency, i.e., where the consistency fields will emanate from.

## 1.4 Document Roadmap

In the following chapters we will:

- Study the related work, related to both the cache and wikis, in Chapter 2, we will also try to frame under taxonomies the current existing systems and their advantages and disadvantages;
- Describe our architecture, for both the cache and wiki in Chapter 3, including a set of related use cases for the cache and wiki system;
- Describe our implementation, for both the cache and wiki, in Chapter 4, discussing also the classes that are used to implement VFC in both the cache and wiki system;
- Present the tests done to both the cache and wiki, in Chapter 5, including the comparison to the other caches, with focus on the goals of our work;
- Finally we will close with a conclusion, describing the work done, future work and the advantages and disadvantages of the presented solution.

[5]Measured as the length of the shortest chain of links leading to them

# 2 Related Work

In this part we are going to analyze web caches and wiki systems, in the context of web caches we are going to analyze some important characteristics of any web cache, on the wiki systems we are going to analyze the types of wikis, the types of users and the architecture of wiki systems.

Finally we are going to discuss some concepts about Vector-Field consistency, in order to provide the reader with information about what is Vector-Field consistency and the current uses for Vector-Field consistency.

## *2.1 Web Caching*

In web caching we are to talk about:

- The structure or architecture of a web cache;
- The two different models of a web proxy;
- How caches keep the content fresh and consistent with the web server hosting them;
- How caches decide what to cache and what to replace in the lack of space for more documents;
- Classify some of the most used caches on the above studied parameters;

### 2.1.1 Web Cache Architecture

The architecture of a distributed web cache can be based on four classical approaches, a hierarchical architecture, a cooperative distributed architecture and a hybrid architecture(Wang 1999)(Rodriguez et al. 2001).

#### 2.1.1.1 Centralized Architecture

This architecture (Fig. 2.1) is the simplest one of all four because it consists of only one cache server, that connects to a set of clients (users) and that makes requests to a web server hosting content whenever a client makes a request.

This architecture was the first one to be used and in spite of the big disadvantage that it is limited in the number of requests that it can serve and that it is a single point of failure, it is still used on very simple cases, like home web cache servers or in networks that serve a small set of clients.

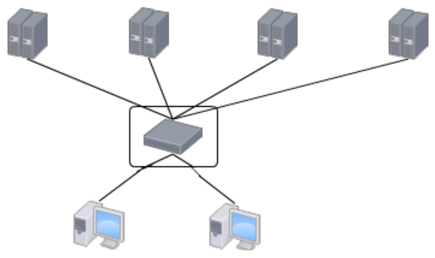


Figure 2.1: Diagram of a centralized cache

#### 2.1.1.2 Hierarchical Architecture

The second approach (Fig. 2.2) distributes the cache servers in a pyramidal tree, where the bottom servers are contacted by the clients and the upper or root server is the only one allowed to connect to web servers hosting content.

In this approach, when a client makes a request to the local server, the server forwards the request upstream in the hierarchy until it is either satisfied by some cache server or it reaches the top of the tree. When it does the root server, connects to the web server that can satisfy the request and distributes the response to the caches below it, so that the document is available to the lower levels.

The Adaptive Web Caching or Top-10 prefetch(Markatos & Chronaki 1998) is one example of a cache system that uses the hierarchical model.

One advantage of this distributed architecture is that it can save bandwidth, because documents can be served more quickly since there is some probability that, the document is on some cache along the hierarchy, so that only a few client requests have to be sent to the web servers hosting the content.

On the downside, there are delays associated with each caching level, because for each new level, there is another set of caches processing requests that cause delays; and since a document travels down in the hierarchy to the web caches the same document is stored on multiple caches wasting possible memory or disk space for some pages.

#### 2.1.1.3 Cooperative Distributed Architecture

In the third approach (Fig. 2.3), there is no hierarchy, but a set of cache servers receiving connections from clients, that are connected among them, so that when a request comes to a server, that server checks if any other neighbor server containing the document and if it is retrieves it from there instead of making a connection to the original web server.

The approaches based on this method, can use:

- A hash of the document URL to know which server contains or which server should contain the document served by a given URL;
- A central server that decides which server keeps or should keep a given document;
- A cache routing table that is multicasted by a given cache server that specifies the URL´s that the cache server is responsible for.

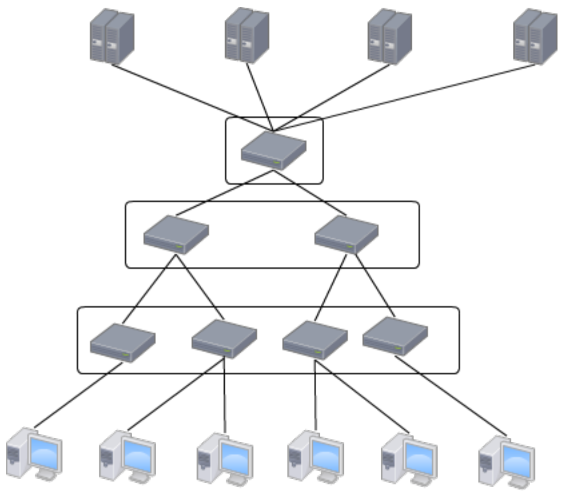


Figure 2.2: Diagram of an hierarchical cache

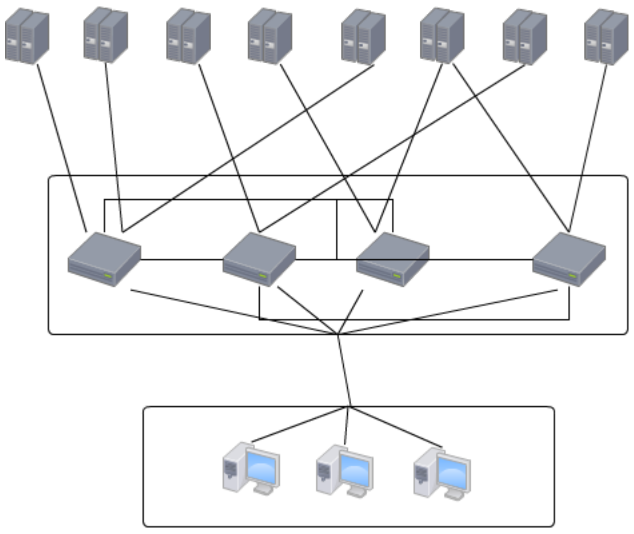


Figure 2.3: Diagram of a cooperative distributed cache

The Cache Array Routing Protocol(Valloppillil & Ross 1998) is an example of a protocol based on the cooperative distributed protocol.

This method has the advantage that it has very low latency compared to the second one and low disk usage, because each document is stored only once leading to a better document distribution.

The disadvantage is that the used bandwidth is more than with the first scheme, because each cache has to coordinate itself with other caches in order to know where a page is or where a page should be, for example, in the Cache Array Routing Protocol.

#### 2.1.1.4 Hybrid Architecture

In the fourth approach (Fig.2.4), there can be a tree of cache servers, just like in the second approach but the caches in the same level are connected to each other like the second approach

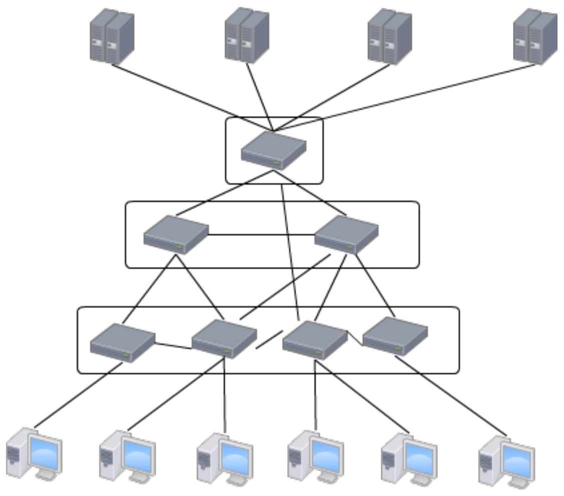


Figure 2.4: Diagram of a hybrid cache

and upon receiving a request a cache server contacts first the servers at the same level as it and only if the requested document is not in those servers, the request is sent up in the hierarchy. The same process is repeated, until either the request is satisfied or the root cache server is contacted, and in that case it happens the same as in the hierarchical scheme.

In a variation of this protocol, when deciding the neighbor caches to make a request, an requesting cache can choose only the neighbors with a round trip time below a certain threshold, even if another neighbor has the document.

The Internet Cache Protocol(Wessels & Claffy 1997) used by Squid [1] among others, is an example of a protocol based on a hybrid scheme.

What makes this architecture interesting is the fact that if planned in a correct way, then the disadvantages of the hierarchical and cooperative distributed schemes can be mostly mitigated and the advantages combined.

#### 2.1.1.5 Comparison of Architectures

After having analyze all of the cache architectures, we will compare them according to the number of servers that can be involved and the topology of the architecture.

- Centralized Architecture:

  **Number of servers** 1;

  **Topology** Single;

- Hierarchical Architecture:

  **Number of servers** Any;

  **Topology** Vertical;

[1]Squid is an open source proxy/cache server.

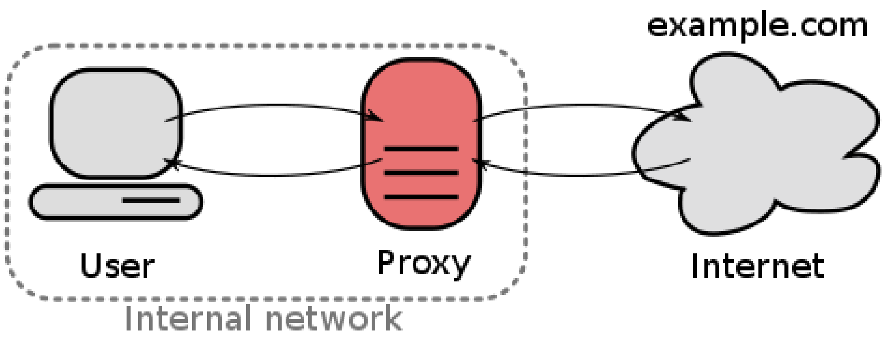


Figure 2.5: Diagram of the forward proxy model

- Cooperative Distributed Architecture:

  **Number of servers** Any;

  **Topology** Horizontal;

- Hybrid Architecture:

  **Number of servers** Any;

  **Topology** Both vertical and horizontal;

### 2.1.2 Models of Web Proxies

In the world of web proxies, there are two distinct models for web proxies(2001)(Davison 2001), that influence the chosen consistency and algorithms.

#### 2.1.2.1 Forward Proxy

The first model of proxy and the most used is the forward proxy model(Fig.2.5), where a proxy is sitting directly in front of a client and acts as an intermediary between a client and a server and hides the client from the server enabling the client to be directly served by the proxy if it has caching capabilities.

#### 2.1.2.2 Reverse Proxy

The second model of proxy is the reverse proxy model(Fig.2.6) and is associated with a web server and is usually controlled by the same entity or someone on behalf of it. So in this model the proxy appears as a normal web server to clients, that upon a request may connect to a back-server to handle the request or if the request is in cache to directly handle the request.

This type of model, allows for load-balancing, since a reverse proxy may choose an different back-server depending on its location, load and type of client, allowing for differentiation according to whether the client is using a mobile device or not; it also allows a provider to exploit the location of a client, in order to redirect the request to the proxy that is closer to the client.

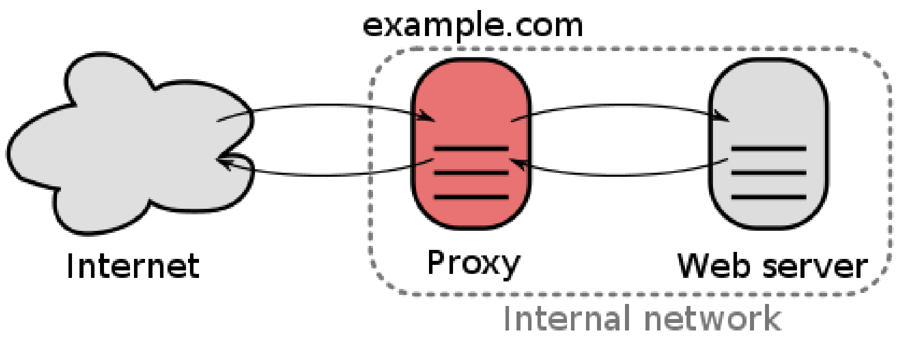


Figure 2.6: Diagram of the reverse proxy model

This model also allows for blocking or filtering of requests in case of attacks to a determined web server and depending of the proxy software to distribute the processing of a web application, allowing some of the processing to happen on the actual proxy.

This types of proxies are also associated with content distribution networks, that use reverse proxies to directly serve static data.

#### 2.1.2.3 Comparison of Proxy Models

After having studied the two possible proxy models, we will compare them, according to the most common usage scenario, awareness to the technology behind a page (for example, if it is aware or not if a page is written in ASP, J2EE or other server language or framework) and special needs (if there are special needs in terms of server or installation for the given proxy model).

- Forward Proxy:

  **Usage scenario** Near the client, by an ISP or a network provider;

  **Server awareness** No, since the cache can only see already processed pages;

  **Special needs** None.

- Reverse Proxy:

  **Usage scenario** Near the server, by a cloud network provider and page host company or a content distribution network (commonly refered as CDN);

  **Server awareness** Yes, since the cache can help the server to further process the requests, allowing the cache to finish the final page (for example, a cache can store static templates and only ask a server for the unique information to fill in those templates);

  **Special needs** Depends on whether or not the cache is aware of the technology behind the server.

### 2.1.3 Cache Consistency

Since a cache server must keep its cached content as fresh as possible, while reducing the traffic between it and the web server where the cached documents belong, it must decide when to update, how to update and if there is a need or an expected gain to store a given document in cache, so that the client sees a page with the minimum staleness possible, without having to download the page from the original server.

So, for this problem of web cache consistency two classic solutions exist, consistency by validation and consistency by invalidation. There is also another subdivision into strong consistency and weak consistency, where the first one implements a strong notion of consistency, where the document is always fresh, while the second one only grants that the document is possibly unchanged in the server and therefore valid(Rabinovich & Spatschek 2002)(Cao & Oezsu 2002).

#### 2.1.3.1 Consistency by Validation

The first and most used strategy for consistency on the cached documents is consistency by validation, in that the cache servers are responsible for contacting the web servers hosting the documents they contain, when a specified condition is reached.

The most used scheme based on this strategy is the one used by the own HTTP 1.1 protocol(Fielding et al. 1999) where there is a set of HTTP headers that can be set by either the client in the request or by the server in the response and that control the caching process and when the cache should validate the stored documents.

So for HTTP, a cache can send a response to a request if that response was validated with the server, if it is valid according to the parameters set by the origin server or the client, if it contains a warning header (if it is not valid, according to the client or server parameters) or if it is an error message.

Also according to the HTTP protocol a cache must obey to the following headers, present inside the request:

**Cache-control** This is a header that a server hosting content or client can set in the request or answer and that directly applies to the cache allowing a explicit cache control:

- **no-cache** This option tells the cache not to store anything and instructs the cache to simply route the request to the server;
- **no-store** This options tells the cache not to store the page on disk cache (if that feature is available) and to try to remove the page from memory as soon as possible;
- **max-age** The maximum amount of time that a document can be served from a cache without a re-validation specified in seconds, this is only available in a request;
- **max-stale** The amount of time that a stale document may be served to a client without a re-validation with the server hosting the document specified in seconds, this is only available in a request;
- **no-transform** The document can not be changed by the cache;
- **only-with-cached** The cache must answer with what has available or return a error message if it has not anything available;

- **must-revalidate** The cache must re-validate the document when the document becomes invalid, even if it is configured to ignore the staleness value and return a stale response, this is only available in a response;
- **public** The document may be cached by any cache server, including a shared cache;
- **private** The document may only be cached by a non-shared cache (ie, one that is available in the browser).

**Expires** Specifies the date when the document received in the response, becomes stale and it is specified using RFC 1123 date format;

**Date** Specifies the date at which the document was generated by the server;

**Age** The age of the response if it comes from another cache.

In order to determine when a document becomes stale, HTTP 1.1 uses the following formula if either the max-age or the expires header is set:

```
received_age = max(max(0, received_response_time - date_value), age_value)
initial_age = received_age + (received_response_time - request_time)
current_age = initial_age + (now - received_response_time)
lifetime = max(max_age_header_value, expires_header_value - date_header_value)
stale_age = initial_age + lifetime
```

In this scenario, a page must be checked for modifications when the calculated stale_age is bigger than the calculated lifetime. If neither header is set, the server is free to use another heuristic, that is usually based on the fact that unchanged documents tend to keep unchanged for long periods of time(Cate 1992).

While verifying if a request may use the cached response, the HTTP protocol, specifies the Vary tag, so that the web caches, may use only certain headers to verify if two requests are identical or not.

Also, a response may contain ETag values or modification dates, to allow the document to be validated by the server, so that when documents need to be checked for a modification, those values are used to create a conditional request which avoids the transmission of a full response, using only a response code of 304 (not modified), if the values match.

If the origin web server cannot be contacted, the HTTP protocol specifies that the web cache may serve the cached response, if it adds a warning header to the response in order to notify the client that the page may be stale, therefore allowing the usage of stale pages instead of failing when a web server is offline.

The disadvantages of HTTP based consistency are that requests using the same URL and method may return different responses that caches might not perceive because caches in HTTP cache only the request URL and the response(Mogul 2004), and that the cache may make unnecessary requests to a server, if a proper max age or expire date is not set by the web server hosting the requested document.

Also, the HTTP protocol has not a way of specifying if a given document is dynamic or not, in which cases caches to rely on heuristics such as the presence of cookies in the request or the URL of the request in order to try to guess such information and prevent the caching of those pages, even if by default HTTP allows the caching of dynamic documents, given that the cache control headers allow it.

#### 2.1.3.2 Consistency by Invalidation

The second strategy of consistency on the cached documents is consistency by invalidation(Howard et al. 1988)(Lee et al. 2009). Instead of caches contacting a web server, when a document becomes stale, the web servers take note of the web caches that request the documents and, on the first contact, piggyback a callback with the response. So that when a page becomes invalid because it changed, a web server can simply transverse the list of callbacks and notify the caches where the document has become stale and therefore is invalid.

While this approach potentially reduces the staleness of a document, it also forces the server to maintain state and therefore one must change the server code.

**2.1.3.2.1 Version 2.1** Another variation of the cache consistency based on invalidation is the use of leases(Gray & Cheriton 1989) in that a cache obtains a lease for each page it receives from a server and that grants the cache a given time where the obtained document is valid, except when it is sent a specific invalidation from the server for that object.

This solves the fact that a client may fail and also improves the basic invalidation protocol because the server only needs to notify clients that hold valid leases in the case of a document change, which reduces the amount of state in the server.

Also, if a cache is unable to contact with a server the documents it holds from it are potentially out of date and therefore the clients can be warned that they are receiving a copy that is out of date, using the standard HTTP protocol.

**2.1.3.2.2 Version 2.2** On the other side it was later observed that there was a potential problem in the lease protocol as described above(Yin et al. 1998), in that if a set of objects has to be used more frequently than its lease time then advantages of leases are lost.

Therefore a solution was proposed to that problem using two types of leases.

The first type is granted to single documents and are called object leases, while the second type is granted to a set of documents from the same server and are called volume leases.

So that the object leases can be long and the volume caches are short enough to allow the servers to write an object if they need to, for example, if a server suddenly gets down and then reboots, then the server only has to wait for the longest volume lease granted.

Using this modification when a client requests a document, the cache server checks both the object lease and the volume lease of that document and if they are both valid, it returns the document in cache to the client. But if any of them is invalid, the cache server sends a lease renewal to the web server and if the document(s) has(have) changed, the web server piggybacks the change delta or the current document(s) in the lease renewal response.

When a server wants to modify a document, it sends invalidations to the caches that have valid leases, either object or volume leases and only modifies a document when all responses are received or when the granted leases expire, if some cache server cannot be contacted.

**2.1.3.2.3 Version 3** In the paper by Yu(Yu et al. 1999), it is described another approach (piggyback) to cache consistency based on invalidation, where there is a cache architecture based on a hierarchy, where each parent cache uses multicast, to communicate with their children. There is also a connection between each web server and the top level cache, that works in a similar way to the top level caches of the hierarchical architecture.

So in this scheme, a parent cache (or web server) sends a periodic message (heartbeat) to all of its children (or root caches that requested any of its documents) and piggybacks all the invalid (or changed) documents in that message, so that the children invalidate the set of pages piggybacked in the message. Also, if for a given time T (an T corresponding to 5 samples was used by the authors) there is no message received from the parent cache, the pages belonging to that parent in the children are automatically invalidated.

When a page is requested by a client, the request is sent up in the hierarchy until it reaches a cache that has a valid version of the page or the original server if none of the caches has a valid version of the page.

#### 2.1.3.3 Analysis

Based on the two most common used consistency schemes used on the web, it is clearly that the one that is mostly used is the consistency by validation method, but given the rise in the use of CDN´s[2] and cloud computing under a http server, the consistency by invalidation might find a use, since those http servers are typically in control of CDN and cloud providers, that may change the software and adapt the application code to the infrastructure, in order to mitigate possible problems with reliability and fault-tolerance that may affect these systems.

Also, there is a big disadvantage of these systems that is that consistency by invalidation, tends to increase the cache and server state pollution rate, since an object may be requested once at distance intervals of time.

In terms of proxy models, while consistency by validation can be used by reverse and forward proxies, consistency by invalidation must be used by reverse proxies only.

In terms of offline availability the consistency by validation wins, because the scheme, tends to be less dependent of an origin web server than the consistency by invalidation. Which gives the following comparative information:

- Consistency by Validation:

  **Popularity** High;

  **Usage scenarios** Both, forward proxies and reverse proxies;

  **Server dependency** None;

  **Client awareness** High;

- Consistency by Invalidation:

  **Popularity** Low;

[2]Content Distribution Networks

**Usage scenarios** Reverse proxies only;

**Server dependency** High;

**Client awareness** Low;

### 2.1.4 Cache Replacement Strategies

Because a cache cannot hold all valid requested documents on cache (even if it uses a disk cache), there is a need of a cache replacement strategy or algorithm, that determines when a given document is removed from cache or when a document may be replaced by other.

Since there are many metrics by which these strategies may be classified, we shall adopt the taxonomies commonly used(Podlipnig & Böszörmenyi 2003), that classify the strategies into:

**Recency based strategies** These strategies use the age of the document, to decide if it should be replaced/deleted or not, one advantage of these strategies is that the document age is easy to determine, while one disadvantage is that an an old document that is potentially replaceable according to these strategies, can be extremely popular;

**Frequency based strategies** These strategies use the usage frequency of a document to decide if it should be replaced/deleted or not, an advantage is that it considers documents regardless of their age and therefore avoid one disadvantage of recency based strategies, but since it does not take into account time, it can happen that an object that was extremely popular in the past is never removed;

**Frequency-Recency based strategies** These strategies use both the age and frequency of the document to decide if it should be replaced/deleted or not and can have the advantages of the two methods above, without any of the disadvantages if they are well combined, because they can determine if an old document is still popular and whether a popular document in the past is still a popular document;

**Function based strategies** These strategies use a formula that may use a certain amount of measurable features of the documents in order to decide if the document should be replaced/deleted or not, these strategies by being based on a formula instead of a data structure like the common implementations of the other three strategies above can adapt more to a usage pattern change and are easier to implement;

**Random strategies** These strategies use a random approach when deciding if a certain document should be replaced/deleted or not and have the advantage of simplicity.

**Machine Learning based strategies** These strategies use techniques from machine learning to either improve traditional algorithms or replace them and are commonly divided into two phases and learning phase and a running phase, where in the first phase the algorithms are trained and in the second phase, the algorithms apply the acquired knowledge.

So in the following subsections, I describe with some detail some algorithms based on the above strategies that are simple and widely used in existing cache systems ordered by strategy family as specified above.

#### 2.1.4.1 Recency Based Strategies

**2.1.4.1.1 Pyramidal Selection Scheme** This is a strategy(Aggarwal et al. 1999) based on recency that uses some knowledge about the document size when deciding which document to replace, in such a way that the authors describe their strategy as a way to solve the knapsack problem.

The authors consider a measure of dynamic frequency as $\frac{1}{\Delta T_{ik}}$ where $\Delta T_{ik}$ is the number of accesses to other documents since the last access to a certain document $i$, so that the aim is to minimize the sum of the dynamic frequencies for the documents that are going to be remove while removing enough documents to insert the new one.

So the authors consider initially an approach where the documents are ordered by a non-decreasing order of $S_i \times \Delta T_{ik}$ and then the objects higher in the list are chosen to be removed until there is enough space on cache.

But since that approach is expensive in terms of processing time, the authors develop an alternative that is PSS, in which the documents are classified according to their size and placed into a structure similar to a pyramid, in that the cache space is divided into $\lceil \log (M+1) \rceil$ where $M$ is the cache memory size, and where each group has objects sized in between $2^{n-1}$ and $2^i - 1$ and is managed as a Least Recently Used List.

So, that when the cache needs to remove a document or documents, the values of $S_i \times \Delta T_{ik}$ from the least used objects of each group are compared.

The authors also develop a strategy that takes into account the distance from the cache to the web server hosting the documents and one that takes into account the document expiration time.

The result is a method based on recency that takes into account several other metrics like size (and expiration time or cost) and creates an algorithm that is easy to implement and does not require a lot of computational power.

**2.1.4.1.2 LRU-Min** This strategy[3](Abrams et al. 1995) is based on the Least Recently Used algorithm and modifies it in order to include a parameter T that is initially assign to the size of the object to remove and an auxiliary list L, and proceeds as following:

1. Add to L all documents equal or larger than the value T, given that L may be empty after this step;
2. Using LRU remove from cache all objects in subset L, until there is either space for the new object or the list L is empty;
3. If there still is not enough space to add the new object, set T to it´s half and repeat step one.

[3]Least Recently Used - Minimal

#### 2.1.4.2 Frequency Based Strategies

**2.1.4.2.1 LFU-DA** This strategy[4](Arlitt et al. 2000) is based on frequency and tries to reduce some disadvantages associated with a pure frequency based strategy, by introducing a parameter K that is used to reduce the frequency count of all object by two, if the average frequency count is bigger than that parameter. It also has another parameter specifies the maximum frequency count that objects may have.

From these parameters, the first one is calculated in a dynamic way, using the following formula:

$$K_i = C_i * F_i + L$$

In this formula the L parameter has the following formula:

$$L_i = \begin{cases} 0, & \text{if object is new} \\ minK_{i-1}, & \text{otherwise} \end{cases}$$

The value $C_i$ is an additional parameter used to tune the formula, if wished (the authors have used one as the value of the parameter).

**2.1.4.2.2 $\alpha$-aging** This strategy(Zhang et al. 1999) uses the notion of virtual ticks or periods that may have a varying or fixed value, that are used to apply an aging function to every object in cache. Therefore this strategy has the notion of two stages.

In the first stage, all of the objects in cache are incremented by one every time they are hit (frequency count), according to the normal LFU algorithm.

In the second stage, that happens when there is a tick, the strategy, uses the following aging function in all objects in cache:

$$K_i = \alpha * K_{i-1}$$

Where $\alpha$ is a tuneable value between zero and one and that makes the algorithm ranging from a LRU to a LFU behavior.

Finally in the case of a tie, the strategy uses LRU in order to decide which object to remove.

#### 2.1.4.3 Frequency-Recency Based Strategies

**2.1.4.3.1 LRU*** LRU*(Chang & McGregor 1999) is a strategy that combines both frequency and recency, into a simple scheme, that keeps all documents indexed in a LRU list and when a document in cache is requested by a client, it is moved to the start of the LRU list and the hit counter is incremented by one.

When there is a need for removing some document, the hit counter of the last recently used document (ie. the one at the back of the LRU list) is checked and if it is zero, the document is

[4]Least Frequently Used - Dynamic Aging

removed. If not, the counter is decreased by one and placed at the start of the LRU list. This is done until there is enough space for placing the new document. Also, in order to prevent documents from having a large hit count, the authors proposed that the maximum hit count is five.

This scheme is simple to implement, even if it does not take into account the size of the document that is removed and the fact that a large document may be a better option for removal than a lot of smaller documents.

**2.1.4.3.2 Hyper-G** This strategy(Abrams et al. 1996), combines LRU, LFU and the size of objects in order to provide alternative mechanisms in the event of a tie between objects to remove. So this strategy uses LFU first in an attempt to remove objects, followed by LRU on the tie objects and finally if there is still a tie then the largest object is removed. Until there is enough space to place the new object.

#### 2.1.4.4 Function Based Strategies

**2.1.4.4.1 GD-Size** The GD-Size[5](Cao & Irani 1997) strategy is a function based strategy where each document in cache contains a value calculated using:

$$K_i = \frac{C_i}{S_i} + L$$

Where $C_i$ is the cost of retrieving the document from the server hosting it, $S_i$ is the size of the document and $L$ is an aging factor and is initially zero. When the values are recalculated the value of L is the minimum value of $K_i$ for all the documents.

Then the objects with the lowest $k_i$ value are removed, until there is enough space to store the new document in cache.

**2.1.4.4.2 GDSF** The GDSF[6](Arlitt et al. 2000) strategy is a function based strategy where each document in cache contains a value calculated using the following formula:

$$K_i = F_i \times \frac{C_i}{S_i} + L$$

In that $F_i$ is the frequency count (ie. the number of hits that the document had), $C_i$ is the cost of retrieving the document from the server hosting it, $S_i$ is the size of the document and $L$ is an aging factor and is initially zero. When the values are recalculated the value of L is the minimum value of $K_i$ for all the documents.

This strategy is based on the GD-Size strategy, but it adds a frequency count to it, so that documents that have a high frequency in the past can have some advantage to the documents that have lower frequency. Then, when there is a need to remove a document, the document with the lowest value of $K_i$ is removed.

This strategy, unlike the other two, does not use a specific data structure or some combination of frequency/recency or recency/size and therefore can better adapt to a dynamic

[5]GreedyDual-Size
[6]GreedyDual-Size with Frequency

environment. But the cost to retrieve a document cannot be determined exactly (due to the fact that a document may use different paths from the origin server to the cache server) and the estimated value is very hard to compute and requires constant adaptation.

#### 2.1.4.5 Random Based Strategies

**2.1.4.5.1 Harmonic** This strategy(Hosseini-Khayat 1998) uses a weight random function, in which the probability of an object being selected for removal is equal to the inverse, of its cost.

#### 2.1.4.6 Machine Learning Based Strategies

**2.1.4.6.1 NNPCR** This is a strategy[7](Cobb & ElAarag 2008) based on machine learning specifically on the usage of a neural network as a way to give a score of the cacheability of a given page, in the interval of 0 and 1. The input features of the neural network are the recency, frequency and size of the page.

In the learning phase, the strategy uses an hour long log in order to train the neural network and a second hour long log from a different day in order to verify the neural network results.

**2.1.4.6.2 SVM-GDSF** This strategy[8](Qian et al. 2012) uses a Support Vector Machine together with the GDSF strategy, explained above in order to improve the byte hit-ratio of the common GDSF strategy.

In this strategy, the Support Vector Machine uses as inputs the recency of the object, the frequency of the object, given both of the two parameters are based on a sliding window (thirty minutes was used by the authors), then the global frequency of the object, the retrieve time of the object, the size of the object and the type of the object (whether it is an image, HTML file, etc.) and produces a binary response in order to indicate if the object is going to be reused or not.

Given this number, the strategy changes the regular GDSF in the following way, with $W_i$ being the injected parameter:

$$K_i = F_i \times \frac{C_i}{S_i} + L + W_i \tag{2.1}$$

**2.1.4.6.3 C4.5-GDS** This strategy[9](Qian et al. 2012) uses a C4.5 decision tree, in order to improve the greedy dual-size strategy, in a different way than the GDSF algorithm mentioned above. This strategy uses a C4.5 decision tree that has as inputs the recency of the object, the frequency of the object, given both of the two parameters are based on a sliding window (thirty minutes was used by the authors), then the global frequency of the object, the retrieve time of the object, the size of the object and the type of the object (whether it is an image, HTML file, etc.) and produces a number between zero and one, that is the probability of an object being reused.

[7]Neural Network Proxy Cache Replacement
[8]Support Vector Machine Greedy Dual-Size with Frequency
[9]C4.5 - Greedy Dual-Size

This number ($P_i$) is calculated in an accumulated way into a parameter $W_i$ , at each request as follows:

$$W_i = \begin{cases} 0, & \text{if object is new} \\ W_{i_o}ld + P_i, & \text{otherwise} \end{cases}$$

With this number being injected in GDS in the following way:

$$K_i = W_i \times \frac{C_i}{S_i} + L$$

#### 2.1.4.7 Analysis

After studying some cache replacement algorithms and following the results of numerous comparative tests between them(Romano & ElAarag 2008), with the traditional algorithms being studied more, one can argue that function based strategies are the ones that have the best results, even if the recent algorithms of machine learning(Qian et al. 2012) may improve upon them.

Also given the rise of large objects being transmitted thought the web like movie or audio files[10], it may be helpful to have several caching areas each one using a different strategy, in order to further differentiate between object types and therefore sizes, allowing for objects to be split across different memory areas that have algorithms appropriate to the type of objects cached. Finally, we present a comparison of all replacement strategies in terms of used parameters, relative complexibility of implementation and resource usage.

- Pyramidal Selecttion Scheme (PSS):

  **Used parameters** Age and size;

  **Relative complexibility** Medium-Low;

  **Relative resource usage** Low;

- LRU-Min:

  **Used parameters** Age and size;

  **Relative complexibility** Low;

  **Relative resource usage** Low;

- LFU-DA:

  **Used parameters** Age and frequency;

  **Relative complexibility** Low;

  **Relative resource usage** Low;

- $\alpha$-aging:

[10] http://www.websiteoptimization.com/speed/tweak/average-web-page/

**Used parameters** Age and size;
**Relative complexibility** Low;
**Relative resource usage** Low;

- LRU*:

  **Used parameters** Age and frequency;
  **Relative complexibility** Low;
  **Relative resource usage** Low;

- Hyper-G:

  **Used parameters** Age, size and frequency;
  **Relative complexibility** Low;
  **Relative resource usage** Low;

- GDSF:

  **Used parameters** Retrieval cost, size, age and frequency;
  **Relative complexibility** Low;
  **Relative resource usage** Low;

- Harmonic:

  **Used parameters** Retrieval cost;
  **Relative complexibility** Low;
  **Relative resource usage** Low;

- NNPCR:

  **Used parameters** Retrieval cost, size and age;
  **Relative complexibility** Medium-High;
  **Relative resource usage** Medium-High;

- SVM-GDSF:

  **Used parameters** Retrieval cost, size and age, type and frequency;
  **Relative complexibility** Medium-High;
  **Relative resource usage** Medium-High;

- C4.4-GDS:

  **Used parameters** Age, frequency, size, type and cost;
  **Relative complexibility** High;
  **Relative resource usage** Medium-High;

### 2.1.5 Commercial Cache Servers

In this section we describe the available cache server software in respect with the parameters studied above, specifically the consistency method they use, the architecture they support, the page replacement algorithms they support, together with the supported HTTP protocol version and some other features.

- Squid:
  - Consistency By Validation (HTTP);
  - LRU or GDSF or LFU-DA;
  - Single or Hybrid (ICP protocol);
  - Forward or Reverse Proxy;
  - HTTP 1.0 in version <3.X and 1.1 incomplete in version 3.X.
- Polipo:
  - Consistency By Validation (HTTP);
  - LRU;
  - Single;
  - Forward Proxy;
  - HTTP 1.1.
- Nginx:
  - Consistency By Validation (HTTP);
  - LRU;
  - Single;
  - Forward or Reverse Proxy;
  - HTTP 1.1.
- Varnish:
  - Any (Consistency by Validation (HTTP) by default);
  - LRU;
  - Any;
  - Reverse Proxy;
  - HTTP 1.1.

#### 2.1.5.1 Squid

Squid[11] is a proxy/cache server that supports HTTP 1.0 client and server, GOPHER and FTP server connections, it is also available for Unix and Windows and supports the use of either

[11] http://www.squid-cache.org

IPv4 or IPv6 protocol. It supports the use of asynchronous connections for handling client requests, so that it can support multiple clients in a scalable and efficient way.

In terms of architecture it can be used as a single proxy/cache server or using ICP protocol in a hybrid distributed architecture, where one can configure if there is a hierarchy or not.

For a cache replacement strategy, Squid can use either a LRU, GDSF or LFU-DA strategy, according to the Squid version and configuration. Since it is a HTTP 1.0 based cache, it uses the consistency method of the protocol, that is based on validation.

Squid is also able to cache Cookies and use them when validating documents, even if those cookies are not distributed to the client because of the requirements of HTTP protocol.

Finally and since Squid can be run as a reverse web cache (ie. a web cache that is on the server side and is used to load balance the connections from the server), some of its clients that use it as a reverse web cache are Wikipedia and Flickr that use it to serve their multiple clients and to serve the images (in the case of Flickr).

Given that Squid only supports HTTP 1.0, that is one disadvantage of it compared to other caches. Also since Squid has so many features it is hard for someone, to adapt it for a particular environment or to change its code, in spite that for the first case, its rich configurations and the availability of many books about it help to mitigate the problems.

It is also important to mention, beginning with version 3, Squid also supports some aspects of HTTP 1.1, with exception of request chunks.

#### 2.1.5.2 Polipo

Polipo[12] is a small proxy/cache server that unlike Squid, fully supports HTTP 1.1 clients and servers, it is available for Unix platforms and also supports either IPv4 or IPv6 protocol. It also supports the use of asynchronous connections for handling client requests, just like Squid.

In terms of architecture it is usable as a single and portable proxy/cache server, in the sense that one user may copy it to several machines and put it to work, without needing to configure it, although he can do so, if he wishes. For a cache replacement strategy it uses LRU and also like Squid, uses the HTTP protocol based validation consistency, except that it always uses HTTP 1.1.

Finally, since Polipo is able to use the SOCKS protocol, it can be used together with a program like Tor[13] and grant anonymity to its clients on the web. Also since it is much smaller than Squid, its code can be easily changed by a user that wants to add new features, it can also serve as a bridge between IPv4 and IPv6 or vice versa.

Also since it is made to be very small, this proxy is embedded in some routers and network equipment.

[12]http://www.pps.jussieu.fr/~jch/software/polipo

[13]Tor is a secure network, that routes the packages through several anonymous servers in order to hide its users identity and prevent them from being localized

#### 2.1.5.3 Nginx

Nginx[14] is a proxy/cache server with reverse capabilities that supports HTTP 1.1 clients and servers, but also acts as a email server and supports IPv6 and IPv4 and uses an asynchronous approach to handle requests.

In terms of architecture, is it usable as a single proxy/cache server and uses LRU for cache replacement strategy and the HTTP protocol based validation consistency, but it is also capable of running CGI scripts and therefore act as a web server that generates content and by using that is also capable of serving dynamic content, unlike the other two options above. It is also able to use filters to transform the documents it serves, for purposes such as document compression, dynamic range requests, image transformation and XSLT transformations.

It also supports the use of SSL and HTTPS in connections in order to securely connect to clients and servers, fast reconfiguration without the need to stop the server, the use of a rewrite module in a similar way that one does in a web server like Apache.

It is also capable of serving pages and to behave as a HTTP server, that includes support for fast CGI and some other technologies in order to integrate scripts written in some languages, so that they can serve pages from the server.

#### 2.1.5.4 Varnish

Varnish[15] is a proxy/cache server that is made specifically to be used as a reverse proxy and cache content directly from a specific set of web servers, in order to respond to multiple clients.

Because of this, it is made to be extremely responsive and at the same time extensive in an unusual way, since it allows the writing of scripts in its own domain specific language, that is called VCL (Varnish configuration Language) and that allows for an customer to override or add additional behavior when certain events like when a new request arrives or a response is received (or retrieved). These scripts are then translated to C by Varnish, compiled and inserted as modules.

Also, it allows the usage of custom modules written directly in C (or a compatible language), that can be used to extend the operations available to an VCL script, effectively allowing the cache to be used with a consistency by invalidation protocol.

## 2.2 Wiki Systems

In this chapter, we describe some categories that are used to classify the relevant aspects of a wiki and to classify the types of editors in a wiki and also some of the architectures used by wikis.

### 2.2.1 Classifying Wikis

There are mainly four types of wikis:

[14] `http://nginx.org/en`
[15] `https://www.varnish-cache.org/`

**Enterprise wikis** That are used within an enterprise(Poole & Grudin 2010) and are subdivided in:

- **Project or Group wikis** That are used to maintain information relevant to a group or project, like project documentation or instructions to new group or project members;
- **Single-user wikis** That hold information relevant to a person within a enterprise or information that the owner wishes to share with others;
- **Enterprise-wide Wikis or encyclopedias** That hold information about the enterprise as a whole and are meant as internal encyclopedias.

**Educational wikis** These wikis(Forte & Bruckman 2007) are used to share knowledge or information relative to a course and are subdivided in:

- **Course wikis** These wikis are created within an educational institution for supporting a given subject;
- **Knowledge wikis** These wikis are meant for teachers that need to share information and subject material between them and may or may not be associated with an education institution;
- **Subject wikis** These wikis are meant to contain books, materials or exercises about a given subject[16]

**Public wikis** This wikis are public wikis and can be accessed and/or edited by anyone and can be divided in two groups:

- **General-purpose public wikis** These wikis contain information about several issues; [17]
- **Specific-purpose public wikis** These wikis are dedicated to one specific purpose or issue. [18]

**Personal wikis** These wikis are meant to replace personal websites.

### 2.2.2 Classifying Wiki Users

On the subject of the editor types, there are two main types of wiki editors, the first type is the occasional editor, that either only edits or creates something about what he or she knows or only changes one section or small paragraphs at once and therefore finds attractive the fact that a page is easily editable.

The other type of editor is the professional or expert editor, that knows a wiki inside out and that is more interesting in the fact that the content of a wiki is accessible to others and usually adopt the behavior of keeping a small page of bookmarked wiki pages that they want to watch for changes so that when a page is changed they read what was changed and either revert the change if they did not approve it or correct the new edit or approve the edition and do nothing (Bryant et al. 2005).

Due to the needs of the professional editors, there is a need for a fast way to tell those editors if the articles they watch have been changed or not and where those changes have happen.

[16]for example. `http://en.wikiversity.org/wiki/Wikiversity:Main_Page`
[17]for example `http://en.wikipedia.org/wiki/Main_Page`
[18]for example the wikis in `http://www.wikia.com/Wikia`

### 2.2.3 Classifying Wikis by Architecture

In this subsection we describe some of the existing wiki architectures and compare their advantages, disadvantages and their use cases (ie. when some architecture is better than other).

#### 2.2.3.1 Centralized Architecture

The centralized architecture is one of the most used wiki architectures and typically consists in a web application, that runs in a web server and that is connected to a database server or file repository that stores the wiki pages belonging to the wiki. When users want to access the wiki, they use the URL of the web server that contains the wiki application.

The advantages of this method is that it is simple, it does not need special software (besides the wiki web application and even that can be done in a similar way to a regular website) and is accessible to anyone that has a HTTP client.

The disadvantages are that since there is only one centralized server, the bandwidth and hardware required to operate a medium to large sized wiki can cost a lot to someone that wants to host the wiki and also if the web server crashes all the data is unavailable and possibly lost (if there are no backups) and finally there is a high coupling between the backend (the database that contains the wiki pages) and the user interface (the web application that is accessed by the user), which opens the door to exploits if the web application has a bad design with vulnerabilities such as cross-site scripting, SQL injection and other problems of a website.

So in order to try to solve some of these problems, the architectures below were designed.

#### 2.2.3.2 Peer-to-Peer Architecture

One example of a peer to peer architecture for wikis is presented the paper by Morris et al.(Morris 2007) and unlike the hybrid approach, it is completely decentralized (or peer to peer), since the wiki users are also the ones that host the pages and there is no central interface to the wiki.

As the requirements for this architecture the authors, considered:

**Redundant decentralization** There is no central server, since everything is stored in peers and when someone wants to see a wiki page, it connects to the peer to peer network in order to talk to the peers and obtain the document he or she wants. Also it is important to provide redundancy since a peer may fail in expected or unexpected ways;

**Unique identification of documents** The documents must be uniquely identified, since each document may have multiple versions and because two users can edit the same version at the same time and create different or conflicting versions of the same document, which gives that the documents may need to be manually merged and for that each version must have its own unique name;

**Usability** This is necessary because a user has a mental model of what it can do with a wiki and the architecture must respect that mental model, so that the user can continue to use the knowledge it has, for example, the user knows that a wiki page may contain links to other pages and that if he clicks on them he will go to the page that was clicked and while simple this can be difficult to implement correctly in a peer-to-peer system where pages might not be in the same storage server;

**Efficient retrieval of documents** Associated with the usability described above, a user may also like to search for a given page, so the protocol must have some way of locating the requested page, so that the hosting peer can send the page to the peer that wants it;

As a peer to peer middleware the authors used JXTA, because it offered an easy way to search for a document, aggressive caching and it allowed for resource and peer discovery (integrated with the search). It also allowed the formation of a group of peers and a way to make that group private using authentication methods.

So with JXTA behind as a peer to peer middleware, when a client wants to access the wiki, he or she launches a client, that connects to the group WikiNet of JXTA and gains access to the resources that belong to that group. Then it opens a JXTA pipe[19] in order to listen for document requests and answer them.

When a document is created, the peer generates a new JXTA advertisement,[20] that contains the publishing date of the document, the title of it, the revision number and the pipe and peer ID of the document holder.

When a document is requested, a peer searches for the advertisement that tells where the document is located and then connects to the specified pipe, so that it can retrieve the wanted page.

Then, in order to increase the redundancy of the page, it publishes a new advertisement specifying that it also has the document (ie. an advertisement that is similar to the one that he searched for, but that has the pipe and peer id corresponding to that peer).

#### 2.2.3.3 Hybrid Architecture

This architecture is described in Urdaneta et al.(Urdaneta et al. 2007)2.7. It presents a way to improve the centralized Wikipedia, using the help of its contributors. So in this architecture the authors split a wiki into two subsystems, one that is centralized and keeps a document index database, that contains the title of each document and some of their words and a front-end that allows the user to query the wiki for a page given a set of keywords, much like a search engine. The other subsystem keeps copies of the pages inside a wiki and a database that indexes them and a front-end that allows such pages to be retrieved and updated. In this last subsystem the placement of the pages, the updates and query requests are handled by a distributed system.

In this paper, for the page placement algorithm the authors considered a cost function that takes into attention the resources demanded by a page, the performance of a node and the evolution of that node, resulting in the following formula:

$$c(N,P,W) = \sum_{p \in P} \alpha\left(\frac{l(p,w)}{l_{tot}(N,W)}\right)^j + \beta\left(\frac{o(p,W)}{o_{tot}(N,W)}\right)^j$$

[19] A JXTA pipe is a way that JXTA has to connect specific peers together so that they share some data

[20] A JXTA advertisement is a way to tell peers about resources and network services available

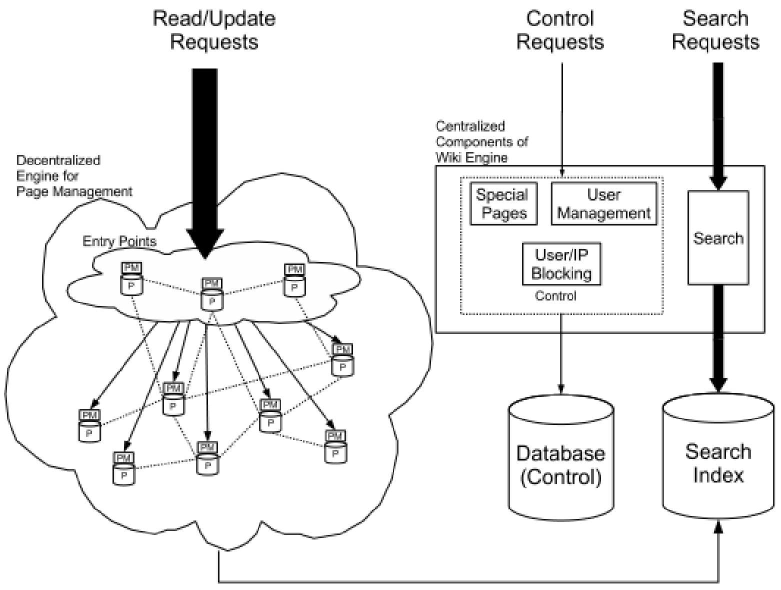


Figure 2.7: Hybrid Wiki

Where $l(p, w)$ is the number of bytes of the received requests for a page p, during a certain time window $w$, $l_{tot}(N, W)$ is the maximum number of bytes that node N can receive within the same time window $w$, $o(p, W)$ is the number of bytes that the request for page p generated as a response in the same time window, while $o_{tot}(N, W)$ is the number of bytes that can be sent from the node N, in the time window w. $\alpha, \beta$ and $j$ are simple weighting constants to balance the cost function.

Finally, the result of the cost function is the summation of all these costs and this provides the value that the page placement algorithm must minimize in order to keep the architecture efficient and fast.

Since this requires global information, the authors of this paper proposed a solution based on the fact that when placing or moving a page, one needs only to calculate if that move will minimize the cost function of those two nodes, so that the nodes can perform this in a concurrent way.

In order for a node to know neighbors so that he can reduce the cost function, the authors considered a gossip based protocol, where the connected nodes change their neighbors list and, for each connection, choose a different set of neighbors based on the exchanged information, so that each node has a new set of other nodes to move the pages it contains.

In order to decide which pages to move, each node chooses a given number of pages, that can be moved and than it tries to see if one page in that set results in a cost reduction and if it does that page is transferred.

In order to save bandwidth, only one page is transferred at a time to a certain neighbor and neighbors that are already receiving a lot of pages are not considered for the placement algorithm. In order to prevent excessive page movement, the nodes only run the algorithm when they load is approaching a certain threshold $T$.

Since any node can receive a query from a user, it is used a Distributed Hash Table, in order to keep an index of pages and the nodes that contain them. Where the key of this DHT is the hash of the name of the page and the value is the node that contains this page, also like a DHT each node is responsible for a given part of the hash table, so that when a query is received a node executes a DHT query in order to know where the page is kept.

In order to prevent arbitrary failures of a node and node departures each page and DHT key can be replicated in various nodes.

#### 2.2.3.4 Discussion of Use Cases

Following the discussion of the main wiki architectures we describe some of the use cases where an architecture might suit best.

So the peer to peer architecture is best when a user wants to create a wiki about a certain subject or when he wants to look something related to a certain subject[21], because the group orientation of the peer to peer architecture associated with the private group feature would allow the edition of the pages to only a set of users.

The hybrid architecture is best when there is already a centralized wiki and due to the number of users of that wiki, it might be useful to set up some data-centers that serve as peers to contain copies of the wiki pages, while the main web server contains the interface and is the single point of access to the users. This, because a peer to peer architecture might pose some privacy (in case of enterprise wikis) or redundancy (since the users might only want to look for something specific and not be interested at all in hosting wiki content) risk.

The centralized architecture is useful for small or private wikis, that have a small number of users, because the problems of bandwidth are nonexistent and the reliability and coupling problems might not be significant.

## 2.3 Vector-Field Consistency

The Vector-Field consistency (or VFC) algorithm(Santos et al. 2007)(Veiga et al. 2010) is an algorithm that unifies consistency enforcing with locality awareness, using the concept of vector-fields.

Where there is a pivot or center point that represents an important object for the system or observation point and generates consistency zones around that object in which we place the objects according to a set of dimensions defined in a vector called the VFC vector.

Each consistency zone has a given consistency degree (Fig. 2.8) and consistency zones are ordered so that as one moves away from the pivot the objects placed have less in common to the given pivot (ie. are more distant).

The consistency zones are shaped like a cube (or square, depending on the number of dimensions or features one wants to monitor), where each consistency zone is defined by a

[21] see for example `http://www.wikia.com/Wikia` that allows users to create wikis about their favorite themes

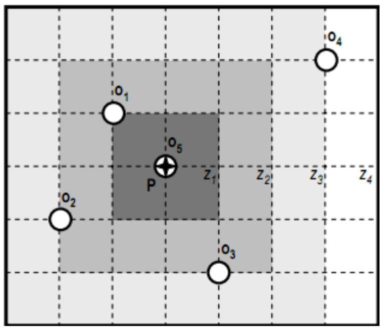


Figure 2.8: Example of a consistency zone

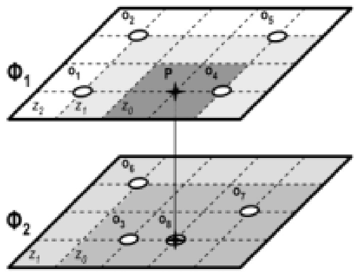


Figure 2.9: Example of multiple views

range that goes from the previous consistency zone (or pivot, in the case of the first zone), to a certain maximum VFC vector value.

In order to position an object in a consistency zone, VFC defines the vector $\kappa = (\Theta, \sigma, \nu)$ to describe the maximum divergence that a zone can tolerate, so that one only needs to check for a zone where $\kappa_i \leq o_i \leq \kappa_{i-1}$ is true, for a given object $o$.

Because one can have multiple pivots and an object may belong to more than a pivot and therefore is surrounded by more than one consistency degree, VFC defines that if that happens, the object belongs to the closest pivot.

In VFC, an object or pivot may also belong to more than one view (Fig. 2.9), where each view has its own set of pivots and therefore one pivot or object may generate more than one consistency zone or belong to more than one consistency zone, in the case of pivots and objects respectively, where a view can be associated with a user, a game field layer, etc.

In previous works VFC was used for mobile game consistency, specially in massive multiplayer online games and first person shooters and the defined VFC vector had three dimensions that were time, sequence and value. Time is the maximum staleness time a game object could have had, sequence the maximum number of lost updates of that object and value the difference between replica contents.

## 2.4 Summary

In this chapter, we analyzed the various architectures for a cache system, the various models for a proxy server, the two great families of cache consistency enforcement approaches, a number of relevant cache replacement strategies and finished with a taxonomy of the commercial cache servers according to the analyzed criteria.

Then we addressed wiki systems, where we classified the wiki types, the types of wiki users and presented relevant wiki architectures.

Finally we explained the VFC algorithm, that serves as the basis for our cache solution and finished with an overview of the past usages of the VFC algorithm in different scenarios.

# 3 Architecture

After discussing some of the related work, we are going to define the architecture of the proposed solution and the relevant algorithms.

We are going to use a centralized architecture for our web cache server, in order to comply with the requirements discussed above of fast accessibility.

In the wiki we are going to use a centralized architecture as the wiki architecture.

## 3.1 Web Cache

Since our work is related to two main parts, the web cache and the wiki, in this chapter, we describe the cache architecture, starting by the VFC adaptation and the description of the cache replacement strategy, an general view of the components of our architecture both server and client side, followed by a use case of a request using the normal HTTP consistency, the registering of a user and his or her consistency definitions and bookmarks including the exchanged network messages and then a request using VFC.

### 3.1.1 VFC Model Adaptation

In the VFC model, we have used a pivot that contains three dimensions, that are:

**Distance** Specifies the number of links that must be crossed from the document that is the pivot, to the current document and must be equal or bigger than zero;

**Recency** Specifies the number of requests that have passed since the last request to this document and must be equal or bigger than zero;

**Frequency** Specifies the number of requests, that the document had while in cache and must be equal or bigger than zero;

Because, we define distance as the number of links between a pivot and a given document, we describe the HTML tags that have a meaning to our algorithm and what they do, so:

**a** This tag refers to an external document or link, so the distance of the linked document is going to be stored as the distance of the current document plus one;

**img** This tag refers to a related image document, the distance of this image document is going to be equal to the distance of the current document, because it is meant to be aggregated with it, using HTML semantics;

**link** This tag is a stylesheet that applies to the HTML document and the distance of this document is going to be equal to the distance of the current document, again because of HTML semantics;

**script** This tag may specify a JavaScript file that is used by the current HTML document, and its distance is going to be equal to the distance of the current document.

Also, since a pivot should typically be the only element with a distance of one, when we are parsing a pivot, we are going to add one to the distance of any of the documents specified by the tags above, but as an exception to all of this, if the pivot is made from an frame page, we are going to keep all pages from the frame in the same area (with a distance of zero).

Also only a bookmark may become a pivot, so if a user makes a request to a document that is not linked to a pivot, then the document isn't cached, even if later (by parsing more pages) it is found that the document did have a connection to a pivot; so that the algorithm can provide a fast way of deciding what to cache in spite of being unable to cache a potential linked page and therefore not providing any prefetching of pages.

If a document needs to be re-validated because it has exceeded the consistency zone limits, and upon parsing the new response, it is found that some of the existing links no longer exist, then they are deleted if they do not have any other parent links, the removal is not recursive, in order to keep the algorithm fast and document are only removed using our cache replacement strategy, described below.

If a user is removed either explicitly or by inactivity, then the pivot references associated with him are removed, again in a non-recursive way, since the pages could be referenced by or be pivots of other users. Also, if a pivot page of a user is a regular page of another user, then that page will have multiple consistency zones and obey to all of them as specified below.

If a document is linked to more than one pivot (they may be from multiple users), then the document has a VFC vector for each connection that is associated with a given pivot, so that when in a request that document needs to be checked for freshness the correct VFC vector is used, according to the pivot and consistency zone of the user, where it is.

### 3.1.2 Cache Replacement Strategy

In a way to make space for new documents or to remove unused documents to make space, there is a need to run a cache document removal process either periodically or when there is a need to make more space for a document.

So, in conformance with the most common used scheme in the existing web caches[1], there are two values important for the document removal/replacement algorithm:

**MaxCacheSize** This is the maximum size of a cache and when this limit is reached, the algorithm is run in order to remove some useless pages or move them to other type of cache;

**MeanCacheSize** This is the value that specifies when the algorithm stops removing documents from cache, when it is run for space reasons;

[1]see the description of squid above

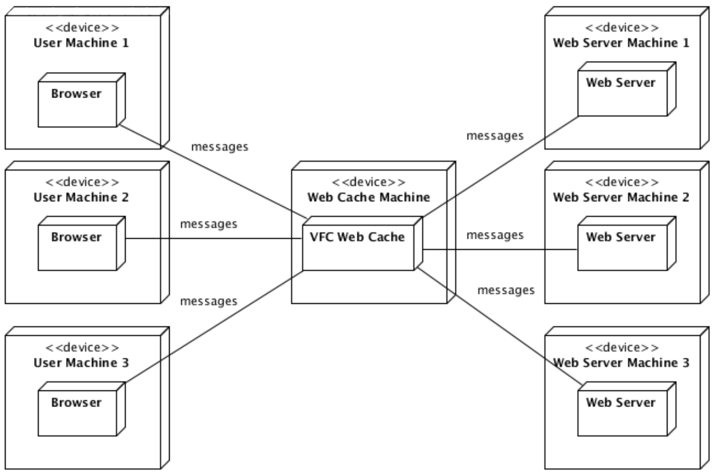


Figure 3.1: Deployment diagram of the web cache and all involved machines

Also, since our cache server has two areas for caches (disk and memory), each area has a set of the above values, so that they can be customized and controlled independently by the cache server administrator. Additionally the units of the two above values are bytes and $MeanCacheSize < MaxCacheSize$.

Also because the VFC vector that each object has specifies useful values for the cache remove/replacement strategy, we specify our strategy using heuristics based on those values, so we have:

**Weighted Distance by Frequency** The documents more distant from the pivots and least used are potential candidates to being replaced/removed from cache (or moved to disk);

**Weighted Recency by Frequency** The documents that were updated a long time ago, but that are not frequently used are potential candidates to being replaced/removed from cache;

Using this heuristics, together with the removal of objects that have no parents, we have an simple cache replacement protocol, that uses the VFC vector to its fullest, while keeping the wanted properties and qualities of user preference awareness this may be changed in order to run a periodic process that removes pages without parents, even if there is no need for removing pages, providing some type of garbage collection.

### 3.1.3 Web Cache Deployment Model

In this subsection, we will give a general overview of all machines, client and server software that are related to our cache and will provide input to our cache and to whom our cache will provide output (Figure 3.1).

So, after looking at the deployment diagram, we add that the cache is based on a forward proxy model and a centralized architecture as defined in the related work chapter in the web cache related section.

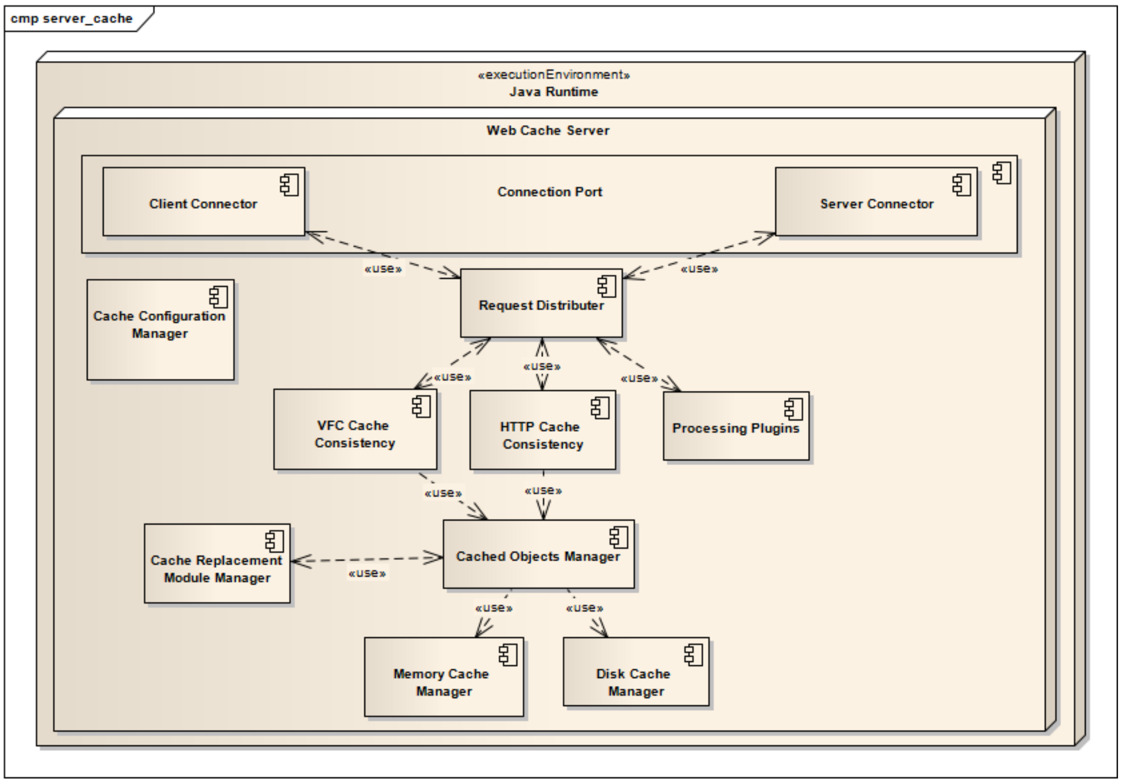


Figure 3.2: Architecture of the web proxy/cache server

### 3.1.4 Base Component Model

We will now give a general view over the components of the web cache architecture, both server and client side.

#### 3.1.4.1 Server Side

As an architecture for the proxy layer (Fig. 3.2), we have a model that divides it into three main blocks and one configuration component that stores global configuration options set by configuration files read upon server startup. So the three main blocks are described below.

**3.1.4.1.1 Port Management** This blocks contains all the ports, that are components that can handle client and/or server connections, in a given protocol (for now there is only ports for HTTP 1.1).

Also, it can be any number of ports, that have a unique identifier and may have only inputs or outputs or both, so that a user may configure a port to receive special requests from a web application or a port that is specific for a certain type of server content like objects with a large size.

The components of this block are described below:

**Port or Connection Port** This component manages the connections both from the clients and to the servers using the pro-actor pattern to handle simultaneous connections from the clients and servers and is divided into two sub-components:

**Client layer** This component is the client side of the port and processes, client requests and sends responses and has sub-components to parse the HTTP request into a generic proxy request and a proxy response to a HTTP response.

**Server layer** This component is the server side of the port and processes, server responses and sends requests and has sub-components to parse the an proxy request to a server request and a server response to a proxy response.

**3.1.4.1.2 Request Processor** This block receives proxy requests from a client connection and using a chain of responsibility distributes the requests for processing by the VFC consistency module or the HTTP consistency module, it may also distribute them through additional modules that may store the message, produce a log of the requests or transform either the request or response.

If a processing module cannot process the request immediately and needs to send an request to the server, this component will also forward those requests to the server connectors and provide the received response.

Finally, the request processor sub-components are described below:

**Request Distributor** This component receives the requests from the ports and introduces them to the chain of responsibility containing the processors and receives requests from the processors and sends them to a web server;

**VFC Cache Consistency** This component manages the VFC protocol and returns the document in cache, if it is fresh according to the VFC parameters and if it exists, otherwise it creates a request for the Request Processor, so that the document is fetched from the original web server, it is also responsible for storing any information, related to VFC and manage the user registration and changes;

**HTTP Cache Consistency** This component uses the HTTP protocol consistency, according to the HTTP RFC;

**Processing Plugins** These are modules that can do other useful things with a request with or without a response, like storing logs and so on.

The VFC Cache consistency, also processes all requests coming from the user browser plugin, like user management, bookmark management and so on. It is also responsible to remove users after a configurable period of inactivity.

**3.1.4.1.3 Storage Component** This block contains components for managing the cache system and the server preferences, specifically:

**Cached Objects Manager** This component manages the cache and is responsible for storing and retrieving objects from the disk and memory caches and completely abstract the storage method, from the consistency managers that use the cache. The documents from the different consistency managers are kept separated, so that they don't conflict;

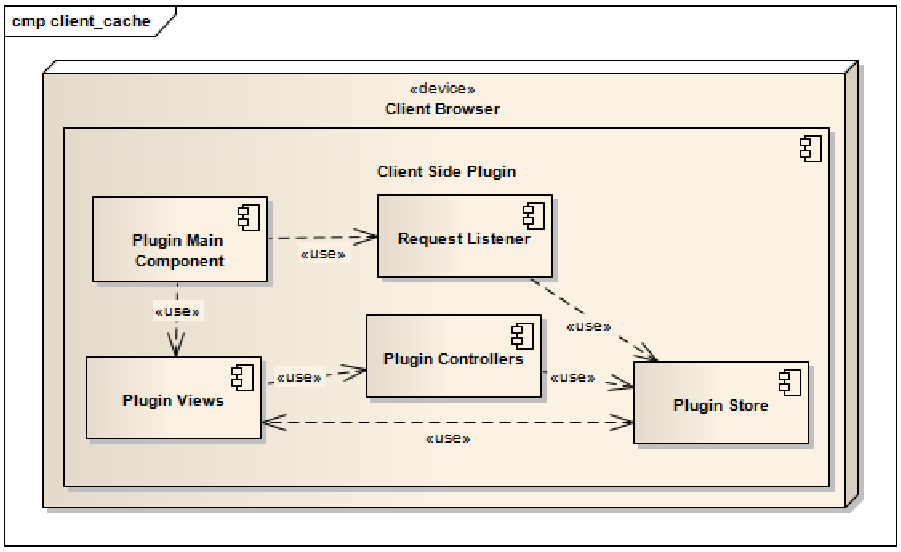


Figure 3.3: Architecture of the web proxy/cache client plugin

**Memory Cache Manager** This component manages a memory cache, that stores documents in memory, using a content-addressed system[2], in order to perform lookup, insert and remove operations in a fast and efficient way;

**Disk Cache Manager** This component manages a disk cache, that stores documents in disk, in order to save space in the memory cache or to store documents that had not been acceded on a long time;

**Cache Replacement Module Manager** This component is responsible to manage all the available cache replacement algorithms and use them according to the user configurations for each consistency method specific cache.

#### 3.1.4.2 Client Side

As an architecture for the proxy layer (Fig. 3.3), we have a model that combines a traditional MVP[3], with a component to change the requests made by the browser and a main component that is represented by a button in the browser and allows the browser user to configure his or her user name, the consistency zone definitions and the bookmarks he wants to consider as pivots. The rest of the blocks are described below.

**3.1.4.2.1 Plugin Views** This component is responsible to present a user interface in order for the user to configure his or her user name, to view the cache generated user name (using an hash of the given user name, browser version and a random unique number), the plugin version, the cache version.

It also allows the user to configure the consistency zone definitions and the maximum VFC vector, before a pivot is updated and to configure his or her bookmarks/pivots to send and optionally adds all of his or her current bookmarks and/or open tabs.

[2]Also known as CAS
[3]Model View Presenter

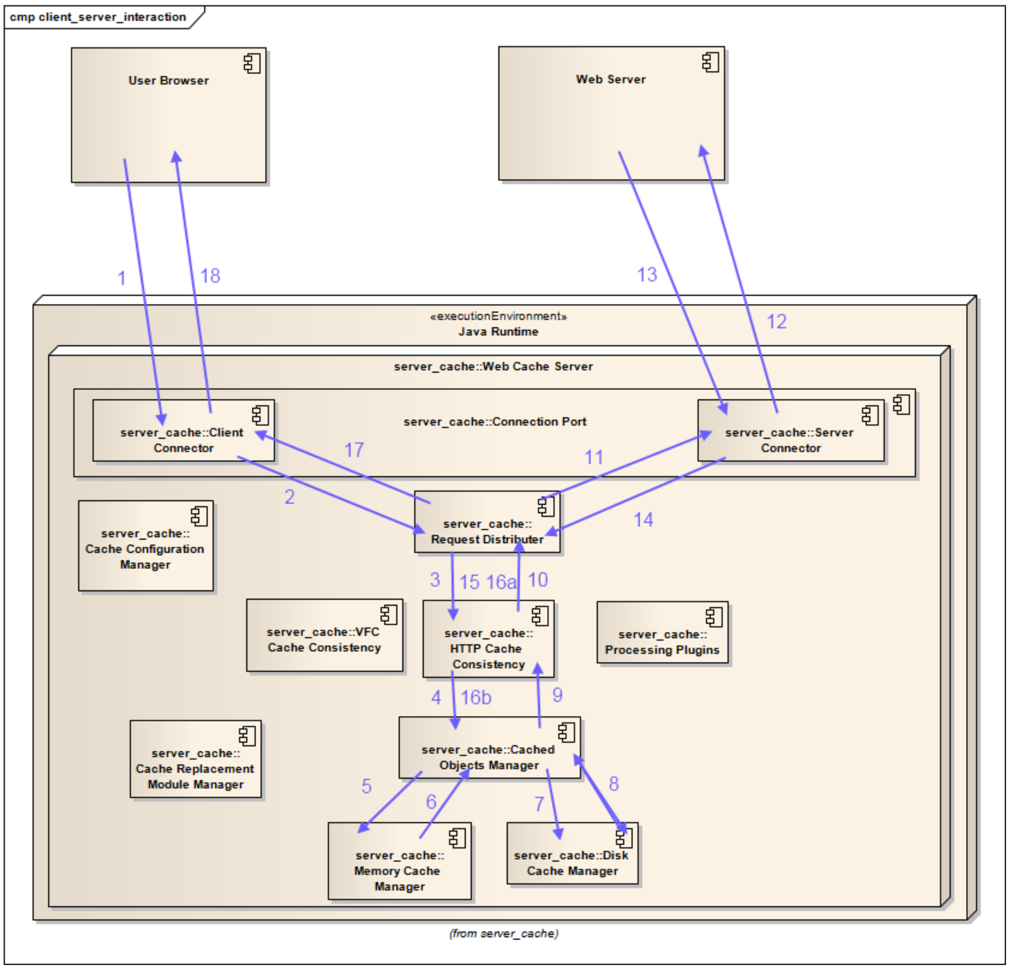

cmp client_server_interaction
User Browser
Web Server
1
18
13
12
«executionEnvironment»
Java Runtime
server_cache::Web Cache Server
server_cache::Connection Port
server_cache::Client Connector
server_cache::Server Connector
17
11
2
14
server_cache:: Request Distributer
server_cache:: Cache Configuration Manager
3
15 16a
10
server_cache::VFC Cache Consistency
server_cache:: HTTP Cache Consistency
server_cache:: Processing Plugins
4
16b
9
server_cache:: Cache Replacement Module Manager
server_cache::Cached Objects Manager
5
6
7
8
server_cache:: Memory Cache Manager
server_cache::Disk Cache Manager
(from server_cache)

Figure 3.4: Use case involving a HTTP request.

**3.1.4.2.2 Plugin Controllers** This component is responsible for listening to changes in the view and do any required communications with the cache server when the user changes his or her name or the consistency zone definitions or any of the other configurable parameters.

**3.1.4.2.3 Plugin Store** This component is responsible to persist the information entered by the user, using the standard means offered by the browser for it.

## 3.1.5 Use Cases

In this section we are going to describe the operations done by the cache when it receives an normal HTTP request, when a new user registers (and its related operations) and when the cache receives a VFC request.

### 3.1.5.1 HTTP Request - Use Case

When a client browser makes a request to the cache (Fig. 3.4), the following steps are done, considering that the page is not in the cache and a server request must be made:

1. The client makes the connection;
2. The client part of the port that receives the connection parses the request, including the headers and their information and builds a generic proxy request object;
3. The generic proxy request object is placed on a queue and retrieved by the request distributor;
4. The request distributor, distributes the request along the chain of responsibility containing the HTTP Cache Consistency, that processes the request;
5. The HTTP Cache Consistency component, sends a get request to the Cached Objects Manager, to see if the request is cached;
6. The Cached Objects Manager, tries to see if the requested object is in the memory cache;
7. The Cached Objects Manager, tries to see if the requested object is in the disk cache;
8. The Cached Objects Manager, sends a failure response to the request;
9. The HTTP Cache Consistency, transforms the client request into a server request and places it in a queue for sending it to the original web server;
10. The Request Distributor, retrieves the server request passing it to a compatible Server Connector, using a queue;
11. The Server Connector, retrieves the request from the queue sending it to the server;
12. Upon receiving a response, the Server Connector, associates the response to the proxy request handling it to the Request Distributor, by placing it in a queue;
13. The Request Distributor, retrieves the request from the queue distributing it to the chain of responsibility, allowing the HTTP Cache Consistency to process it;
14. The HTTP Cache Consistency sends an asynchronous request for the Cached Objects Manager in order to store the received response and simultaneously sends the response to the client request, by placing it in a queue;
15. The Request Distributor, retrieves the request from the queue handling it to the Client Connector;
16. The Client Connector answers the client sending it the response.

#### 3.1.5.2 VFC User Registration - Use Case

When a client browser makes a request to the cache (Fig. 3.5), in order to register a user and his or her information, the following steps are done:

1. The user browser plugin makes a request, to register the user and his or her data, that is a regular HTTP request;
2. The Client Connector, sends the data to the Request Distributor, that places the request in the chain of responsibility;

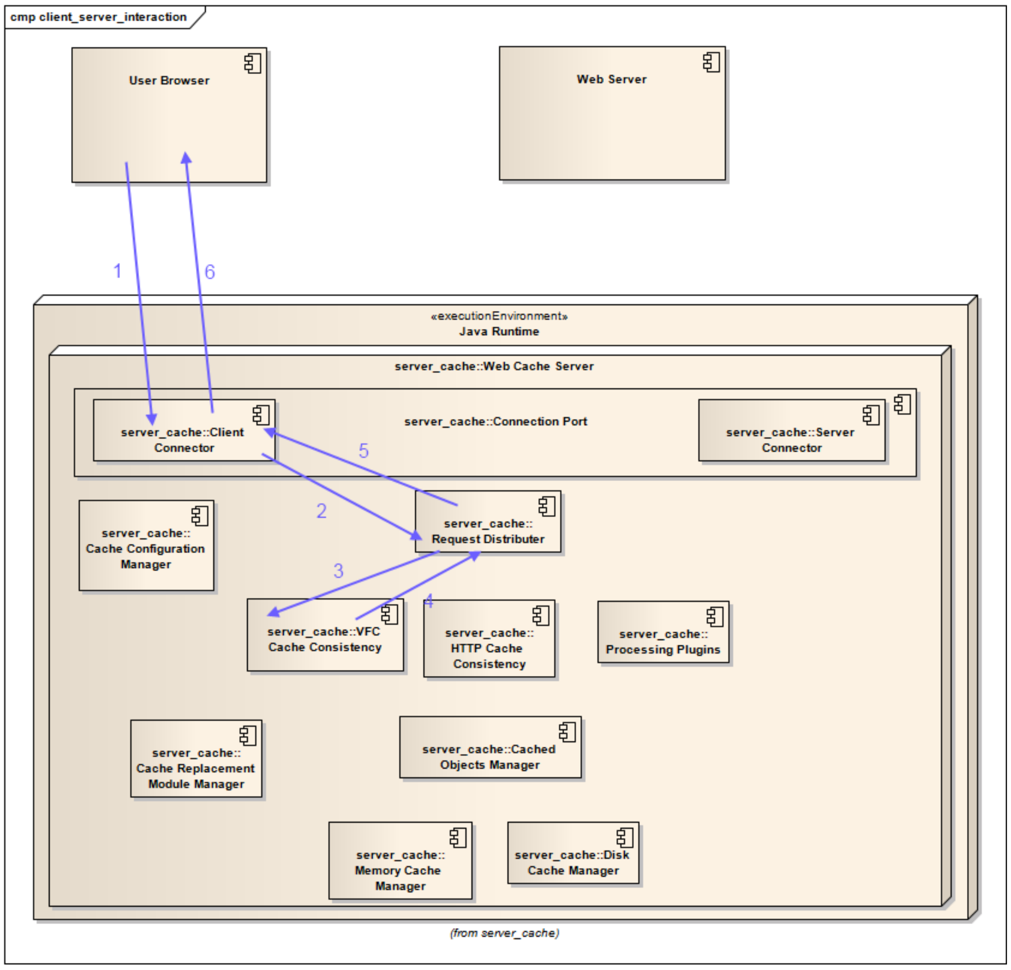

cmp client_server_interaction
User Browser
Web Server
1
6
«executionEnvironment»
Java Runtime
server_cache::Web Cache Server
server_cache::Client Connector
server_cache::Connection Port
server_cache::Server Connector
5
2
server_cache:: Request Distributer
server_cache:: Cache Configuration Manager
3
4
server_cache::VFC Cache Consistency
server_cache:: HTTP Cache Consistency
server_cache:: Processing Plugins
server_cache:: Cache Replacement Module Manager
server_cache::Cached Objects Manager
server_cache:: Memory Cache Manager
server_cache::Disk Cache Manager
(from server_cache)

Figure 3.5: Use case involving a user registration.

3. The VFC Cache Consistency, handles the request processing the received data, adapting the user pivots or the user consistency zones if they change;
4. The VFC Cache Consistency, sends a response to the Request Distributor, that may contain the proxy version and the generated user name;
5. The Request Distributor, sends the response to the Client Connector;
6. The Client Connector sends the response to the User Browser.

The message sent by the user browser plugin, may contain a new user name to create or replace the existing user name or a generated user name, followed by a set of bookmarks or consistency zone definitions that may create new pivots and respective consistency zones or change the existing ones.

#### 3.1.5.3 VFC Request - Use Case

When a client browser makes a request to the cache using VFC, the following steps are done, considering that the page is not in the cache and a server request must be made. We highlight that the first three steps are equal to the use case of the normal HTTP request and are therefore not represented, so that our list starts after the first three steps are done.

1. The first three steps are equal to the normal HTTP case;
2. The request distributor, distributes the request along the chain of responsibility containing the VFC Cache Consistency, that processes the request;
3. The VFC Cache Consistency component, determines the VFC Page corresponding to the request sending a get request to the Cached Objects Manager, to see if the request is cached;
4. The three steps involving the Cached Objects Manager, are equal to the normal HTTP steps;
5. The VFC Cache Consistency, transforms the client request into a server request placing it in a queue for sending it to the original web server;
6. The normal HTTP request steps 10, 11 and 12 are common to the VFC request use case and take place at this time;
7. The Request Distributor, retrieves the request from the queue distributing it to the chain of responsibility, allowing the HTTP Cache Consistency to process it;
8. The VFC Cache Consistency parses the received page if the page is a html request, adds the parsed links as shadow VFC Pages to the consistency zones where the parent is inserted sending an asynchronous request for the Cached Objects Manager in order to store the received response and simultaneously sends the response to the client request, by placing it in a queue. It also adds one to the recency value of all pages;
9. The final client response handling is equal to the steps 15 and 16 of the normal HTTP request.

If for some reason the page already is in cache, the VFC request manager gets the corresponding VFC Page, determines if it is fresh or not, that is, sees if its current pivot is within the consistency zone boundaries and if it is adds one to its frequency and one to the recency of all VFC Pages.

If not it sends a conditional request to the server if possible and retrieves the new page, if there are changes to the page, then the page is re-parsed and the new links merged with the old ones and the page in cache is changed.

## 3.2 VFC-Wiki

### 3.2.1 VFC Adaptations

For the VFC-Wiki architecture and since we are using a centralized architecture, this means that the wiki server has access to all of the data about changes to a page and the current users and their pivots. That are defined as the watched pages and the current page (much like the case with VFC-Web pivots), we use the following as VFC vector dimensions:

**Distance** This is the same as the VFC-Web distance (ie. the number of links between a page and the nearest pivot) and is always bigger or equal to zero (when is the case of embedded images or videos);

**Sequence** This is the maximum number of updates that a user is willing to tolerate, before wanting to be notified about them and is equal or bigger than zero;

We highlight that the rest of the special behavior is identical to the VFC-Web protocol described above, including the handling of multiple users, wiki pages with multiple pivots and users share the same pivot.

### 3.2.2 VFC-Wiki Deployment Model

Similarly to what we have done for the web cache, in this section we will describe the machine and processes involved in our wiki implementation and all of the surrounding machines and processes (Figure 3.6).

So, after looking at the deployment diagram, we add that the wiki is based on a centralized architecture as defined in the related work chapter in the wiki related section.

### 3.2.3 VFC-Wiki Architecture

Since we used XWiki as a base to develop our system, we are going to describe the architecture of XWiki, since our VFC-Wiki is implemented as a plugin upon XWiki.

So, the components in the already implemented wiki, are based on a standard model-view-controller architecture and therefore separate the domain from the view and the wiki logic, as described in figure 3.7.

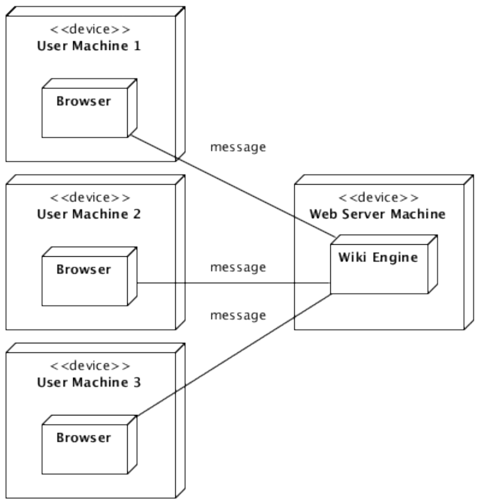


Figure 3.6: Deployment diagram of the wiki and all involved machines

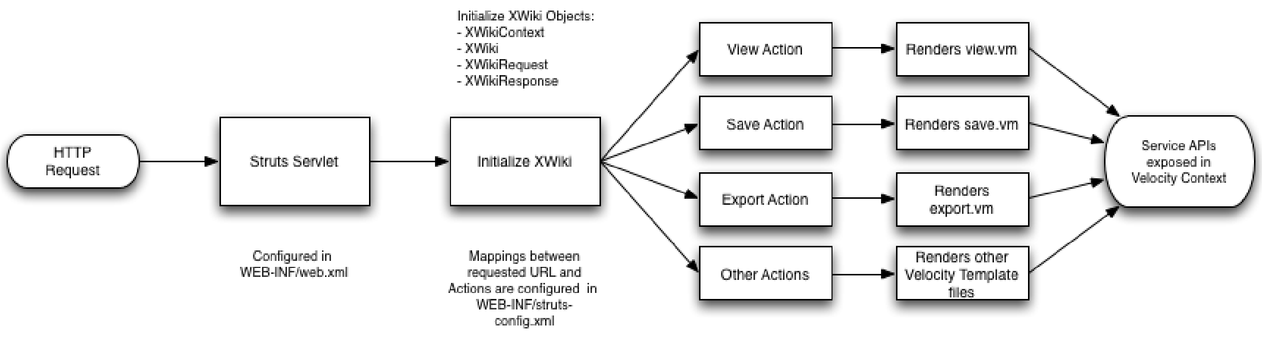

HTTP Request
Struts Servlet
Configured in WEB-INF/web.xml
Initialize XWiki Objects:
- XWikiContext
- XWiki
- XWikiRequest
- XWikiResponse
Initialize XWiki
Mappings between requested URL and Actions are configured in WEB-INF/struts-config.xml
View Action
Save Action
Export Action
Other Actions
Renders view.vm
Renders save.vm
Renders export.vm
Renders other Velocity Template files
Service APIs exposed in Velocity Context

Figure 3.7: Architecture of a XWiki request.

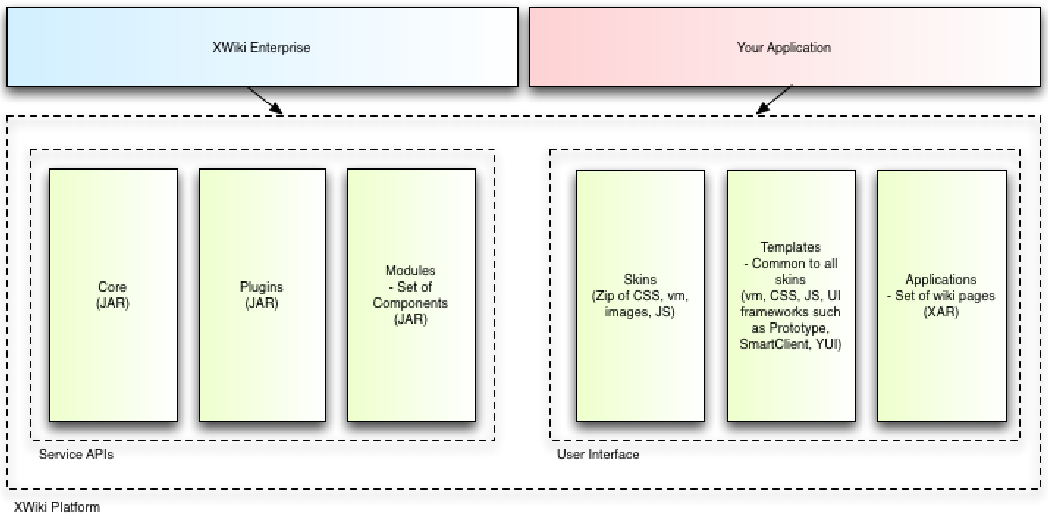

XWiki Enterprise
Your Application
Core (JAR)
Plugins (JAR)
Modules - Set of Components (JAR)
Service APIs
Skins (Zip of CSS, vm, images, JS)
Templates - Common to all skins (vm, CSS, JS, UI frameworks such as Prototype, SmartClient, YUI)
Applications - Set of wiki pages (XAR)
User Interface
XWiki Platform

Figure 3.8: Blocks of XWiki.

In terms of blocks, the already implemented wiki, uses the blocks specified in Figure 3.8.

From this figure, it is important to discuss that the core component implements the model, the localization service, the notification service, the user management, among other services. And that our implementation will be done as a module and an application, since it will involve two plugins and a few pages in order to implement the user interface.

### 3.2.4 Use Cases

In this section we are going to describe the use cases related to the wiki, when a user adds a watched page to his or her user page, when a user removed a watched page from his or her user page and when the user wants to get a list of all updated pages.

#### 3.2.4.1 Add Watched Page - Use Case

When a user adds a watched page to his or her personal list of watched pages, our VFC plugin gets notified of that event and will do the following algorithm in order to respond to it:

1. Get the name and id of the added page;
2. Retrieve the matched page;
3. Check if the page already has VFC information:
   (a) If it has VFC information, than add the VFC information for the current user, to the page (mainly a pair of username/(distance, pivot));
   (b) Else, create a new VFC information object in the page and add the VFC information for the current user;
4. For each, non-repeated child page, do the steps 3 to 4.

#### 3.2.4.2 Remove Watched Page - Use Case

When a user removed a watched page from his or her personal list of watched pages, our VFC plugin gets notified of that event and will do the following algorithm in order to respond to it:

1. Do the first two steps as the add watched page user case;
2. Check if the page has information about the user:
   (a) If it has, then remove the information about the user;
   (b) Else, log an warning;
3. For each, non-repeated child page, do the steps 2 to 3.

#### 3.2.4.3 Get Updated Pages - Use Case

When a user requests the list of updated pages from his or her personal list of watched pages, our VFC plugin gets notified of that event and will do the following algorithm in order to respond to it:

1. Get a list of the user watched pages;
2. For each watched page:
   (a) Check if the page is fresh according to the VFC parameters of the user (Note that in XWiki pages, already have a tracker of the number of changes they have had);
   (b) If the page is not fresh, than add the page to a list of updated pages;
   (c) For each, non-repeated child page, do the steps a) and b);
   (d) Concatenate all updated pages into one list;
3. Concatenate the list of all updated pages received from the for loop, into one list and return it to the user.

#### 3.2.4.4 Notes

Given that an wiki page has a reduced set of pages, compared to the web cache, we have choosen an algorithm that involves recursion, in the hope that the number of child pages and links between the pages, is small or tolerable enough so that the cost of a recursion is insignificant.

## 3.3 Summary

In this section we have analysed our adaptation of the VFC algorithm to the cases of both the wiki and the web cache, including the description of the deployment architecture, that in the case of the web cache, described all the machines that communicated with the cache, including the user browsers and the http servers. While in the case of the wiki, it included the client machines of the users or editors using the wiki.

Then, we described the components of the wiki and cache and finally we described some use cases for both the cache and wiki, with the steps involved in them.

# 4 Implementation

After discussing the architecture we are going to discuss the implementation of the parts related to VFC with more detail.

## 4.1 VFC-Cache

The VFC-Cache was implemented from scratch, using the architecture described above and the Java language, together with some libraries, in order to ease the interfacing with the HTTP protocol, to implement the cache storage system and to implement an way to configure the cache.

About, the VFC component, it was implemented using two sets of classes, one in order to provide storage of pages in their consistency zones and information about the users, and their pivots.

The other was created in order to process the VFC related requests and the use cases of a page retrieval using VFC and user registration within VFC.

### 4.1.1 Relevant Libraries

In this subsection we will describe some of the major libraries used in our implementation, and why we have choosen to use them.

**Netty**[1] This is the library used for handling user and server connections and parsing the received HTTP messages, extracting all of the information contained in the headers, including the HTTP status line;

**ehcache**[2] This is the library that is used to handle cached pages in memory and persist them to disk, also offering an way to search for pages given their URL and to provide hooks (using the Policy interface[3]) so that we can use our custom VFC replacement policy;

**slf4j and logback**[4] These are the libraries used to handle logging in our cache and allow us to see which cache consistency method is being used;

**htmlparser**[5] This is the html parser library used, to parse the html pages and extract the links from it.

[3]http://ehcache.org/documentation/apis/cache-eviction-algorithms#plugging-in-your-own-eviction-algorithm

### 4.1.2 Domain Classes

To implement the domain of VFC (Figures 19, 20, 21 in the appendix), we used the following classes:

**VFCPageList** This is a list that stores every page known to VFC, so that if a similar page (with same URL), is requested from other user or using other pivot, that same page is used instead of a new one (since objects in Java are just references, to the same object);

**StandardVFCLinkVisitor** This class, using a library for parsing HTML, tries to extract all valid URLs (that have at least a path) linked to the page being parsed and builds the information needed for VFC to work, in other words, it associates a link type to an URL, so that the algorithm, can know if the URL comes from an image link or a script link;

**VFCLinkInformation** This class represents the information extracted from a link by the StandardVFCLinkVisitor and contains the URL and type of link;

**Users** This is a singleton, that stores all of the registered users and that contains methods in order to help the main VFC algorithm;

**User** This is an object representing a user and contains its registered user name, his VFC definitions, the pivots that he has and methods to help the VFC algorithm;

**Pivot** This is an object that represents a user pivot and it contains the page represented by this pivot, the maximum VFC value that the pivot can have before requiring an update and a list of consistency zones associated with the pivot and finally some methods to help the VFC algorithm;

**ConsistencyZone** This is an object, that represents a consistency zone and contains the maximum VFC vector, that any object in the consistency zone can have and a list of pages currently in the consistency zone;

**VFCPage** This is an object, that represents a VFCPage and contains the type of link that created the page, the URL associated with the page, the unique id of the request associated with the page, a list of VFC Vectors associated with the page (in order to represent multiple views, of users or pivots), a list of the parent of the page, so that the cache replacement strategy can remove them, when they are no longer needed and a bit that indicates if the page is in cache or not;

**VFCZoneDefinitions** These are the VFC definitions for a user.

**VFCVector** This is an object, representing all the fields of a VFC-Cache vector (distance, recency and frequency);

### 4.1.3 Request Processing and User Registration

To implement the request processing and user registration, we used the following classes (Figure 4.1).

**VFCStandardRequestAction** This class implements the response to a user request;

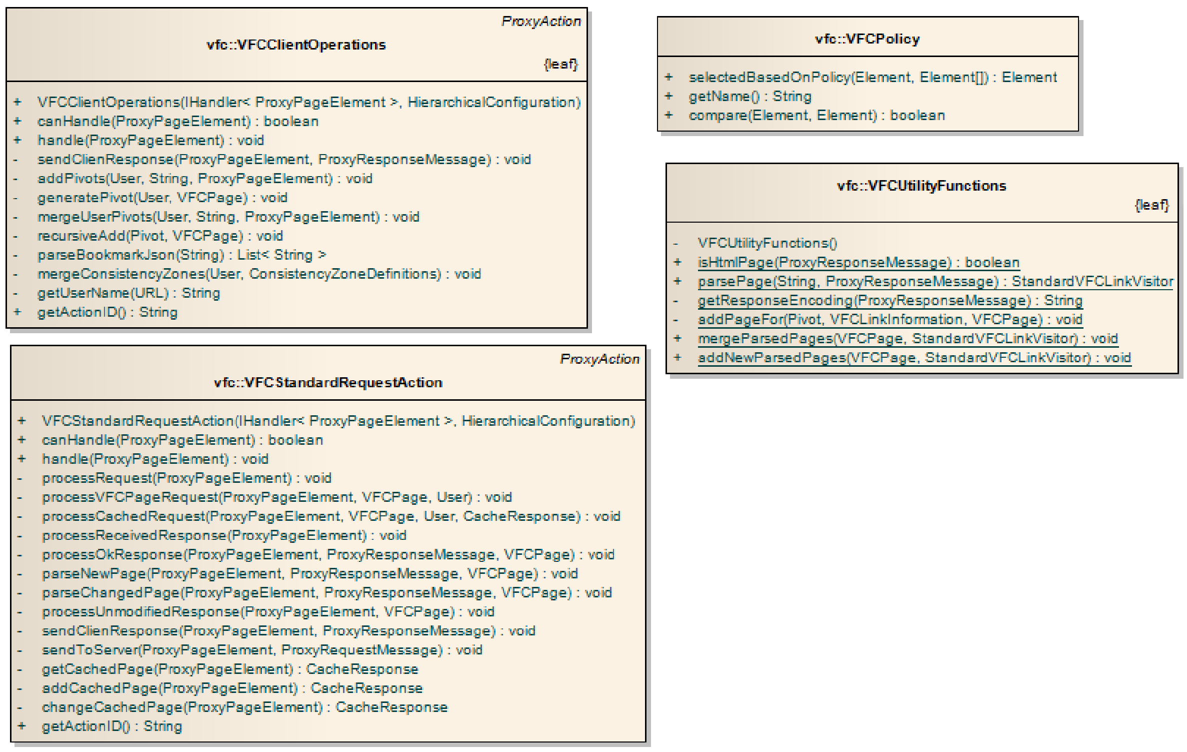

ProxyAction
vfc::VFCClientOperations
{leaf}
+ VFCClientOperations(IHandler< ProxyPageElement >, HierarchicalConfiguration)
+ canHandle(ProxyPageElement) : boolean
+ handle(ProxyPageElement) : void
- sendClienResponse(ProxyPageElement, ProxyResponseMessage) : void
- addPivots(User, String, ProxyPageElement) : void
- generatePivot(User, VFCPage) : void
- mergeUserPivots(User, String, ProxyPageElement) : void
- recursiveAdd(Pivot, VFCPage) : void
- parseBookmarkJson(String) : List< String >
- mergeConsistencyZones(User, ConsistencyZoneDefinitions) : void
- getUserName(URL) : String
+ getActionID() : String
vfc::VFCPolicy
+ selectedBasedOnPolicy(Element, Element[]) : Element
+ getName() : String
+ compare(Element, Element) : boolean
vfc::VFCUtilityFunctions
{leaf}
- VFCUtilityFunctions()
+ isHtmlPage(ProxyResponseMessage) : boolean
+ parsePage(String, ProxyResponseMessage) : StandardVFCLinkVisitor
- getResponseEncoding(ProxyResponseMessage) : String
- addPageFor(Pivot, VFCLinkInformation, VFCPage) : void
+ mergeParsedPages(VFCPage, StandardVFCLinkVisitor) : void
+ addNewParsedPages(VFCPage, StandardVFCLinkVisitor) : void
ProxyAction
vfc::VFCStandardRequestAction
+ VFCStandardRequestAction(IHandler< ProxyPageElement >, HierarchicalConfiguration)
+ canHandle(ProxyPageElement) : boolean
+ handle(ProxyPageElement) : void
- processRequest(ProxyPageElement) : void
- processVFCPageRequest(ProxyPageElement, VFCPage, User) : void
- processCachedRequest(ProxyPageElement, VFCPage, User, CacheResponse) : void
- processReceivedResponse(ProxyPageElement) : void
- processOkResponse(ProxyPageElement, ProxyResponseMessage, VFCPage) : void
- parseNewPage(ProxyPageElement, ProxyResponseMessage, VFCPage) : void
- parseChangedPage(ProxyPageElement, ProxyResponseMessage, VFCPage) : void
- processUnmodifiedResponse(ProxyPageElement, VFCPage) : void
- sendClienResponse(ProxyPageElement, ProxyResponseMessage) : void
- sendToServer(ProxyPageElement, ProxyRequestMessage) : void
- getCachedPage(ProxyPageElement) : CacheResponse
- addCachedPage(ProxyPageElement) : CacheResponse
- changeCachedPage(ProxyPageElement) : CacheResponse
+ getActionID() : String

Figure 4.1: Classes implementing the processing

**VFCClientOperations** This class implements the operations to a user registration;

**VFCUtilityFunctions** This class implements some functions used by the other VFC classes, for adding new pages to the VFC pivot consistency map or to parse new or modified pages, for example;

**VFCPolicy** This class implements the VFC page replacement policy, on top of ehcache, using the specified Policy interface to insert our policy into the the ehcache system.

#### 4.1.3.1 Request Processing

The component used for the request processing is the class VFCStandardRequestAction, using the algorithm specified in the algorithm 1 and algorithm 2.

**Algorithm 1** Process a request using VFC - Part 1

```
if Request is from the user then
    if If the request contains a user name then
        if Page is known to the user then
            if Page is cached then
                if Page is fresh then
                    Serve the page to the user
                else
                    Send a conditional request to the server, if possible or a normal request to the server
                end if
            else
                Send a request to the server
            end if
        else
            if Page is known globally then
                if Page is cached then
                    if Page is fresh then
                        Serve the page to the user
                    else
                        Send a conditional request to the server, if possible or a normal request to the server
                    end if
                else
                    Send a request to the server
                end if
            else
                Send a request to the server and mark the request as uncacheable
            end if
        end if
    else
        Send a request to the server and mark the request as uncacheable
    end if                                              ▷ Continues in listing 2
```

**Algorithm 2** Process a request using VFC - Part 2

```
else                                                        ▷ Continuation of listing 1
    if Request is cacheable then
        Get the page representing the request
        if Request is cached then
            Extract the links of the new page.
            Merge the existing page information, removing the parent of all pages not mentioned in the new page and adding the new pages.
            Increment the frequency value of the visited page;
            Increment the recency value of all other pages.
        else
            Parse the received page;
            Extract all the links;
            For each link, see if a VFCPage already exists and if it does reuse it, otherwise create a new one.
            Increment the frequency value of the visited page;
            Increment the recency value of all other pages.
        end if
        if Request/Response can be cached then
            Queue the received response, for caching within the cache manager;
            if Caching is successful then
                Set the VFCPage as cached;
            else
                Set the VFCPage as not cached;
            end if
        end if
        Send the received response to the user.
    else
        Send the received response to the user.
    end if
end if
```

#### 4.1.3.2 Page Replacement Algorithm

The class responsible for the replacement of pages, is the class VFCPolicy, which by virtue of the usage of ehcache as the inner cache storage system and by obeying to the Policy inteface for page replacement algorithm provided by ehcache, receives as input an array of elements given by ehcache, that are possible candidates for removal, and the element that is going to be added and returns the element to be removed, that should belong to the array of elements given as input, so the algorithm proceeds as listed in algorithm 3.

**Algorithm 3** Choose a page to be replaced.

```
Get the page representing, the pages to be removed
for all VFCPage in candidate list do
    if Page distance <6 and Page frequency >6 then
        Add the value of distance/(frequency+1) to the score of the page in a score list.
    end if
    if Page recency >6 and Page frequency <6 then
        Add the value of recency/(frequency+1) to the score of the page in a score list.
    end if
end for
Get the page with the lowest score and select the page for removal
for all VFCPage in VFCPageList do
    if Page has no parents and the page is not the selected page then
        Remove page from VFCPageList    ▷ Comment: This will trigger a cache remove, for the pages due to the VFCPage finalize method
    end if
end for
Return the selected page
```

#### 4.1.3.3 User registration

The class responsible for registering new users and getting the new pivots and new consistency zone definitions, is the VFCClientOperations. This component uses, an restful interface, on top of HTTP, with some messages, given as Json objects. So the restful interface defines the following methods:

**registerUser(userName, browserName): generatedUserName, proxyVersion** This is the method for registering a user in the cache, using a hash of the userName and browserName, given as query parameters of a get request, that returns a json object containing the generated user name and the version of the proxy.

**unregisterUser(userName)** This is the method for unregistering a user from the cache, using a simple get request with the user name as the parameter query.

**changeBookmark(userName, bookmarks)** This is the method for changing the list of user bookmarks (or pivots), where an empty list, means that all pivots are to be removed.
This method uses a get request with the user name as the query parameter and a JSON object that specifies the bookmarks to add/remove.

**changeConsistency(userName, consistencyZoneDefenitions)** This is the method for changing the list of consistency zones or the maximum vfc of the pivot and is an get request with the userName as a query parameter and a JSON object that specifies the definitions of the consistency zone, that is an array of VFC vectors, corresponding to the maximum VFC value that the consistency zone can take.

## 4.2 VFC-Wiki

For the VFC-Wiki, we used XWiki, because it allowed us to implement the VFC functionality, on top of an already stable and usable wiki and that contains a watchlist functionality, which we use to support our pivots.

Also, XWiki allows us to receive events and store arbitrary information along with the wiki pages.

### 4.2.1 VFC-Wiki Model

For the VFC-Wiki model and using the fact that XWiki allows us to store information, with wiki pages and that a user is also represented by a wiki page. We implemented the model, described in the class diagram of figure 22 that is in the appendix.

**Document** Class that represents the VFC data that a document has, mainly a link to the wiki document represented by it, a list of users/pivots containing the document, along with the distance from the pivot and the consistency zone containing the document and methods to access the links of the current document;

**VFCVectorInformation** It is used by the document class to store information about the document distance to the pivot and the consistency zone where the document belongs according to a given user/pivot;

**VFCVector** Represents a VFCVector and contains the distance and sequence fields of the VFC vector;

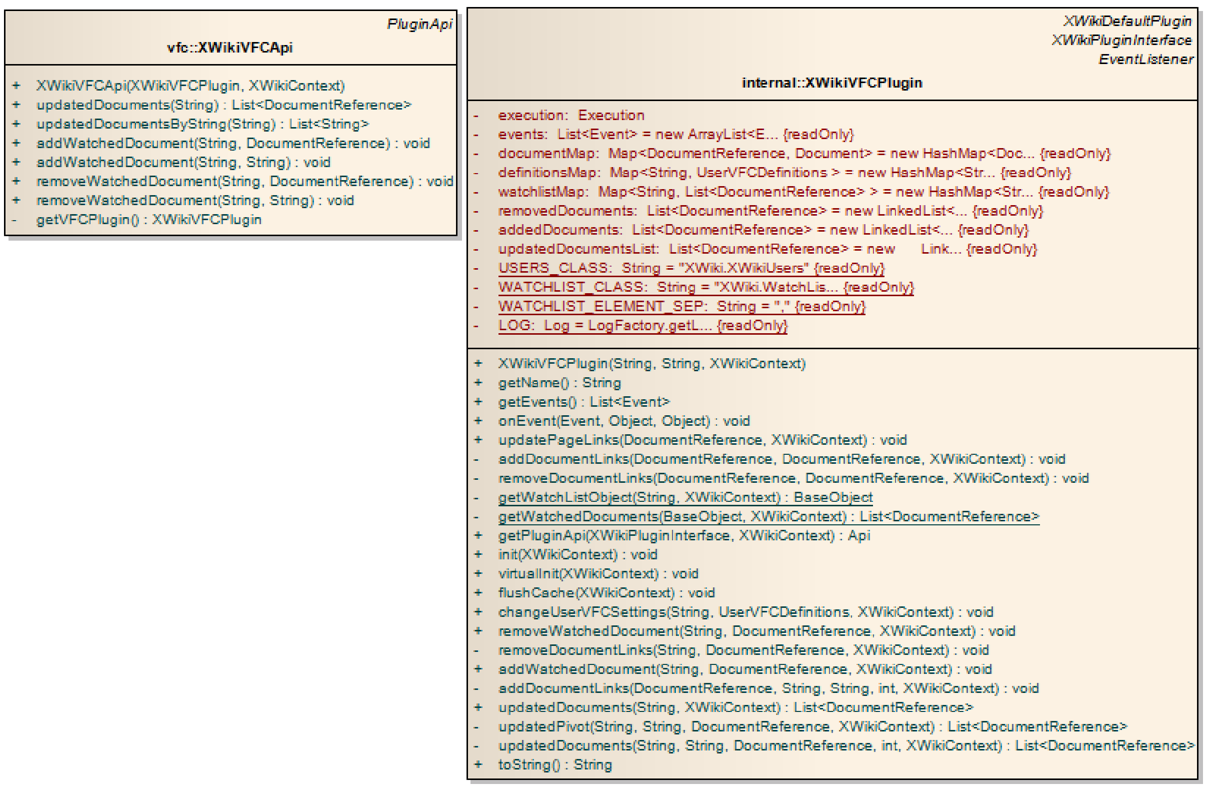
**UserVFCDefinitions** Represents the user VFC definitions, specifically the maximum VFC Vector of the user consistency zones and the maximum VFC Vector of the user pivot;

**VFCDefinitionsConsistencyClass** Static class, that is used as a bridge between XWiki document storage and the VFC plugin, that reads the user VFC Definitions from his or her profile. It also appends the VFC Definitions to the user profile, if they don't already exist in it.

### 4.2.2 VFC-Wiki Operations

For the operational part of VFC-Wiki, that is implemented using the model and the XWiki plugin infrastructure, we created the classes given in figure 4.2. Since XWiki allows us to be notified when a document[6] is created, modified and/or deleted, we base the plugin under that envent-based functionality. So we have the following classes:

[6]For XWiki, an document can be a wiki document, a set of wiki configurations or a user profile



```
PluginApi
vfc::XWikiVFCApi

+ XWikiVFCApi(XWikiVFCPlugin, XWikiContext)
+ updatedDocuments(String) : List<DocumentReference>
+ updatedDocumentsByString(String) : List<String>
+ addWatchedDocument(String, DocumentReference) : void
+ addWatchedDocument(String, String) : void
+ removeWatchedDocument(String, DocumentReference) : void
+ removeWatchedDocument(String, String) : void
- getVFCPlugin() : XWikiVFCPlugin
```

```
XWikiDefaultPlugin
XWikiPluginInterface
EventListener
internal::XWikiVFCPlugin

- execution: Execution
- events: List<Event> = new ArrayList<E... {readOnly}
- documentMap: Map<DocumentReference, Document> = new HashMap<Doc... {readOnly}
- definitionsMap: Map<String, UserVFCDefinitions > = new HashMap<Str... {readOnly}
- watchlistMap: Map<String, List<DocumentReference> > = new HashMap<Str... {readOnly}
- removedDocuments: List<DocumentReference> = new LinkedList<... {readOnly}
- addedDocuments: List<DocumentReference> = new LinkedList<... {readOnly}
- updatedDocumentsList: List<DocumentReference> = new      Link... {readOnly}
- USERS_CLASS: String = "XWiki.XWikiUsers" {readOnly}
- WATCHLIST_CLASS: String = "XWiki.WatchLis... {readOnly}
- WATCHLIST_ELEMENT_SEP: String = "," {readOnly}
- LOG: Log = LogFactory.getL... {readOnly}

+ XWikiVFCPlugin(String, String, XWikiContext)
+ getName() : String
+ getEvents() : List<Event>
+ onEvent(Event, Object, Object) : void
+ updatePageLinks(DocumentReference, XWikiContext) : void
- addDocumentLinks(DocumentReference, DocumentReference, XWikiContext) : void
- removeDocumentLinks(DocumentReference, DocumentReference, XWikiContext) : void
- getWatchListObject(String, XWikiContext) : BaseObject
- getWatchedDocuments(BaseObject, XWikiContext) : List<DocumentReference>
+ getPluginApi(XWikiPluginInterface, XWikiContext) : Api
+ init(XWikiContext) : void
+ virtualInit(XWikiContext) : void
+ flushCache(XWikiContext) : void
+ changeUserVFCSettings(String, UserVFCDefinitions, XWikiContext) : void
+ removeWatchedDocument(String, DocumentReference, XWikiContext) : void
- removeDocumentLinks(String, DocumentReference, XWikiContext) : void
+ addWatchedDocument(String, DocumentReference, XWikiContext) : void
- addDocumentLinks(DocumentReference, String, String, int, XWikiContext) : void
+ updatedDocuments(String, XWikiContext) : List<DocumentReference>
- updatedPivot(String, String, DocumentReference, XWikiContext) : List<DocumentReference>
- updatedDocuments(String, String, DocumentReference, int, XWikiContext) : List<DocumentReference>
+ toString() : String
```

Figure 4.2: Classes implementing the VFC-Wiki plugin

**XWikiVFCApi** This is the interface that the document interface scripts see and corresponds to the public interface of the plugin and allows a user to directly add pivots, remove pivots and to get a list of updated documents, according to VFC;

**XWikiVFCPlugin** This is the internal or hidden part of the plugin, that implements the public API and listens to the events provided by the XWiki core engine. It also adds a VFCDefinitionsConsistencyClass object to the user profile, if it does not already have one.

Also, as we said the plugin listens to the creation, change or removal of XWiki documents, so according to the event we do one of the actions, mentioned in the paragraphs, below.

#### 4.2.2.1 Document created

This event is fired after a document is created and performs the actions explained in algorithm 4.

In this algorithm it is important to note that whenever the user is first created and if the user does not have a VFC Definition object, one will be created and appended to his or her profile.

Also, when the watched documents (or pivots) children are added to the consistency zones, this is done according to a recursive algorithm, so that all of the linked documents are added, without the same document being added twice (in that case, the recursion is stopped).

**Algorithm 4** Action executed after a document is created

```
if The document is a user page then
    Parse the VFC Definitions of the user
    Parse the list of watched documents
    Insert all the documents watched by the user in the new consistency zones
end if
```

#### 4.2.2.2 Deleting Document

This event is fired before a document is removed and does the actions explained in algorithm 5.

In this algorithm it is important to note, that when the watched documents are removed, only the information about the given user is removed, using an recursive algorithm, so that all of the linked documents are also removed from the user consistency zones, unless there is a cycle in the linked documents ( one document, points towards a document already seen).

**Algorithm 5** Action executed before a document is removed

```
if The document is a user page then
    Remove all the references to the user from the pivots and their children documents
    Remove the cached user VFC definitions
    Remove the cached list of watched files (or pivots)
end if
```

#### 4.2.2.3 Updating Document

This event is fired before a document is updated and does the actions explained in algorithm 6.

In this algorithm, the old children of the document are cached using a method, inside the Document class, that is the backupOldChildren method.

**Algorithm 6** Action executed before a document is updated

```
if The document is not a user page then
    if The document is known to the VFC System then
        Cache the old children of the document;
    end if
end if
```

**4.2.2.3.1 Document Updated** This event is fired after a document is updated and does the actions explained in algorithm 7.

When the documents are reinserted in the consistency zones, after a definition change, only the definitions associated with the user are changed and for all documents and their children using a recursive algorithm that stops if an cycle is detected.

**Algorithm 7** Action executed before a document is updated

```
if The document is a user page then
    Get the user VFC Definitions
    if The definitions are valid then
        if The definitions have changed then
            Reinsert all the documents to the consistency zones
            Replace the old cached definitions with the new definitions
        end if
        Get the list of new and removed watch documents
        if There are new watched documents then
            Add the documents to the user consistency zones
        end if
        if There are new removed documents then
            Remove the documents from the user consistency zones
        end if
    end if
else
    Get the new and removed children
    if There is a removed document then
        Remove the pivot/users of the parent document from the removed document
    end if
    if There is a new document then
        Add the children to the consistency zones of the parent users.
    end if
end if
```

## 4.3 Summary

In this section we have addressed the specific details of the implementation of both wiki and the web cache, presenting the classes that make up the domain of both wiki and web cache, some classes involved in the implementation of the algorithms for the wiki and web cache operation.

Also, for the web cache we presented some of the major libraries that were used and why they were used.

# 5 Evaluation

After discussing the architecture, the algorithms and the relevant implementation details, we are going to evaluate the project in quantitative, qualitative and comparative terms. In the quantitative subsection, we are going to show screenshots of VFC-Cache working, in the qualitative subsection, we are going to compare VFC-Cache while using VFC and the standard http consistency protocol. In the comparative terms, we are going to compare VFC-Cache with other caches, like Squid and Polipo, in different configuration scenarios.

In the quantitative and comparative tests, we are specially focused in reducing the number of uninteresting cached pages, in order to be able to cache more interesting pages, since that is the main aim of the VFC algorithm.

All of the quantitative and comparative tests, were done using an automated tool and a list of visited urls (in order to simulate what a real user would do) together with a list of bookmarks selected from those urls. Then, several features were measured for each of the tested caches. With the numbers about the number of cache-hits, cache-misses and page state being generated from the logs, of each cache (in this cache our developed cache and squid).

In the quantitative section, we will also present a test case using fifa98 website access log(Arlitt & Jin 1998), in which we consider a population of 21 users that access a common url set and analyse the gains that our VFC cache could offer given that scenario.

## 5.1 Qualitative Evaluation

Like told above, in this section we are going to show how our developed solution works and we will start by the VFC-Cache, specifically by showing the aspect of the developed client plugin, that is shown in figures 5.1, 23, 5.2, 24 and 25.

So that, in the first image we have the global aspect of the menu of the plugin, in the second image we have the menu entry to edit some general preferences like the address of the proxy url, the user name and the addon version.

In figure 23, we have the help section of our plugin, in figure 24 we have the consistency zone definitions that are set by the user and finally the list of user bookmarks.

In the terms of the VFC-Cache working, in figure 5.3 we have an example of a request to a document that is present in one of the consistency zones of the user, but it has not been yet retrieved, since no request for it has arrived.

In the figure 5.4, we have an example of a request to the same document, but since that document is in cache and is fresh, the cached document is used to service the user request.

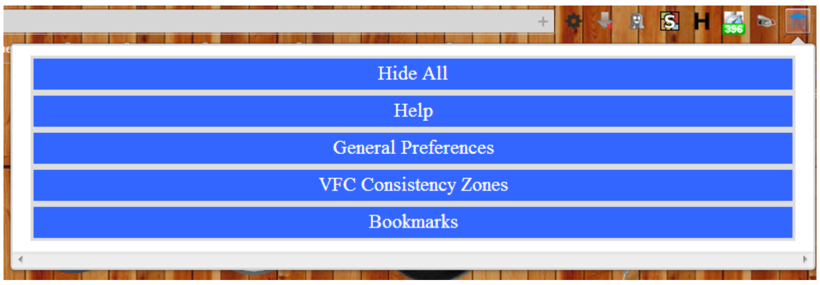
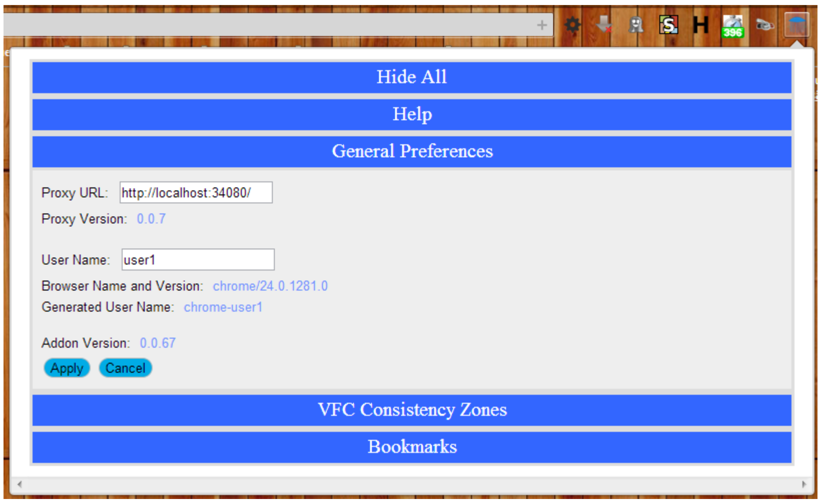


Figure 5.1: Initial aspect of the VFC-Cache client plugin

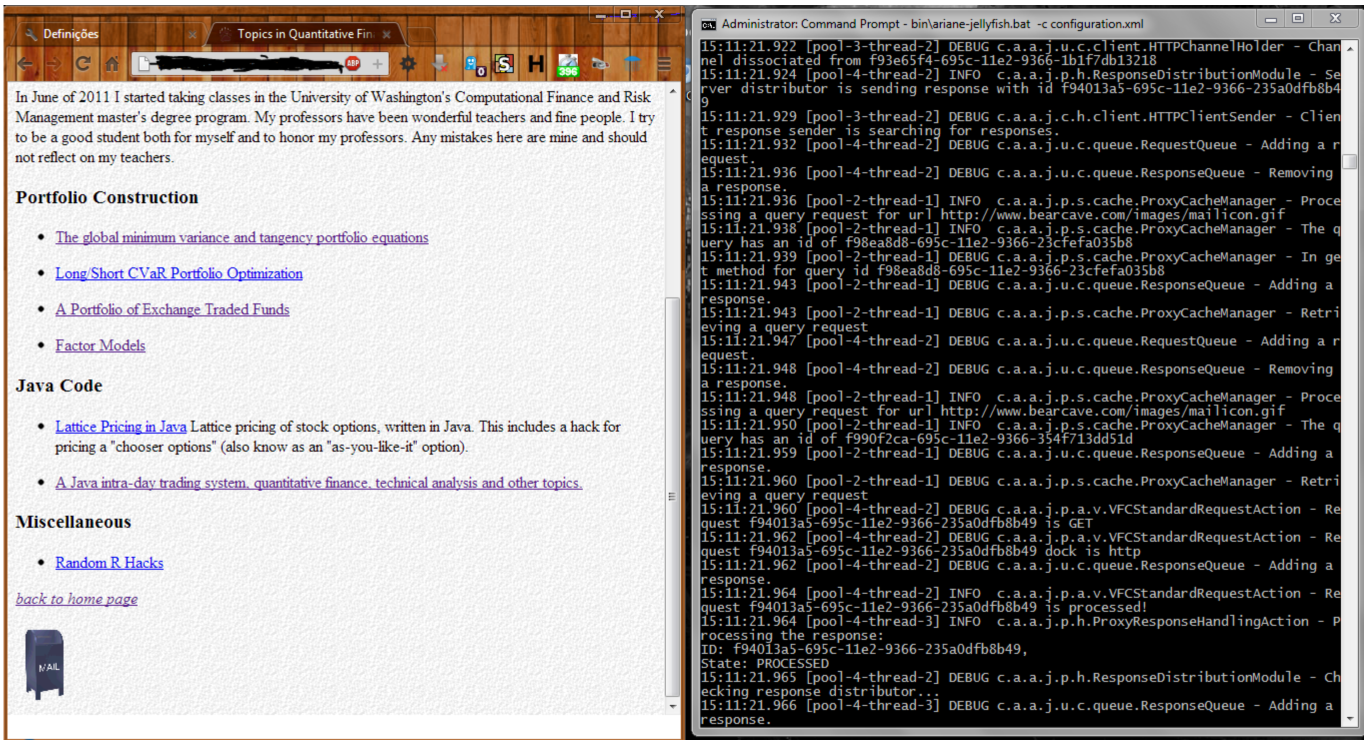


Proxy URL: http://localhost:34080/
Proxy Version: 0.0.7

User Name: user1
Browser Name and Version: chrome/24.0.1281.0
Generated User Name: chrome-user1

Addon Version: 0.0.67

Figure 5.2: General section of the VFC-Cache client plugin



[redacted]

In June of 2011 I started taking classes in the University of Washington's Computational Finance and Risk Management master's degree program. My professors have been wonderful teachers and fine people. I try to be a good student both for myself and to honor my professors. Any mistakes here are mine and should not reflect on my teachers.

**Portfolio Construction**

- The global minimum variance and tangency portfolio equations
- Long/Short CVaR Portfolio Optimization
- A Portfolio of Exchange Traded Funds
- Factor Models

**Java Code**

- Lattice Pricing in Java Lattice pricing of stock options, written in Java. This includes a hack for pricing a "chooser options" (also know as an "as-you-like-it" option).
- A Java intra-day trading system, quantitative finance, technical analysis and other topics.

**Miscellaneous**

- Random R Hacks

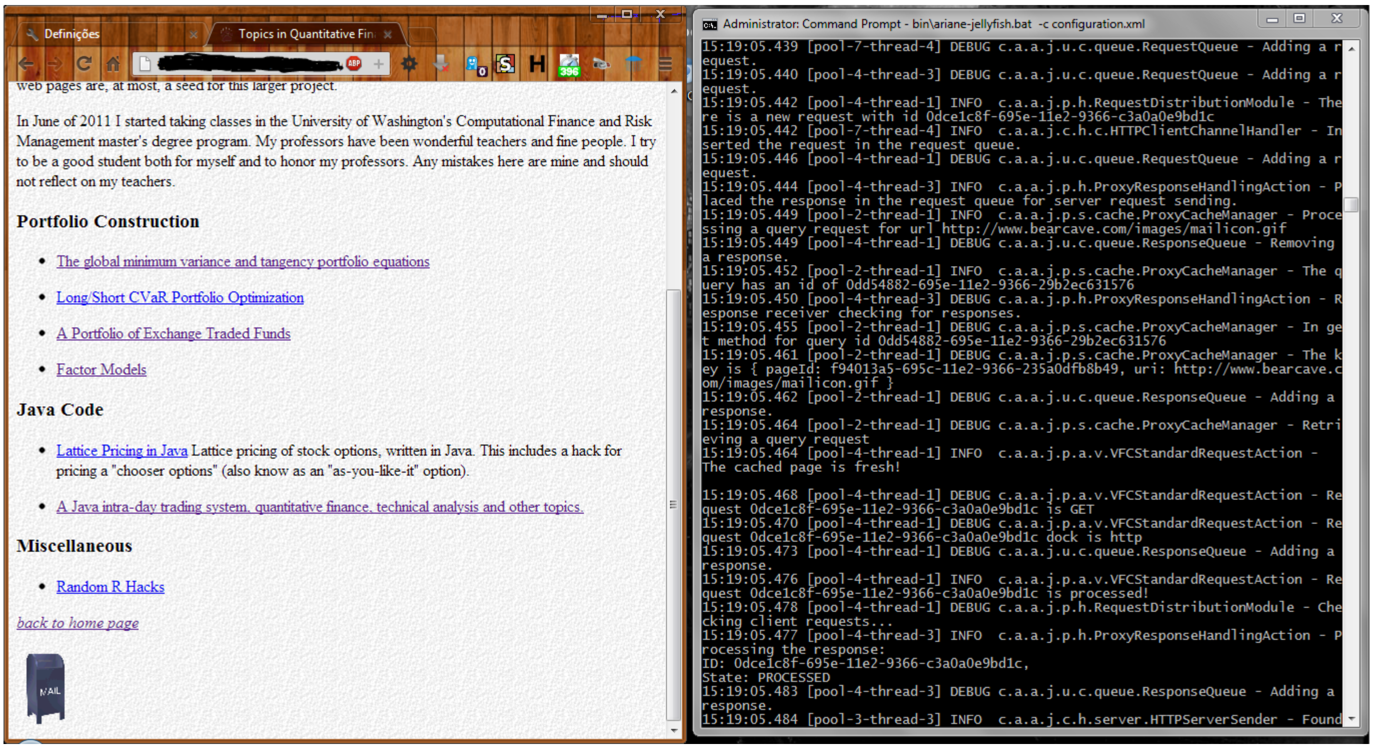
*back to home page*

Administrator: Command Prompt - bin\ariane-jellyfish.bat -c configuration.xml

```
15:11:21.922 [pool-3-thread-2] DEBUG c.a.a.j.u.c.client.HTTPChannelHolder - Chan
nel dissociated from f93e65f4-695c-11e2-9366-1b1f7db13218
15:11:21.924 [pool-4-thread-2] INFO  c.a.a.j.p.h.ResponseDistributionModule - Se
rver distributor is sending response with id f94013a5-695c-11e2-9366-235a0dfb8b4
9
15:11:21.929 [pool-3-thread-2] DEBUG c.a.a.j.c.h.client.HTTPClientSender - Clien
t response sender is searching for responses.
15:11:21.932 [pool-4-thread-2] DEBUG c.a.a.j.u.c.queue.RequestQueue - Adding a r
equest.
15:11:21.936 [pool-4-thread-2] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Removing
a response.
15:11:21.936 [pool-2-thread-1] INFO  c.a.a.j.p.s.cache.ProxyCacheManager - Proce
ssing a query request for url http://www.bearcave.com/images/mailicon.gif
15:11:21.938 [pool-2-thread-1] INFO  c.a.a.j.p.s.cache.ProxyCacheManager - The q
uery has an id of f98ea8d8-695c-11e2-9366-23cfefa035b8
15:11:21.939 [pool-2-thread-1] DEBUG c.a.a.j.p.s.cache.ProxyCacheManager - In ge
t method for query id f98ea8d8-695c-11e2-9366-23cfefa035b8
15:11:21.943 [pool-2-thread-1] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Adding a
response.
15:11:21.943 [pool-2-thread-1] DEBUG c.a.a.j.p.s.cache.ProxyCacheManager - Retri
eving a query request
15:11:21.947 [pool-4-thread-2] DEBUG c.a.a.j.u.c.queue.RequestQueue - Adding a r
equest.
15:11:21.948 [pool-4-thread-2] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Removing
a response.
15:11:21.948 [pool-2-thread-1] INFO  c.a.a.j.p.s.cache.ProxyCacheManager - Proce
ssing a query request for url http://www.bearcave.com/images/mailicon.gif
15:11:21.950 [pool-2-thread-1] INFO  c.a.a.j.p.s.cache.ProxyCacheManager - The q
uery has an id of f990f2ca-695c-11e2-9366-354f713dd51d
15:11:21.959 [pool-2-thread-1] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Adding a
response.
15:11:21.960 [pool-2-thread-1] DEBUG c.a.a.j.p.s.cache.ProxyCacheManager - Retri
eving a query request
15:11:21.960 [pool-4-thread-2] DEBUG c.a.a.j.p.a.v.VFCStandardRequestAction - Re
quest f94013a5-695c-11e2-9366-235a0dfb8b49 is GET
15:11:21.962 [pool-4-thread-2] DEBUG c.a.a.j.p.a.v.VFCStandardRequestAction - Re
quest f94013a5-695c-11e2-9366-235a0dfb8b49 dock is http
15:11:21.962 [pool-4-thread-2] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Adding a
response.
15:11:21.964 [pool-4-thread-2] INFO  c.a.a.j.p.a.v.VFCStandardRequestAction - Re
quest f94013a5-695c-11e2-9366-235a0dfb8b49 is processed!
15:11:21.964 [pool-4-thread-3] INFO  c.a.a.j.p.h.ProxyResponseHandlingAction - P
rocessing the response:
ID: f94013a5-695c-11e2-9366-235a0dfb8b49,
State: PROCESSED
15:11:21.965 [pool-4-thread-2] DEBUG c.a.a.j.p.h.ResponseDistributionModule - Ch
ecking response distributor...
15:11:21.966 [pool-4-thread-3] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Adding a
response.
```

Figure 5.3: VFC-Cache server, dealing with a request for the first time, meaning that since it does not have it, the request has to go to the server.

So in the first figure, the request is sent to the server while in the second figure the request is done by the cache and therefore the image shows an debug line with ”The cached page is fresh”.



web pages are, at most, a seed for this larger project.

In June of 2011 I started taking classes in the University of Washington's Computational Finance and Risk Management master's degree program. My professors have been wonderful teachers and fine people. I try to be a good student both for myself and to honor my professors. Any mistakes here are mine and should not reflect on my teachers.

**Portfolio Construction**

- The global minimum variance and tangency portfolio equations
- Long/Short CVaR Portfolio Optimization
- A Portfolio of Exchange Traded Funds
- Factor Models

**Java Code**

- Lattice Pricing in Java Lattice pricing of stock options, written in Java. This includes a hack for pricing a "chooser options" (also know as an "as-you-like-it" option).
- A Java intra-day trading system, quantitative finance, technical analysis and other topics.

**Miscellaneous**

- Random R Hacks

*back to home page*

Administrator: Command Prompt - bin\ariane-jellyfish.bat -c configuration.xml

```
15:19:05.439 [pool-7-thread-4] DEBUG c.a.a.j.u.c.queue.RequestQueue - Adding a r
equest.
15:19:05.440 [pool-4-thread-3] DEBUG c.a.a.j.u.c.queue.RequestQueue - Adding a r
equest.
15:19:05.442 [pool-4-thread-1] INFO  c.a.a.j.p.h.RequestDistributionModule - The
re is a new request with id 0dce1c8f-695e-11e2-9366-c3a0a0e9bd1c
15:19:05.442 [pool-7-thread-4] INFO  c.a.a.j.c.h.c.HTTPClientChannelHandler - In
serted the request in the request queue.
15:19:05.446 [pool-4-thread-1] DEBUG c.a.a.j.u.c.queue.RequestQueue - Adding a r
equest.
15:19:05.444 [pool-4-thread-3] INFO  c.a.a.j.p.h.ProxyResponseHandlingAction - P
laced the response in the request queue for server request sending.
15:19:05.449 [pool-2-thread-1] INFO  c.a.a.j.p.s.cache.ProxyCacheManager - Proce
ssing a query request for url http://www.bearcave.com/images/mailicon.gif
15:19:05.449 [pool-4-thread-1] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Removing
a response.
15:19:05.452 [pool-2-thread-1] INFO  c.a.a.j.p.s.cache.ProxyCacheManager - The q
uery has an id of 0dd54882-695e-11e2-9366-29b2ec631576
15:19:05.450 [pool-4-thread-3] DEBUG c.a.a.j.p.h.ProxyResponseHandlingAction - R
esponse receiver checking for responses.
15:19:05.455 [pool-2-thread-1] DEBUG c.a.a.j.p.s.cache.ProxyCacheManager - In ge
t method for query id 0dd54882-695e-11e2-9366-29b2ec631576
15:19:05.461 [pool-2-thread-1] DEBUG c.a.a.j.p.s.cache.ProxyCacheManager - The k
ey is { pageId: f94013a5-695c-11e2-9366-235a0dfb8b49, uri: http://www.bearcave.c
om/images/mailicon.gif }
15:19:05.462 [pool-2-thread-1] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Adding a
response.
15:19:05.464 [pool-2-thread-1] DEBUG c.a.a.j.p.s.cache.ProxyCacheManager - Retri
eving a query request
15:19:05.464 [pool-4-thread-1] INFO  c.a.a.j.p.a.v.VFCStandardRequestAction -
The cached page is fresh!

15:19:05.468 [pool-4-thread-1] DEBUG c.a.a.j.p.a.v.VFCStandardRequestAction - Re
quest 0dce1c8f-695e-11e2-9366-c3a0a0e9bd1c is GET
15:19:05.470 [pool-4-thread-1] DEBUG c.a.a.j.p.a.v.VFCStandardRequestAction - Re
quest 0dce1c8f-695e-11e2-9366-c3a0a0e9bd1c dock is http
15:19:05.473 [pool-4-thread-1] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Adding a
response.
15:19:05.476 [pool-4-thread-1] INFO  c.a.a.j.p.a.v.VFCStandardRequestAction - Re
quest 0dce1c8f-695e-11e2-9366-c3a0a0e9bd1c is processed!
15:19:05.478 [pool-4-thread-1] DEBUG c.a.a.j.p.h.RequestDistributionModule - Che
cking client requests...
15:19:05.477 [pool-4-thread-3] INFO  c.a.a.j.p.h.ProxyResponseHandlingAction - P
rocessing the response:
ID: 0dce1c8f-695e-11e2-9366-c3a0a0e9bd1c,
State: PROCESSED
15:19:05.483 [pool-4-thread-3] DEBUG c.a.a.j.u.c.queue.ResponseQueue - Adding a
response.
15:19:05.484 [pool-3-thread-3] INFO  c.a.a.j.c.h.server.HTTPServerSender - Found
```

Figure 5.4: VFC-Cache server, dealing with a request for the subsequent time, meaning that since the cache page is fresh, the web cache can imeadiatly send the cached page and therefore our processor prints “The cached page is fresh” on its log.

## 5.2 Quantitative Evaluation

In this section, we will present and discuss two types of tests, one that is done using an automated browser tool, with the dataset presented at the appendix and the other using the fifa 1998 website access logs, with a specific scenario, presented in that subsection.

### 5.2.1 Automated Tests

In this subsection we are going to compare the VFC scheme with the regular http scheme using the developed base cache, focusing on the number of cached documents, with the dataset of pages used for both the quantitative and comparative tests being listed in the appendix.

So in the ocupation of the cache memory (Figure 5.5), we see that one of the main objectives of VFC, that is to reduce the number of unimportant cached pages was successfully achieved, with a ratio of $0.26:1$ pages from VFC to standard http, meaning $26.08\%$ of pages or in overall less $74\%$. Which means that we have more space to store more interesting pages, using the same memory space as other caches or if the cache administrator likes, using a machine with less memory.

In terms of memory usage (Figures 5.6 and 5.7), we see that the differences are not so big, in spite of the fact that VFC has to parse all html pages and store extra information, when compared with the standard cache.

In terms of cpu usage (Figures 5.8 and 5.7) and considering that the tests were run in an quad-core machine (with 4 cpus), we see that the VFC algorithm has some extra processing, mostly due to the fact of having to parse the html and managing a set of consistency zones, which make the algorithm more cpu heavy. And this also reflects in the latency graphics shown below.

In terms of latency (Figure 5.10), we see that the VFC algorithm can reduce the load for the most used pages, in spite of behaving almost like a direct cache for the pages that are not bookmarks, which reflects that the bookmarks should be extremely well chosen and that the algorithm is more cpu heavy than the standard http caching algorithm.

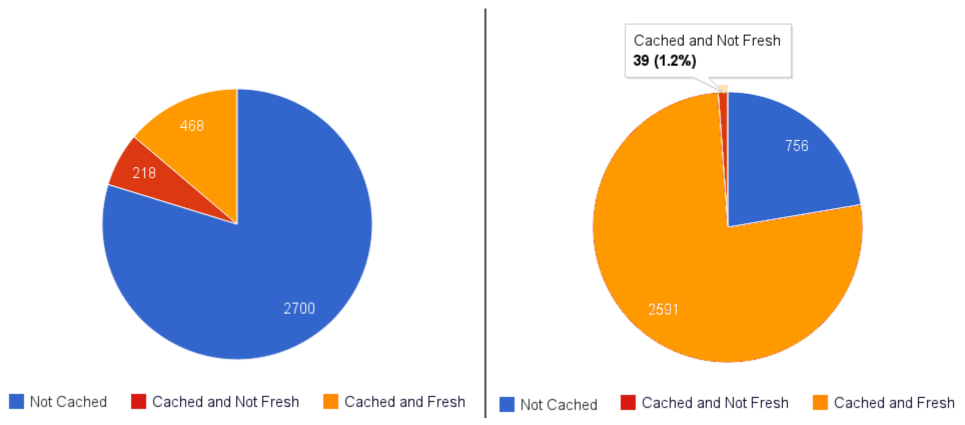


Figure 5.5: Number of cached pages, VFC on the left and standard http on the right

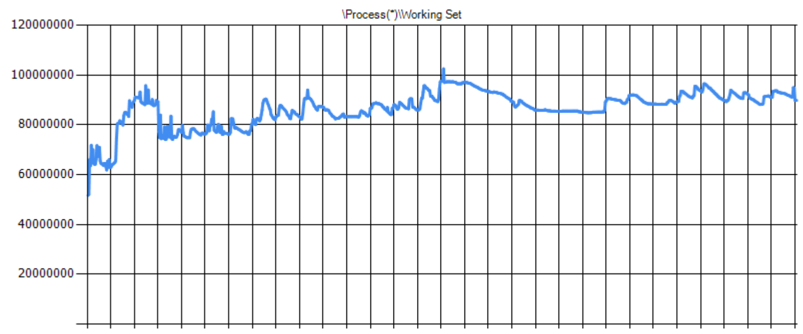


Figure 5.6: Memory usage of the VFC cache.

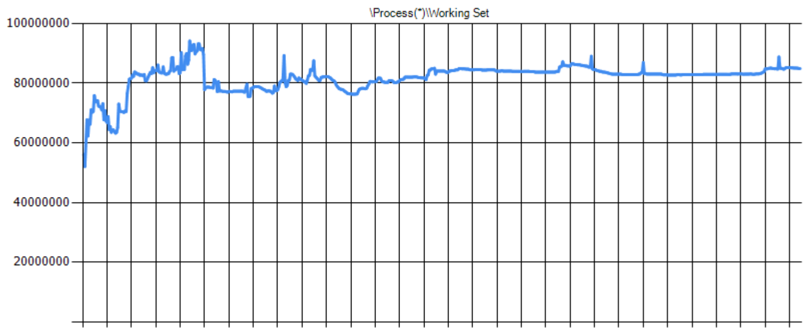


Figure 5.7: Memory usage of the standard cache.

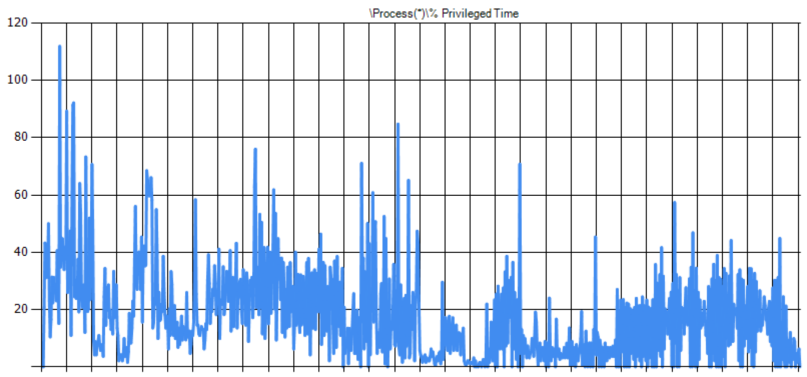


Figure 5.8: CPU usage of the VFC cache.

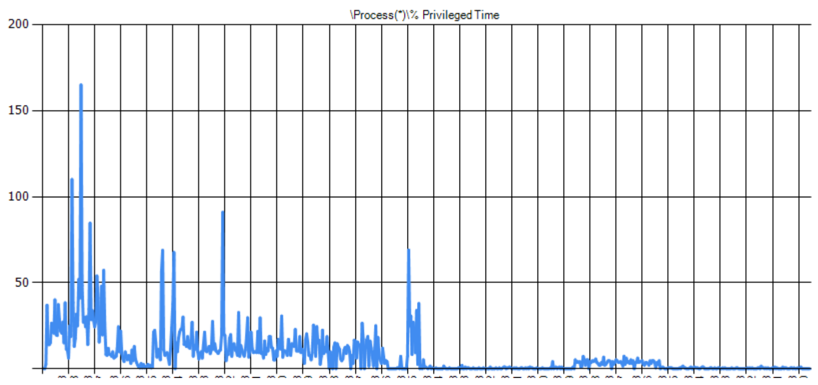


Figure 5.9: CPU usage of the standard cache.

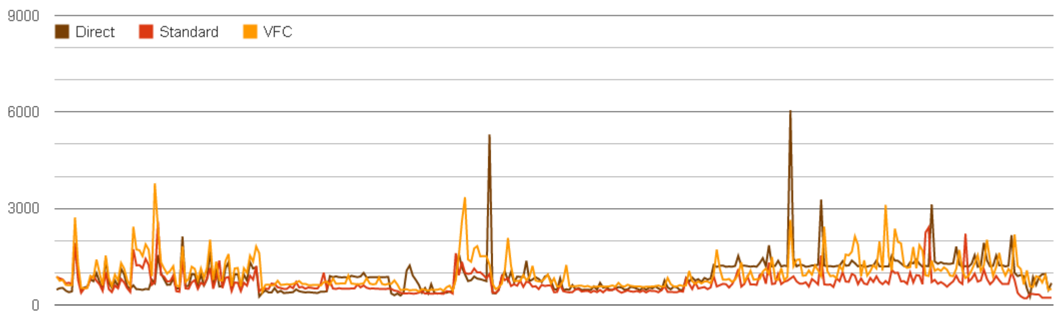


Figure 5.10: Latency of page loading, using a direct connection, a standard connection and the VFC algorithm

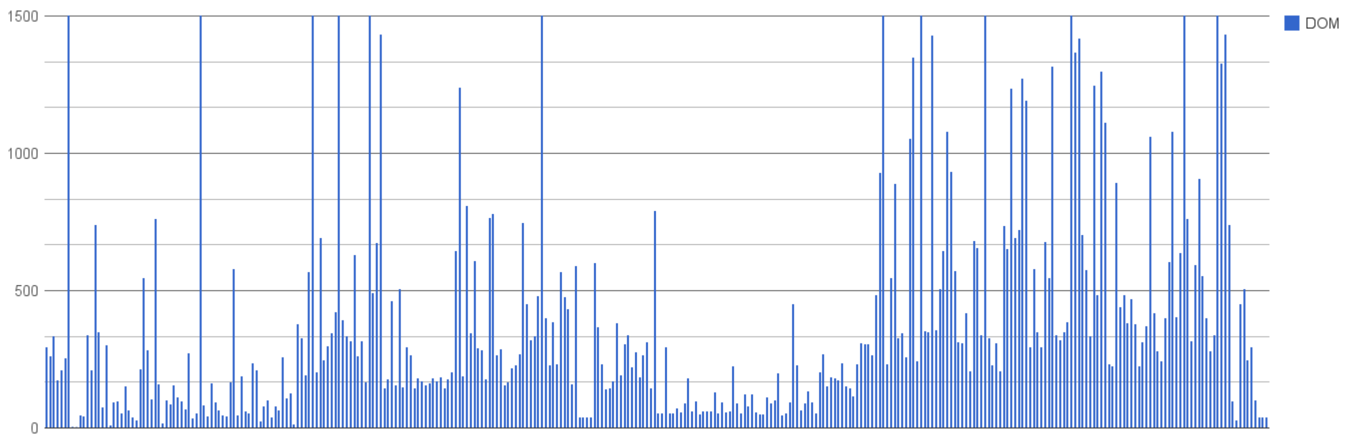


Figure 5.11: Number of DOM nodes

Finally, for comparison we present a graph (Figure 5.11) with the number of DOM nodes of the parsed HTML pages and that directly influence the VFC algorithm, since all of them have to be created in memory and parsed by the DOM HTML parser we used.

### 5.2.2 Fifa 1998 tests

In this subsection, we are going to present a synthetic test using the fifa 1998 website access log data. For this test we have considered the logs of July 15th of 1998, then we have filtered the first 80000 entries of that log (after a conversion to the common log format) and then we have selected users number 55, 60, 79, 111, 210, 245, 179, 366, 381, 388, 465, 595, 623, 638, 746, 998, 1509 and 1548 which we gave as common interest the fact that they are all interested in french pages and hence the their set of pivots involve pages that begin with the url "/french/*".

For this test we have compared the number and size of pages that a regular cache would be allowed to cache (in other words, everything), the number and size of pages that the VFC cache would be allowed to cache (in other words, everything with the prefix given above). From the second set we have also filtered the data in order to analyse the pages that would be wasted (in other words, pages that are cached but needed only one time) and pages that would not be wasted (in other words, pages that are cached and needed several times).

So, given the results presented in figures 5.12 and 5.13, we notice that although the number of wasted pages is high it is close to the number of not wasted pages, even if the number of wasted bytes is higher than the number of not wasted bytes.

But, if we take into attention the number of repeated accesses to the pages and multiply those by the size of those pages (whose cached state, has prevented the server and client from making a direct connection to the server), we get a number of 351795 bytes, which far surpasses the number of wasted bytes, making the cache eficient.

That contributes further to the conclusion that if well choosen a bookmark/pivot can balance the cost that it takes to have it in memory, by allowing an increased number of cache hits, relative to the regular case, thereby justifying the trade between the size of the store page and the number of cache hits.

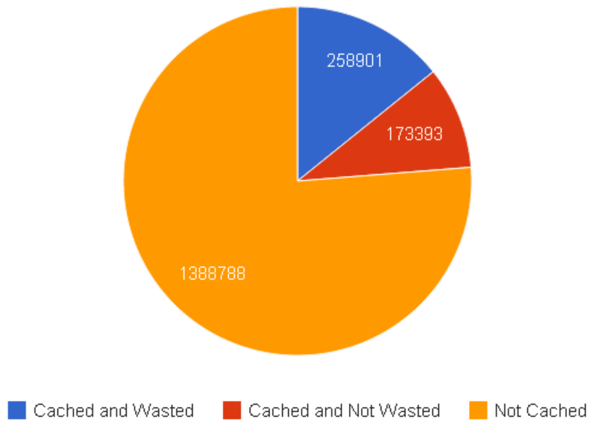


Figure 5.12: Results of the fifa 1998 test in bytes

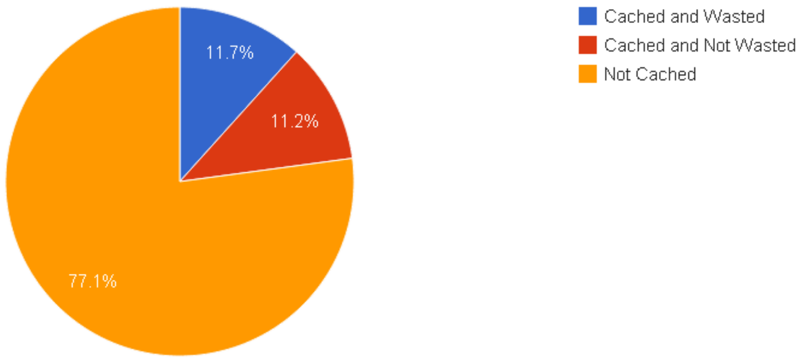


Figure 5.13: Results of the fifa 1998 test in number of pages

## 5.3 Comparative Evaluation

In this section, we will compare our solution against other existing solutions, specifically Squid 2, Squid 3 and Polipo, focusing in the number of cached documents.

So in the number of cached documents (Figures 5.5 (presented in the quantitative tests) and 5.14) and given that in the second figure, the first graphic (counting from the top left of the figure) is from Squid v2 using GDSF, next to the right we have Squid v2 using the LRU Heap, next to the right we have Squid v2 using the LFU-DA, in the next row, we have Squid v2 using LRU, then Squid v3 using GDSF, then Squid v3 using LRU Heap, in the next row we have Squid v3 using LFU-DA and finally Squid v3 using LRU.

We were unable to get any information from polipo, due to the fact that polipo does not produce any statistics of the number of cached pages, cache-hits and cache-misses. And once again we obtain satisfatory results in the number of cached pages, which shows that VFC might work in resulting the number of cached pages.

In terms of latency (Figure 5.15), we see that the VFC algorithm can reduce the load for the most used pages, in spite of behaving almost like a direct cache for the pages that are not bookmarks. Also our cache is unable to parse CSS files or javascript files which can force the browser to transfer more pages, particularly images that are used in the background of the page or as the background of buttons or other visual elements, which could be solved by either employing a regular cache on top of our VFC system, using the space saved by the VFC algorithm or partially by parsing the CSS files.

This is noticeable since in average most pages used in the test and in the bookmarks, retrieved at least two elements without the usage of the VFC algorithm, being mostly images and pieces of HTML pages, used in ads, that are produced using javascript.

In terms of memory usage and cpu usage (Figures 5.8, 5.6, 5.9, 5.7, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 in the appendix) this is where most differences are and these can be attributed to the fact that most of the commercial caches are fine tuned to extract the most performance out of the operations they do.

Also in the memory aspect, all of the other caches were coded in C or C++, while our implementation was coded in java, so the java virtual machine, simply allocated a predefined amount of memory independent of its usage or not by the program, even if we configured the cache storage subsystem to have similar sizes as the other caches, an maximum of 18MB.

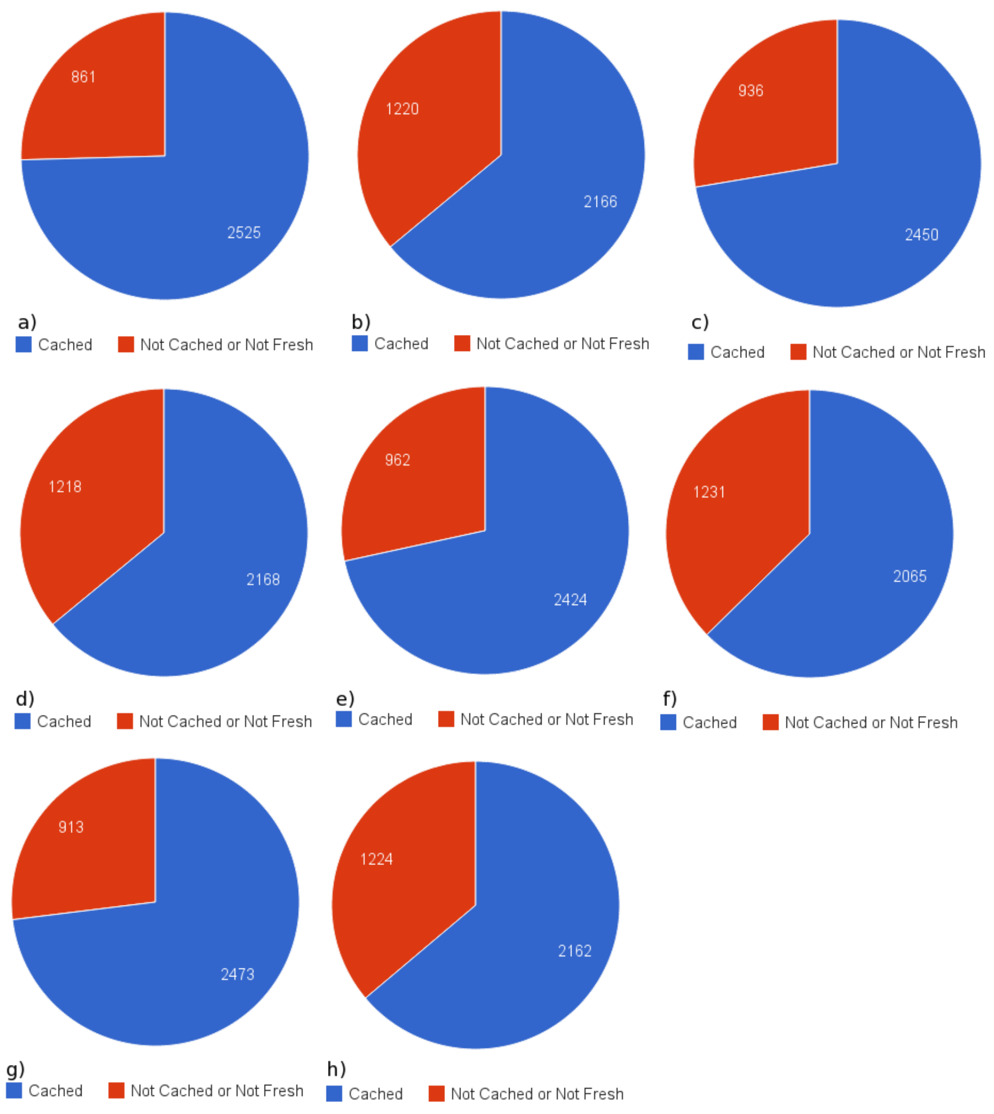


Figure 5.14: Cached pages of the other caches, where a) is Squid2 with GDSF, b) is Squid2 with LRU Heap, c) is Squid2 with LFU-DA, d) is Squid2 with LRU, e) is Squid3 with GDSF, f) is Squid3 with LRU Heap, g) is Squid3 with LFU-DA and h) is Squid 3 with LRU

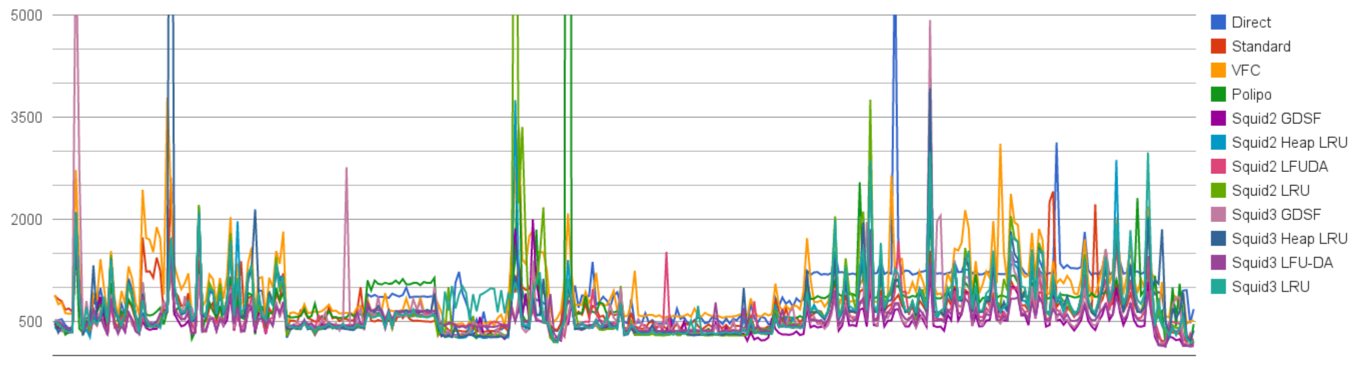


Figure 5.15: Latency of all caches.

## 5.4 Summary

In this evaluation we have performed some quantitative, qualitative and comparative tests and discussed their results with possible causes for some suboptimal performance verified in some of the tests.

We also confirmed that our cache system is able to store more pages in the same amount of cache memory, than in regular solutions and that if well choosen a bookmark can reduce the number of cache misses and stay more time in the cache memory and therefore in the long run, all our system is able to grant that all interesting pages to a user or user community stay in memory and prevent unnecessary communications with a web server, allowing for better bandwidth usage and less load in the origin web servers.

# 6 Conclusions

After having studied and presented some of the related work, where we analyzed the various architectures for a cache system, the various models for a proxy server, the two great families of cache consistency, some of the cache replacement strategies and finished with a taxonomy of the commercial cache servers according to the analyzed criteria.

Then, in the wiki systems, we classified the wiki types, the types of wiki users and some of the possible wiki architectures, ending with an explaination the VFC algorithm, that we intent to use in our cache solution, together with an overview of the past usages of the VFC algorithm.

In the next chapter, we described the architecture of our solution, where we analysed our adaptation of the VFC algorithm to the cases of both the wiki and the web cache, the deployment architecture of both cache and wiki system, the components of the cache and wiki and finally we described some use cases related to both the cache and the wiki.

Following that we provied some details about the implementation of our solution, where we have talked about the specific details of the implementation of both wiki and the web cache, presenting the classes that make up the domain of both wiki and web cache and some of the algorithms and classes involved in the wiki and web cache operations. For the web cache we presented some of the major libraries that were used and why they were used.

Next we followed to the evaluation tests, that were divided into qualitative, quantitative and comparative tests, some of them involving other caching solutions.

So in this conclusion we start by remembering the reader that our initial and most important aim, was to reduce the number of uninteresting cached pages, while keeping the working set of the user (if possible of a group of related users, like students, investigators of a department or employees of an enterprise) in cache in order to give the perception of a fast page load (or lower load latency) and to reduce the need to query the web server.

We also had the aim of doing the same for wiki pages, in order for a wiki moderator to keep an "eye" over a group of pages related to the domain of his knowledge.

Also while implementing we discovered some disadvantages of this method, some of them more important and serious than others, which we summarize below.

**Possibility of cycles in the graph of pages** Since we track, the links between pages it is possible that we follow a link to a page that is upper in the graph hierarchy, which results in the same pivot to be linked to a resource twice. Which while not very serious might complicate the allocation and removal algorithms, particularly if we use pointers to represent the elements in a tree;

**Serialization problems** Related to the problem above, we have serialization problems, since we must be careful not to serialize the same node twice and be careful while designing the serialization algorithm, since we must be able to detect cycles. Which lead us to move to a Java based solution instead of one based on C or C++;

**Problems with dynamic pages** Since we have to parse a page, our algorithm has problems with dynamic pages that use javascript to add or remove nodes from a HTML page, since we can only use the page without processing the javascript or else we had to implement a full headerless browser in the server which would increase the latency;

**Problems with CSS** Related to the problem above we have pages that use CSS to define elements like background images and component images (like buttons), since those elements might force the browser to retrieve more elements related to a page, that could have been cached;

**Freeform of the HTML** We also have a problem with the HTML pages, since usually HTML is not as strict as XML and some browsers also tolerate some semantic errors within HTML, which makes that our cache also has to behave like a browser while parsing pages and be as forgiving as the most relaxed browser in terms of HTML semantic and that adds to the processing time and memory amount since some of those errors require the reordering of some of the elements in the DOM tree, that we are forced to build instead of using something more lighter in terms of processing time and memory like SAX;

**Streaming and Flash content** There are also problems with content that is transmitted in real time like videos, music files and even live chat, but actually those problems also happen with most of the other caches, so most users are aware that for streaming content they cannot use a cache;

**Overral cost of cached page** Since there is a high cost for both the parsing and maintainable of pages in cache, since upon considered stale a page could have been potentially changed to an entirely different page with different links (rendering all of the existing connections useless), the algorithm is very sensitive to pages that represent unwanted ads (often present in a "interesting" page to a user) or bookmarked pages that are not really interesting to a user (maybe he or she bookmarked that page only to read it later just once).

In the field of contributions, our paper has shown that it is possible, albeit with some practical or implementation cost, to use bookmarks and the VFC algorithms in order to reduce the number of cached pages and the number of watched pages in a wiki, some of those practical shortcomings were reflected in the Evaluation that we have done, particularly when comparing our cache with other existing caches.

As future work, there is an aspect that can be further explored that is related to finding out what are the real interesting pages for a user, for example the frequency it is actually used to start a browsing session, since a bookmark may be a bookmark for several reasons, most of which are not related to a bookmark being interested in a way, that is a frequently visited page.

# Bibliography

(2001). *Web protocols and practice: HTTP/1.1, Networking protocols, caching, and traffic measurement*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Abdulla, G. (1998). *Analysis and modeling of world wide web traffic*. Ph. D. thesis. AAI9953804.

Abrams, M., C. R. Standridge, G. Abdulla, E. A. Fox, & S. Williams (1996, August). Removal policies in network caches for world-wide web documents. *SIGCOMM Comput. Commun. Rev. 26*(4), 293–305.

Abrams, M., C. R. Standridge, G. Abdulla, S. Williams, & E. A. Fox (1995). Caching proxies: Limitations and potentials. Technical report, Blacksburg, VA, USA.

Aggarwal, C., J. L. Wolf, & P. S. Yu (1999, January). Caching on the world wide web. *IEEE Trans. on Knowl. and Data Eng. 11*, 94–107.

Arlitt, M., L. Cherkasova, J. Dilley, R. Friedrich, & T. Jin (2000, March). Evaluating content management techniques for web proxy caches. *ACM SIGMETRICS Performance Evaluation Review 27*(4), 3–11.

Arlitt, M. & T. Jin (1998, August). 1998 world cup web site access logs.

Bryant, S. L., A. Forte, & A. Bruckman (2005). Becoming wikipedian: transformation of participation in a collaborative online encyclopedia. In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, GROUP '05, New York, NY, USA, pp. 1–10. ACM.

Cao, L. & M. Oezsu (2002). Evaluation of strong consistency web caching techniques. *World Wide Web 5*(2), 95–123.

Cao, P. & S. Irani (1997). Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, USITS'97, Berkeley, CA, USA, pp. 18–18. USENIX Association.

Cate, V. (1992). Alex-a global filesystem. In *Proceedings of the 1992 USENIX File System Workshop*, Number 7330, pp. 1–12. Citeseer.

Chang, C. & A. McGregor (1999). The LRU* WWW proxy cache document replacement algorithm.

Cobb, J. & H. ElAarag (2008, September). Web proxy cache replacement scheme based on back-propagation neural network. *J. Syst. Softw. 81*(9), 1539–1558.

Davison, B. D. (2001, July). A web caching primer. *IEEE Internet Computing 5*(4), 38–45.

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, & T. Berners-Lee (1999). RFC 2616: Hypertext transfer protocol–HTTP/1.1, June 1999. *Status: Standards Track 1*(11), 1829–1841.

Forte, A. & A. Bruckman (2007). Constructing text:: Wiki as a toolkit for (collaborative?) learning. In *Proceedings of the 2007 international symposium on Wikis*, WikiSym '07, New York, NY, USA, pp. 31–42. ACM.

Gray, C. & D. Cheriton (1989, November). Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. *SIGOPS Oper. Syst. Rev. 23*, 202–210.

Hosseini-Khayat, S. (1998). *Investigation of generalized caching*. Ph. D. thesis, St. Louis, MO, USA. UMI Order No. GAX98-07761.

Howard, J. H., M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, & M. J. West (1988, February). Scale and performance in a distributed file system. *ACM Transactions on Computer Systems 6*, 51–81.

Lee, B.-H., S.-H. Lim, J.-H. Kim, & G. C. Fox (2009). Lease-based consistency schemes in the web environment. *Future Generation Computer Systems 25*(1), 8 – 19.

Markatos, E. & C. Chronaki (1998). A top-10 approach to prefetching on the web. In *Proceedings of INET*, Volume 98, pp. 276–290.

Mogul, J. (2004). Clarifying the fundamentals of HTTP. *Software: Practice and Experience 34*(2), 103–134.

Morris, J. (2007). DistriWiki:: a distributed peer-to-peer wiki network. In *Proceedings of the 2007 international symposium on Wikis*, pp. 69–74. ACM.

Podlipnig, S. & L. Böszörmenyi (2003). A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR) 35*(4), 374–398.

Poole, E. S. & J. Grudin (2010). A taxonomy of wiki genres in enterprise settings. In *Proceedings of the 6th International Symposium on Wikis and Open Collaboration*, WikiSym '10, New York, NY, USA, pp. 14:1–14:4. ACM.

Qian, F., K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, & O. Spatscheck (2012). Web caching on smartphones: ideal vs. reality. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, New York, NY, USA, pp. 127–140. ACM.

Rabinovich, M. & O. Spatschek (2002). *Web caching and replication*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Rodriguez, P., C. Spanner, & E. Biersack (2001). Analysis of web caching architectures: hierarchical and distributed caching. *Networking, IEEE/ACM Transactions on 9*(4), 404–418.

Romano, S. & H. ElAarag (2008). A quantitative study of recency and frequency based web cache replacement strategies. In *Proceedings of the 11th communications and networking simulation symposium*, CNS '08, New York, NY, USA, pp. 70–78. ACM.

Santos, N., L. Veiga, & P. Ferreira (2007). Vector-field consistency for ad-hoc gaming. *Middleware 2007*, 80–100.

Urdaneta, G., G. Pierre, & M. Van Steen (2007). *A Decentralized Wiki Engine for Collaborative Wikipedia Hosting*, pp. 156–163. Citeseer.

Valloppillil, V. & K. W. Ross (1998). Cache array routing protocol v1.1. *1*.

Veiga, L., A. Negrão, N. Santos, & P. Ferreira (2010, August). Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency. *Journal of Internet Services and Applications 1*(2), 1–21.

Wang, J. (1999, October). A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review 29*, 36–46.

Wessels, D. & K. Claffy (1997). Application of internet cache protocol (icp), version 2. *RFC Editor United States* (2187).

Yin, J., L. Alvisi, M. Dahlin, & C. Lin (1998, may). Using leases to support server-driven consistency in large-scale systems. In *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on*, pp. 285–294. IEEE.

Yu, H., L. Breslau, & S. Shenker (1999, August). A scalable web cache consistency architecture. *SIGCOMM Comput. Commun. Rev. 29*, 163–174.

Zhang, J., R. Izmailov, D. Reininger, M. Ott, & N. U. S. A (1999). Web caching framework: Analytical models and beyond. In *Proceedings of the 1999 IEEE Workshop on Internet Applications*, WIAPP '99, Washington, DC, USA, pp. 132–. IEEE Computer Society.

# Appendices

## .1 Figures and Graphics

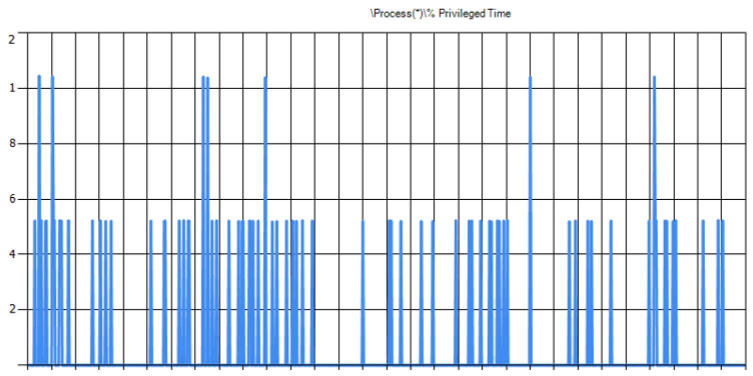


Figure 1: CPU usage of the polipo cache.

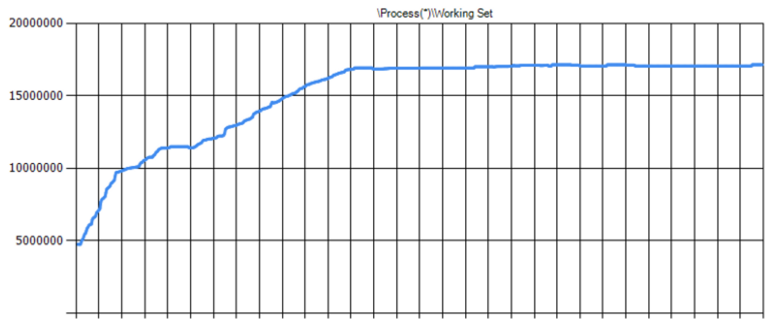


Figure 2: Memory usage of the polipo cache.

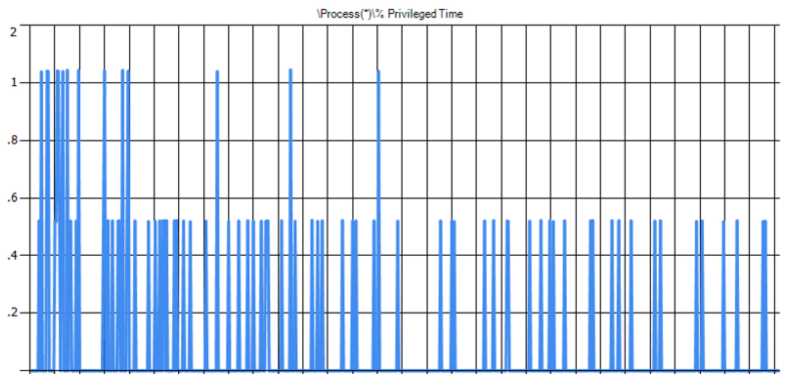


Figure 3: CPU usage of the Squid v2 using GDSF.

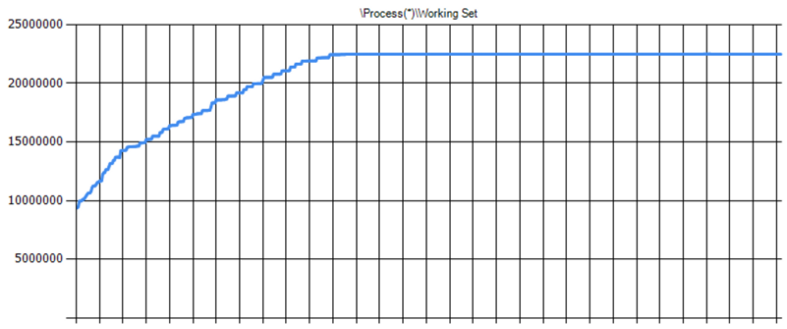


Figure 4: Memory usage of the Squid v2 using GDSF.

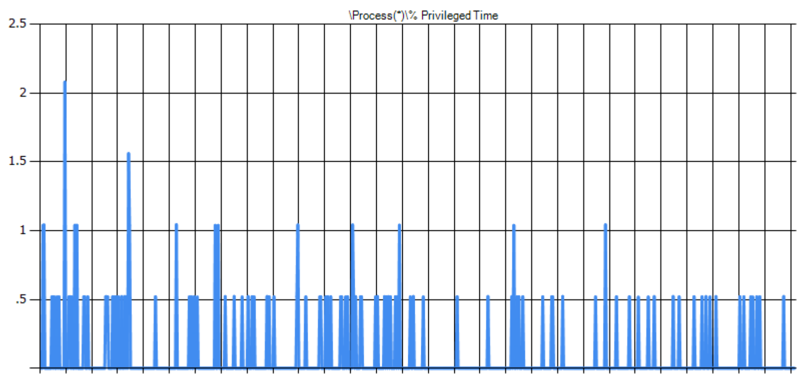


Figure 5: CPU usage of the Squid v2 using an Heap LRU.

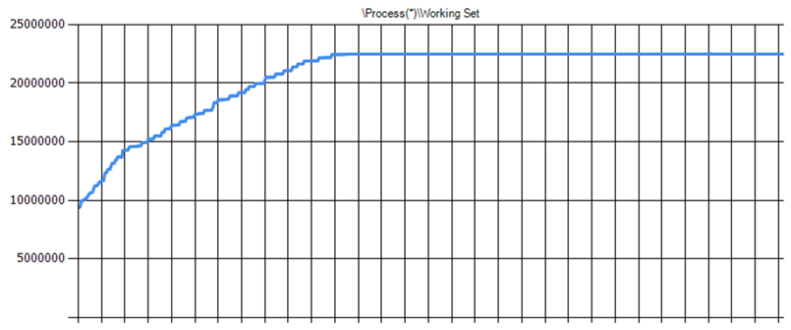


Figure 6: Memory usage of the Squid v2 using an Heap LRU.

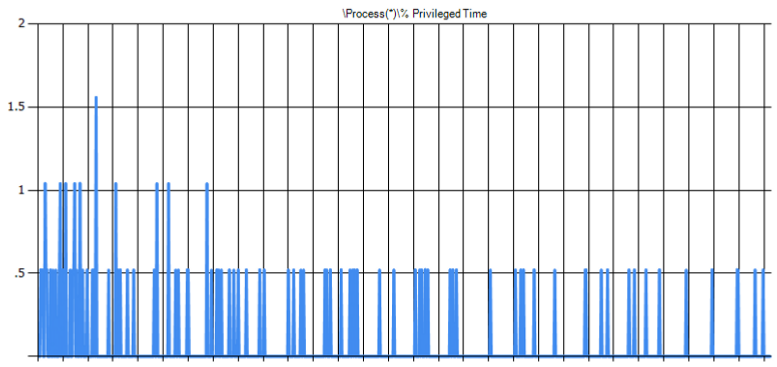


Figure 7: CPU usage of the Squid v2 using LFU-DA.

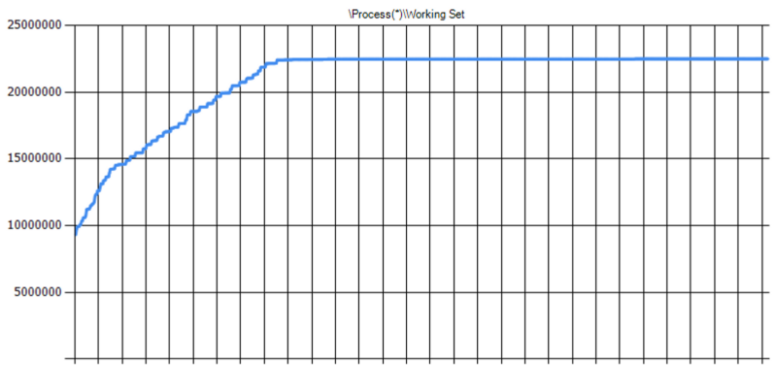


Figure 8: Memory usage of the Squid v2 using LFU-DA.

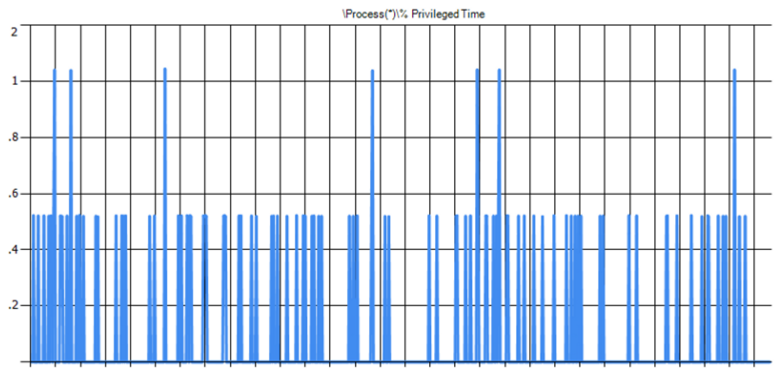


Figure 9: CPU usage of the Squid v2 using LRU.

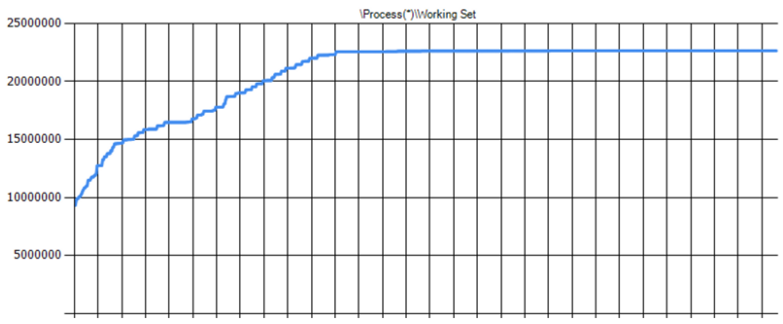


Figure 10: Memory usage of the Squid v2 using LRU.

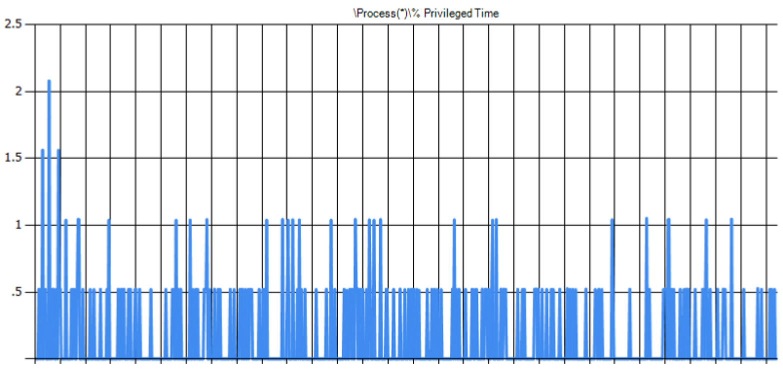


Figure 11: CPU usage of the Squid v3 using GDSF.

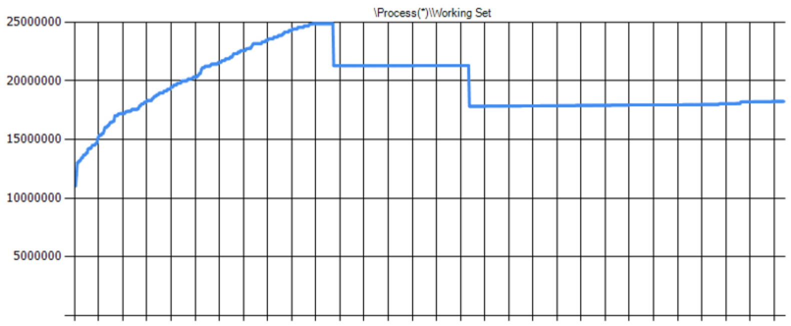


Figure 12: Memory usage of the Squid v3 using GDSF.

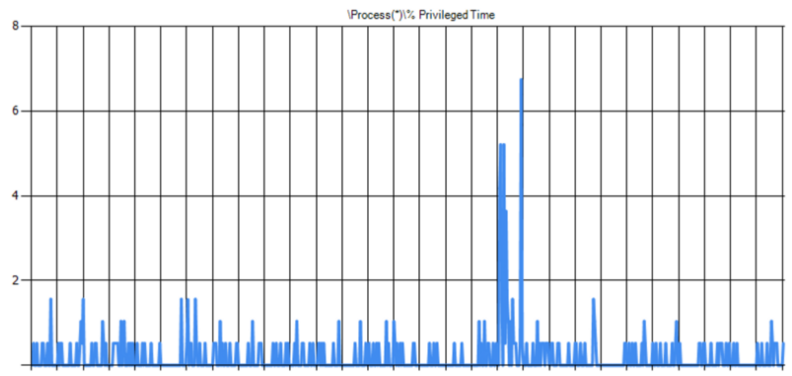


Figure 13: CPU usage of the Squid v3 using an Heap LRU.

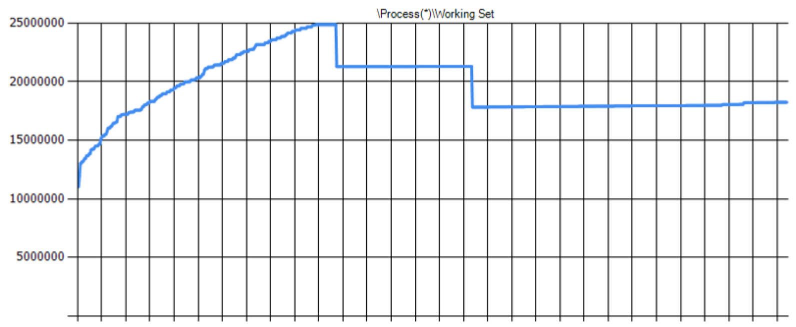


Figure 14: Memory usage of the Squid v3 using an Heap LRU.

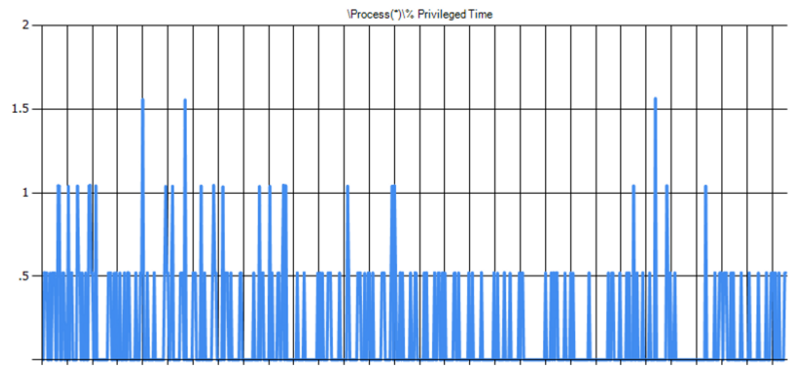


Figure 15: CPU usage of the Squid v3 using LFU-DA.

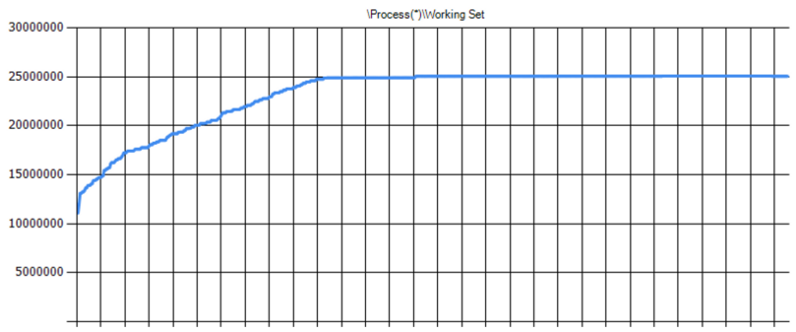


Figure 16: Memory usage of the Squid v3 using LFU-DA.

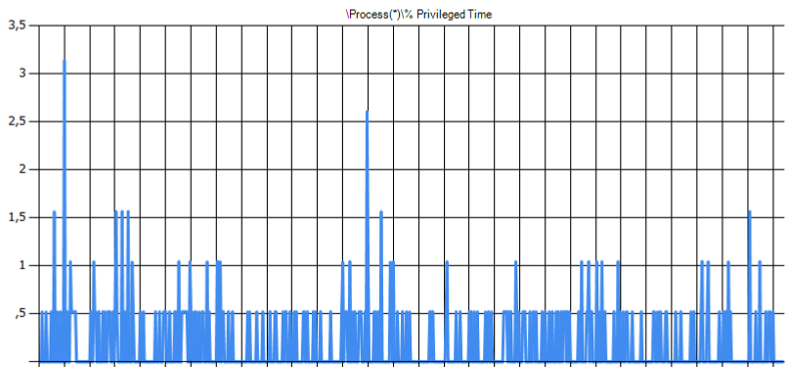


Figure 17: CPU usage of the Squid v3 using LRU.

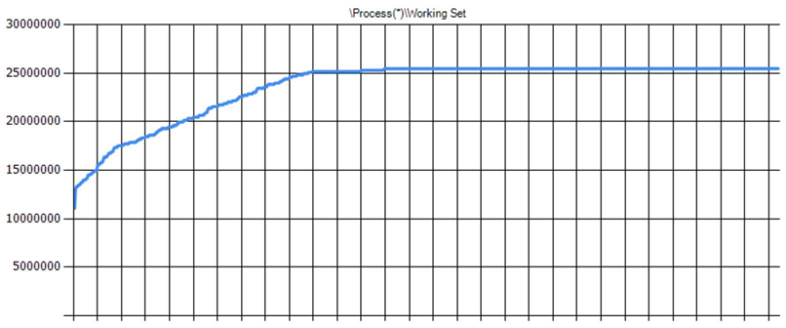


Figure 18: Memory usage of the Squid v3 using LRU.

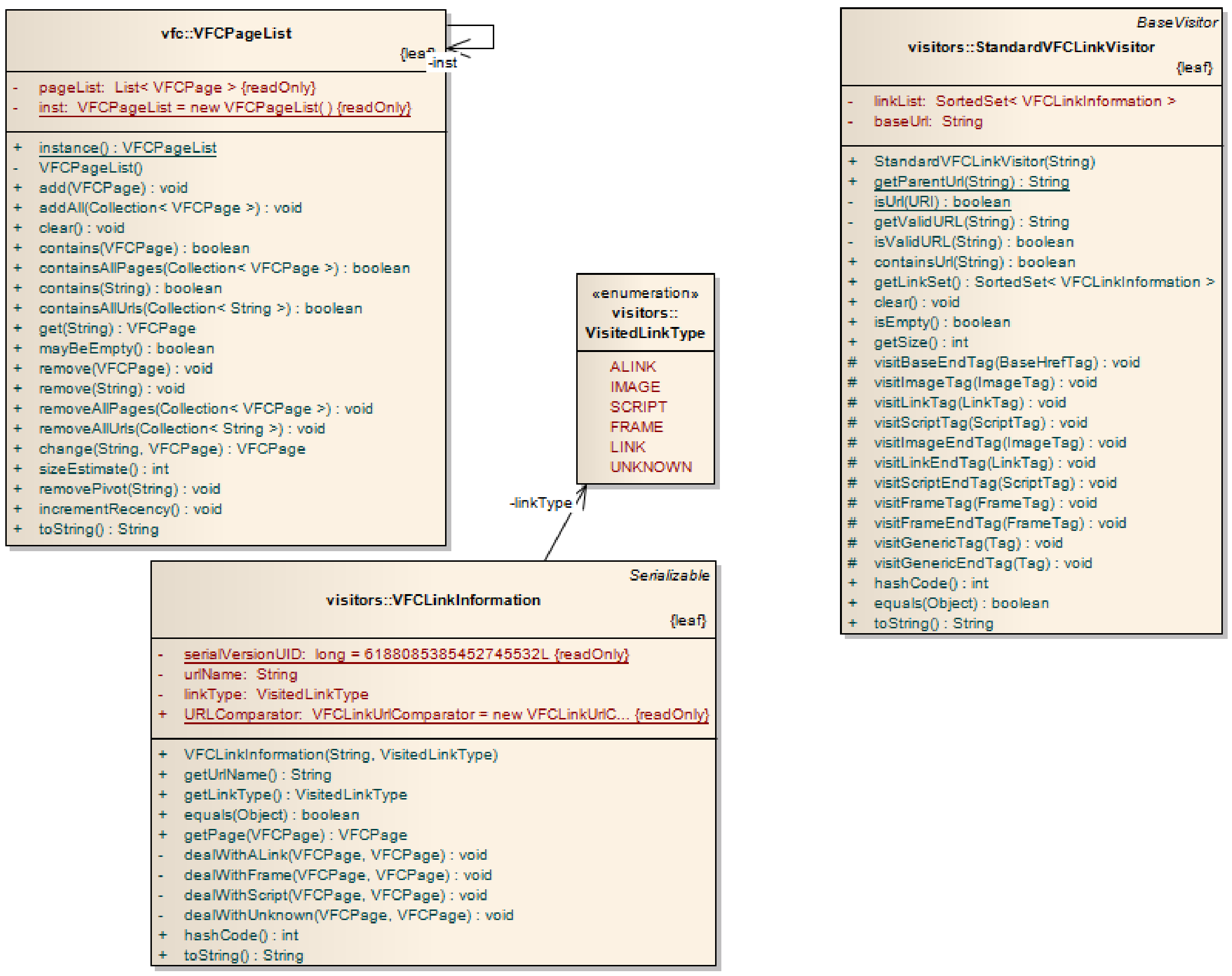

vfc::VFCPageList
{leaf}
-inst
- pageList: List< VFCPage > {readOnly}
- inst: VFCPageList = new VFCPageList( ) {readOnly}
+ instance() : VFCPageList
- VFCPageList()
+ add(VFCPage) : void
+ addAll(Collection< VFCPage >) : void
+ clear() : void
+ contains(VFCPage) : boolean
+ containsAllPages(Collection< VFCPage >) : boolean
+ contains(String) : boolean
+ containsAllUrls(Collection< String >) : boolean
+ get(String) : VFCPage
+ mayBeEmpty() : boolean
+ remove(VFCPage) : void
+ remove(String) : void
+ removeAllPages(Collection< VFCPage >) : void
+ removeAllUrls(Collection< String >) : void
+ change(String, VFCPage) : VFCPage
+ sizeEstimate() : int
+ removePivot(String) : void
+ incrementRecency() : void
+ toString() : String
«enumeration»
visitors::
VisitedLinkType
ALINK
IMAGE
SCRIPT
FRAME
LINK
UNKNOWN
-linkType
Serializable
visitors::VFCLinkInformation
{leaf}
- serialVersionUID: long = 6188085385452745532L {readOnly}
- urlName: String
- linkType: VisitedLinkType
+ URLComparator: VFCLinkUrlComparator = new VFCLinkUrlC... {readOnly}
+ VFCLinkInformation(String, VisitedLinkType)
+ getUrlName() : String
+ getLinkType() : VisitedLinkType
+ equals(Object) : boolean
+ getPage(VFCPage) : VFCPage
- dealWithALink(VFCPage, VFCPage) : void
- dealWithFrame(VFCPage, VFCPage) : void
- dealWithScript(VFCPage, VFCPage) : void
- dealWithUnknown(VFCPage, VFCPage) : void
+ hashCode() : int
+ toString() : String
BaseVisitor
visitors::StandardVFCLinkVisitor
{leaf}
- linkList: SortedSet< VFCLinkInformation >
- baseUrl: String
+ StandardVFCLinkVisitor(String)
+ getParentUrl(String) : String
- isUrl(URI) : boolean
- getValidURL(String) : String
- isValidURL(String) : boolean
+ containsUrl(String) : boolean
+ getLinkSet() : SortedSet< VFCLinkInformation >
+ clear() : void
+ isEmpty() : boolean
+ getSize() : int
# visitBaseEndTag(BaseHrefTag) : void
# visitImageTag(ImageTag) : void
# visitLinkTag(LinkTag) : void
# visitScriptTag(ScriptTag) : void
# visitImageEndTag(ImageTag) : void
# visitLinkEndTag(LinkTag) : void
# visitScriptEndTag(ScriptTag) : void
# visitFrameTag(FrameTag) : void
# visitFrameEndTag(FrameTag) : void
# visitGenericTag(Tag) : void
# visitGenericEndTag(Tag) : void
+ hashCode() : int
+ equals(Object) : boolean
+ toString() : String

Figure 19: Domain Classes - Part 1

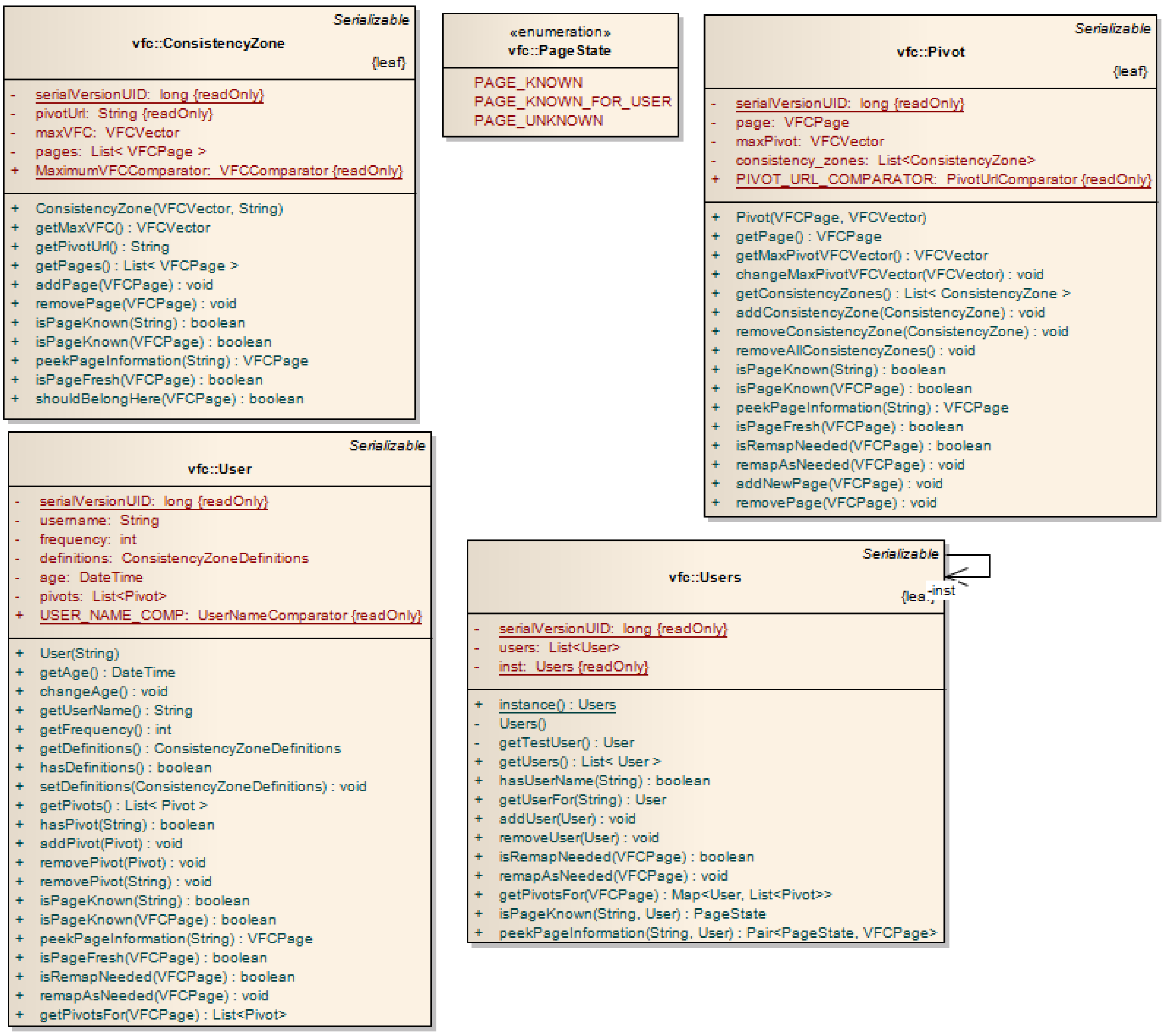

Serializable
vfc::ConsistencyZone
{leaf}
- serialVersionUID: long {readOnly}
- pivotUrl: String {readOnly}
- maxVFC: VFCVector
- pages: List< VFCPage >
+ MaximumVFCComparator: VFCComparator {readOnly}
+ ConsistencyZone(VFCVector, String)
+ getMaxVFC() : VFCVector
+ getPivotUrl() : String
+ getPages() : List< VFCPage >
+ addPage(VFCPage) : void
+ removePage(VFCPage) : void
+ isPageKnown(String) : boolean
+ isPageKnown(VFCPage) : boolean
+ peekPageInformation(String) : VFCPage
+ isPageFresh(VFCPage) : boolean
+ shouldBelongHere(VFCPage) : boolean
«enumeration»
vfc::PageState
PAGE_KNOWN
PAGE_KNOWN_FOR_USER
PAGE_UNKNOWN
Serializable
vfc::Pivot
{leaf}
- serialVersionUID: long {readOnly}
- page: VFCPage
- maxPivot: VFCVector
- consistency_zones: List<ConsistencyZone>
+ PIVOT_URL_COMPARATOR: PivotUrlComparator {readOnly}
+ Pivot(VFCPage, VFCVector)
+ getPage() : VFCPage
+ getMaxPivotVFCVector() : VFCVector
+ changeMaxPivotVFCVector(VFCVector) : void
+ getConsistencyZones() : List< ConsistencyZone >
+ addConsistencyZone(ConsistencyZone) : void
+ removeConsistencyZone(ConsistencyZone) : void
+ removeAllConsistencyZones() : void
+ isPageKnown(String) : boolean
+ isPageKnown(VFCPage) : boolean
+ peekPageInformation(String) : VFCPage
+ isPageFresh(VFCPage) : boolean
+ isRemapNeeded(VFCPage) : boolean
+ remapAsNeeded(VFCPage) : void
+ addNewPage(VFCPage) : void
+ removePage(VFCPage) : void
Serializable
vfc::User
- serialVersionUID: long {readOnly}
- username: String
- frequency: int
- definitions: ConsistencyZoneDefinitions
- age: DateTime
- pivots: List<Pivot>
+ USER_NAME_COMP: UserNameComparator {readOnly}
+ User(String)
+ getAge() : DateTime
+ changeAge() : void
+ getUserName() : String
+ getFrequency() : int
+ getDefinitions() : ConsistencyZoneDefinitions
+ hasDefinitions() : boolean
+ setDefinitions(ConsistencyZoneDefinitions) : void
+ getPivots() : List< Pivot >
+ hasPivot(String) : boolean
+ addPivot(Pivot) : void
+ removePivot(Pivot) : void
+ removePivot(String) : void
+ isPageKnown(String) : boolean
+ isPageKnown(VFCPage) : boolean
+ peekPageInformation(String) : VFCPage
+ isPageFresh(VFCPage) : boolean
+ isRemapNeeded(VFCPage) : boolean
+ remapAsNeeded(VFCPage) : void
+ getPivotsFor(VFCPage) : List<Pivot>
Serializable
vfc::Users
{lea
-inst
- serialVersionUID: long {readOnly}
- users: List<User>
- inst: Users {readOnly}
+ instance() : Users
- Users()
- getTestUser() : User
+ getUsers() : List< User >
+ hasUserName(String) : boolean
+ getUserFor(String) : User
+ addUser(User) : void
+ removeUser(User) : void
+ isRemapNeeded(VFCPage) : boolean
+ remapAsNeeded(VFCPage) : void
+ getPivotsFor(VFCPage) : Map<User, List<Pivot>>
+ isPageKnown(String, User) : PageState
+ peekPageInformation(String, User) : Pair<PageState, VFCPage>

Figure 20: Domain Classes - Part 2

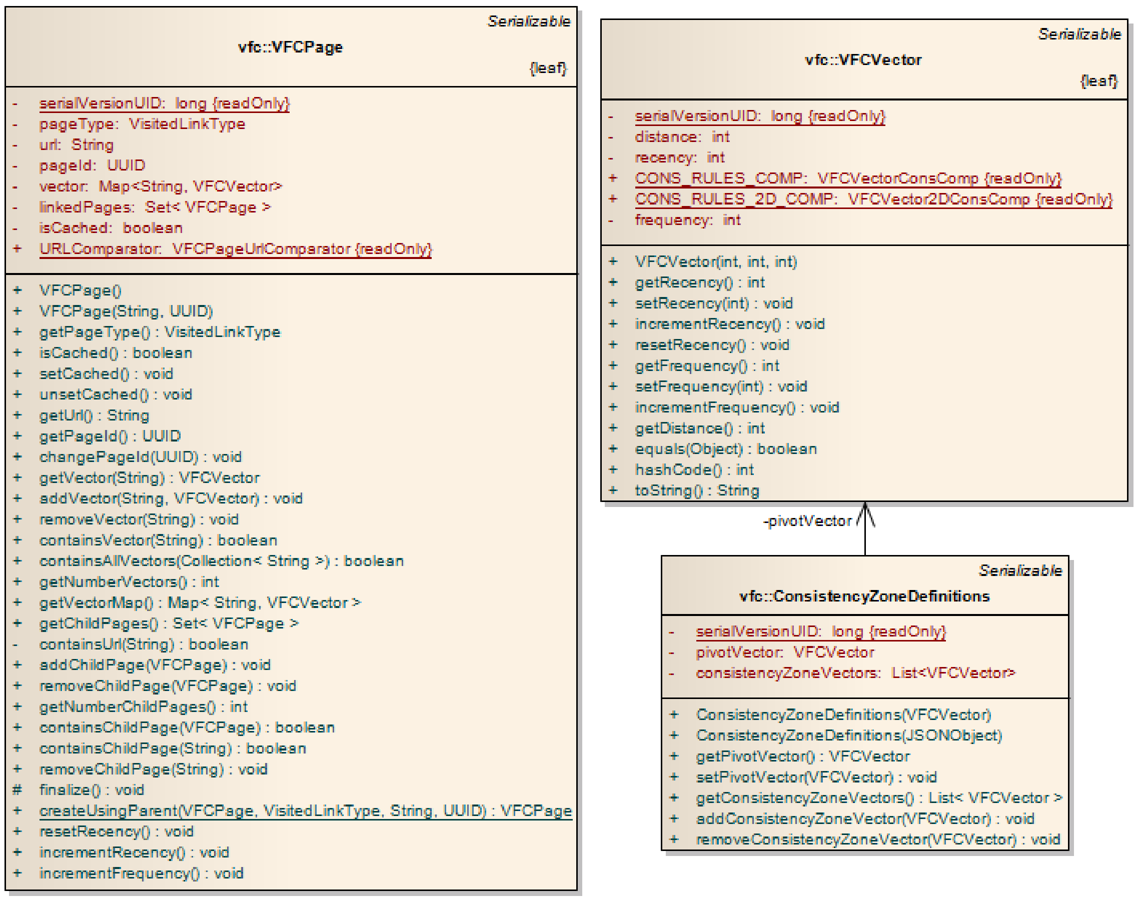

Serializable
vfc::VFCPage
{leaf}
- serialVersionUID: long {readOnly}
- pageType: VisitedLinkType
- url: String
- pageId: UUID
- vector: Map<String, VFCVector>
- linkedPages: Set< VFCPage >
- isCached: boolean
+ URLComparator: VFCPageUrlComparator {readOnly}
+ VFCPage()
+ VFCPage(String, UUID)
+ getPageType() : VisitedLinkType
+ isCached() : boolean
+ setCached() : void
+ unsetCached() : void
+ getUrl() : String
+ getPageId() : UUID
+ changePageId(UUID) : void
+ getVector(String) : VFCVector
+ addVector(String, VFCVector) : void
+ removeVector(String) : void
+ containsVector(String) : boolean
+ containsAllVectors(Collection< String >) : boolean
+ getNumberVectors() : int
+ getVectorMap() : Map< String, VFCVector >
+ getChildPages() : Set< VFCPage >
- containsUrl(String) : boolean
+ addChildPage(VFCPage) : void
+ removeChildPage(VFCPage) : void
+ getNumberChildPages() : int
+ containsChildPage(VFCPage) : boolean
+ containsChildPage(String) : boolean
+ removeChildPage(String) : void
# finalize() : void
+ createUsingParent(VFCPage, VisitedLinkType, String, UUID) : VFCPage
+ resetRecency() : void
+ incrementRecency() : void
+ incrementFrequency() : void
Serializable
vfc::VFCVector
{leaf}
- serialVersionUID: long {readOnly}
- distance: int
- recency: int
+ CONS_RULES_COMP: VFCVectorConsComp {readOnly}
+ CONS_RULES_2D_COMP: VFCVector2DConsComp {readOnly}
- frequency: int
+ VFCVector(int, int, int)
+ getRecency() : int
+ setRecency(int) : void
+ incrementRecency() : void
+ resetRecency() : void
+ getFrequency() : int
+ setFrequency(int) : void
+ incrementFrequency() : void
+ getDistance() : int
+ equals(Object) : boolean
+ hashCode() : int
+ toString() : String
-pivotVector
Serializable
vfc::ConsistencyZoneDefinitions
- serialVersionUID: long {readOnly}
- pivotVector: VFCVector
- consistencyZoneVectors: List<VFCVector>
+ ConsistencyZoneDefinitions(VFCVector)
+ ConsistencyZoneDefinitions(JSONObject)
+ getPivotVector() : VFCVector
+ setPivotVector(VFCVector) : void
+ getConsistencyZoneVectors() : List< VFCVector >
+ addConsistencyZoneVector(VFCVector) : void
+ removeConsistencyZoneVector(VFCVector) : void

Figure 21: Domain Classes - Part 3

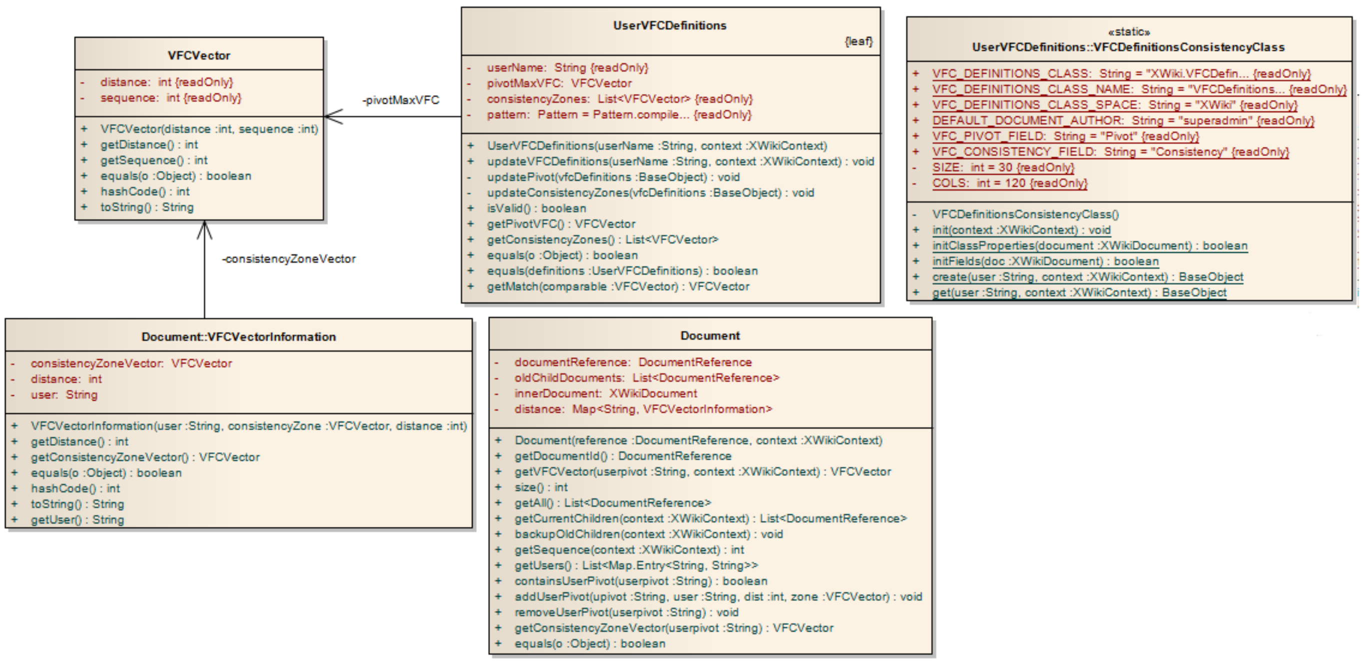
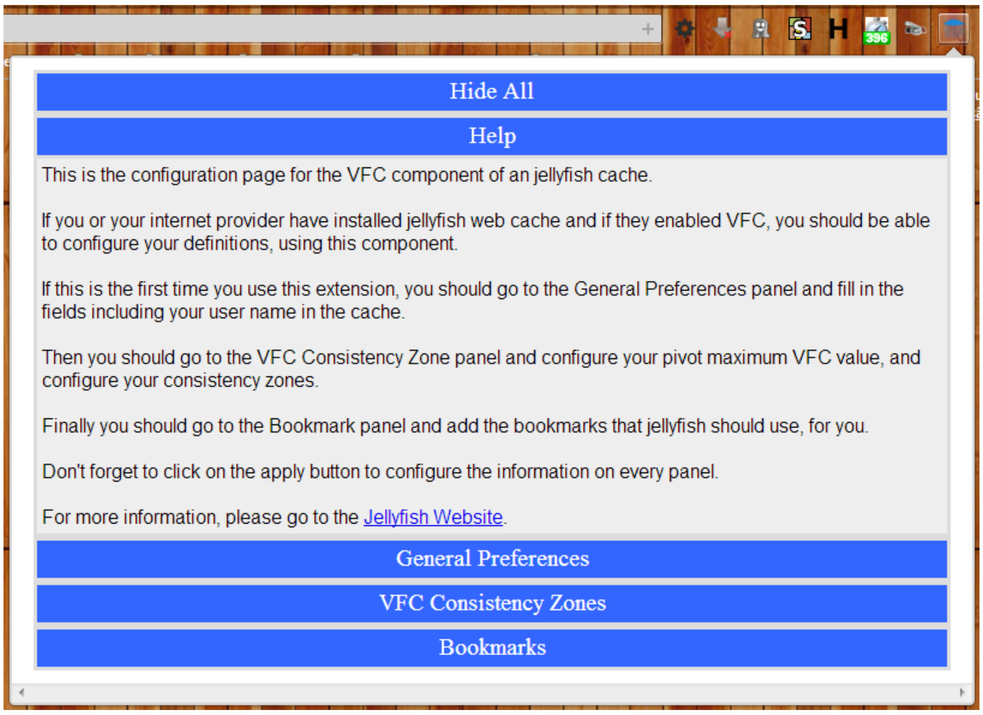

VFCVector
- distance: int {readOnly}
- sequence: int {readOnly}
+ VFCVector(distance :int, sequence :int)
+ getDistance() : int
+ getSequence() : int
+ equals(o :Object) : boolean
+ hashCode() : int
+ toString() : String
-pivotMaxVFC
UserVFCDefinitions
{leaf}
- userName: String {readOnly}
- pivotMaxVFC: VFCVector
- consistencyZones: List<VFCVector> {readOnly}
- pattern: Pattern = Pattern.compile... {readOnly}
+ UserVFCDefinitions(userName :String, context :XWikiContext)
+ updateVFCDefinitions(userName :String, context :XWikiContext) : void
- updatePivot(vfcDefinitions :BaseObject) : void
- updateConsistencyZones(vfcDefinitions :BaseObject) : void
+ isValid() : boolean
+ getPivotVFC() : VFCVector
+ getConsistencyZones() : List<VFCVector>
+ equals(o :Object) : boolean
+ equals(definitions :UserVFCDefinitions) : boolean
+ getMatch(comparable :VFCVector) : VFCVector
«static»
UserVFCDefinitions::VFCDefinitionsConsistencyClass
+ VFC_DEFINITIONS_CLASS: String = "XWiki.VFCDefin... {readOnly}
+ VFC_DEFINITIONS_CLASS_NAME: String = "VFCDefinitions... {readOnly}
+ VFC_DEFINITIONS_CLASS_SPACE: String = "XWiki" {readOnly}
+ DEFAULT_DOCUMENT_AUTHOR: String = "superadmin" {readOnly}
+ VFC_PIVOT_FIELD: String = "Pivot" {readOnly}
+ VFC_CONSISTENCY_FIELD: String = "Consistency" {readOnly}
- SIZE: int = 30 {readOnly}
- COLS: int = 120 {readOnly}
- VFCDefinitionsConsistencyClass()
+ init(context :XWikiContext) : void
+ initClassProperties(document :XWikiDocument) : boolean
+ initFields(doc :XWikiDocument) : boolean
+ create(user :String, context :XWikiContext) : BaseObject
+ get(user :String, context :XWikiContext) : BaseObject
-consistencyZoneVector
Document::VFCVectorInformation
- consistencyZoneVector: VFCVector
- distance: int
- user: String
+ VFCVectorInformation(user :String, consistencyZone :VFCVector, distance :int)
+ getDistance() : int
+ getConsistencyZoneVector() : VFCVector
+ equals(o :Object) : boolean
+ hashCode() : int
+ toString() : String
+ getUser() : String
Document
- documentReference: DocumentReference
- oldChildDocuments: List<DocumentReference>
- innerDocument: XWikiDocument
- distance: Map<String, VFCVectorInformation>
+ Document(reference :DocumentReference, context :XWikiContext)
+ getDocumentId() : DocumentReference
+ getVFCVector(userpivot :String, context :XWikiContext) : VFCVector
+ size() : int
+ getAll() : List<DocumentReference>
+ getCurrentChildren(context :XWikiContext) : List<DocumentReference>
+ backupOldChildren(context :XWikiContext) : void
+ getSequence(context :XWikiContext) : int
+ getUsers() : List<Map.Entry<String, String>>
+ containsUserPivot(userpivot :String) : boolean
+ addUserPivot(upivot :String, user :String, dist :int, zone :VFCVector) : void
+ removeUserPivot(userpivot :String) : void
+ getConsistencyZoneVector(userpivot :String) : VFCVector
+ equals(o :Object) : boolean

Figure 22: Classes implementing the VFC-Wiki model



This is the configuration page for the VFC component of an jellyfish cache.

If you or your internet provider have installed jellyfish web cache and if they enabled VFC, you should be able to configure your definitions, using this component.

If this is the first time you use this extension, you should go to the General Preferences panel and fill in the fields including your user name in the cache.

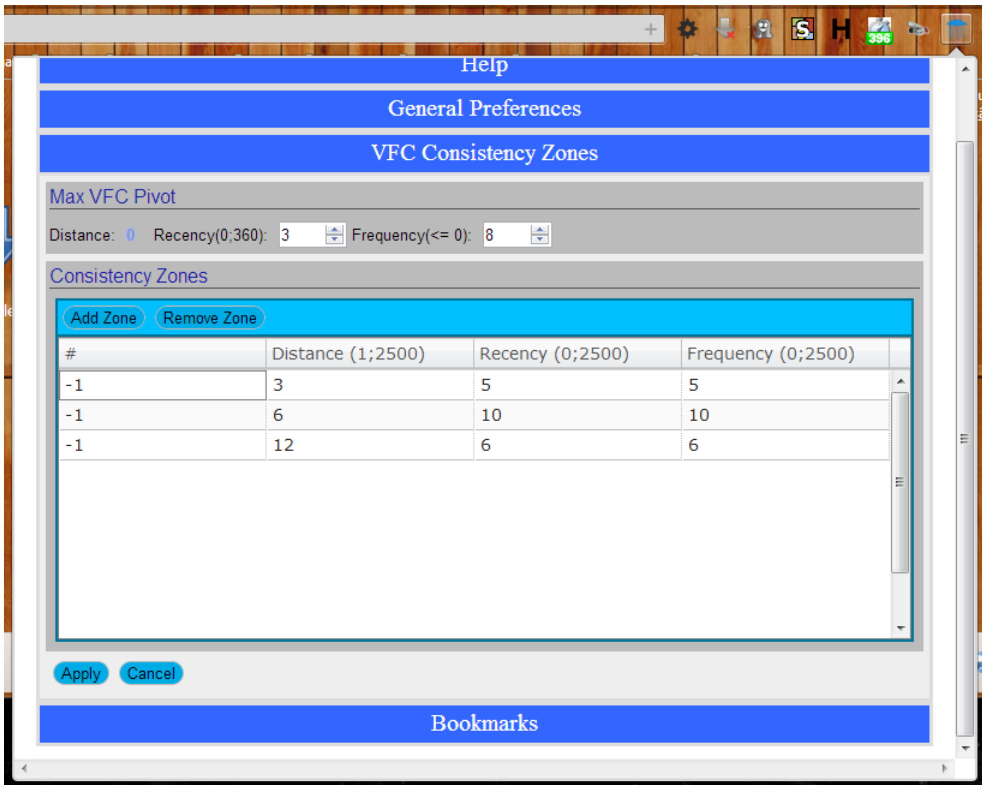
Then you should go to the VFC Consistency Zone panel and configure your pivot maximum VFC value, and configure your consistency zones.

Finally you should go to the Bookmark panel and add the bookmarks that jellyfish should use, for you.

Don't forget to click on the apply button to configure the information on every panel.

For more information, please go to the Jellyfish Website.

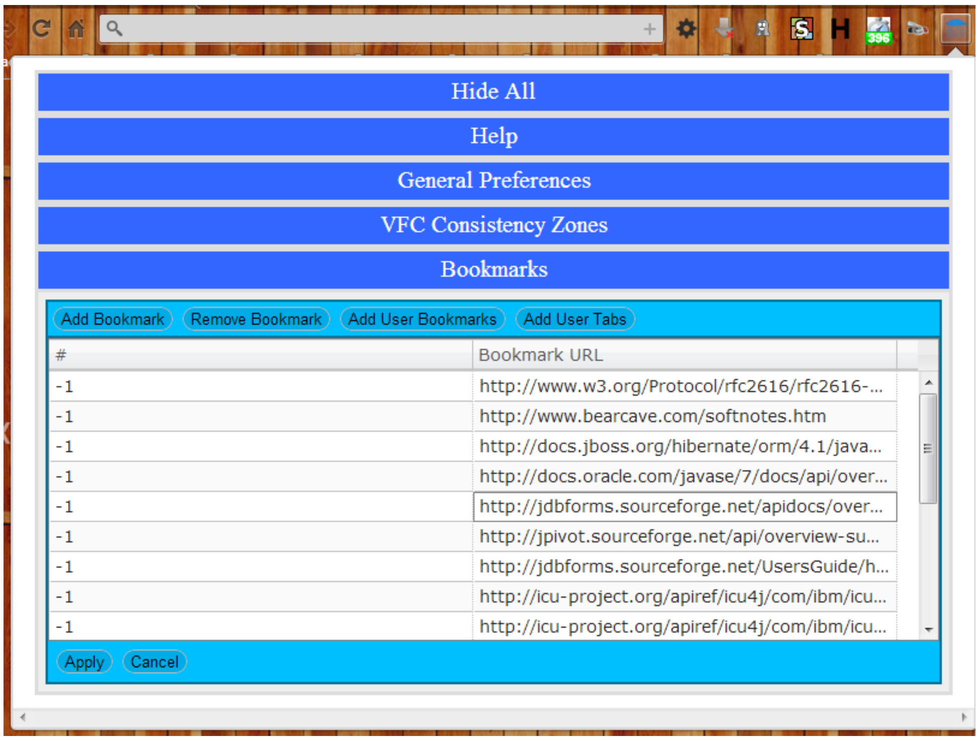
Figure 23: Help section of the VFC-Cache client plugin



Distance: 0 Recency(0;360): 3 Frequency(<= 0): 8

| # | Distance (1;2500) | Recency (0;2500) | Frequency (0;2500) |
|---|---|---|---|
| -1 | 3 | 5 | 5 |
| -1 | 6 | 10 | 10 |
| -1 | 12 | 6 | 6 |

Figure 24: VFC Consistency zone configuration section of the VFC-Cache client plugin



| # | Bookmark URL |
|---|---|
| -1 | http://www.w3.org/Protocol/rfc2616/rfc2616-... |
| -1 | http://www.bearcave.com/softnotes.htm |
| -1 | http://docs.jboss.org/hibernate/orm/4.1/java... |
| -1 | http://docs.oracle.com/javase/7/docs/api/over... |
| -1 | http://jdbforms.sourceforge.net/apidocs/over... |
| -1 | http://jpivot.sourceforge.net/api/overview-su... |
| -1 | http://jdbforms.sourceforge.net/UsersGuide/h... |
| -1 | http://icu-project.org/apiref/icu4j/com/ibm/icu... |
| -1 | http://icu-project.org/apiref/icu4j/com/ibm/icu... |

Figure 25: Bookmark configuration section of the VFC-Cache client plugin

## .2 List of Pages Used In the Test

- http://www.virtualbox.org/manual/UserManual.html
- http://www.gnu.org/
- http://www.gnu.org/gnu/linux-and-gnu.html
- http://www.gnu.org/licenses/licenses.html
- http://www.gnu.org/licenses/license-recommendations.html
- http://www.gnu.org/copyleft/copyleft.html
- http://www.gnu.org/help/help.html
- http://www.rfc-editor.org/rfcxx00.html
- http://www.rfc-editor.org/rfc/rfc959.txt
- http://www.rfc-editor.org/rfc/rfc868.txt
- http://tools.ietf.org/html/rfc867
- http://tools.ietf.org/html/rfc865
- http://tools.ietf.org/html/rfc2067
- http://tools.ietf.org/html/rfc1191
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.5.2
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html
- http://www.bearcave.com/software/java/comp_java.html
- http://www.cs.vu.nl/manta/
- http://www.bearcave.com/softnotes.htm
- http://www.bearcave.com/software/c++types.html
- http://www.bearcave.com/software/networking.html
- http://www.bearcave.com/software/java/comp_infra.html
- http://www.bearcave.com/software/java/misl/software_reuse.html
- http://www.bearcave.com/misl/misl_tech/c++_critique.html
- http://www.bearcave.com/unix_hacks/python.html
- http://tapestry.apache.org/index.html
- http://tapestry.apache.org/documentation.html
- http://tapestry.apache.org/user-guide.html
- http://tapestry.apache.org/annotations.html
- http://tapestry.apache.org/component-templates.html
- http://tapestry.apache.org/injection.html
- http://www.bearcave.com/
- http://www.bearcave.com/bear_contents.shtml
- http://www.bearcave.com/misl/misl_other/dogs/index.html
- http://www.bearcave.com/misl/misl_tech/wavelets/index.html

- http://www.bearcave.com/sourcepg.htm
- http://www.bearcave.com/italy/index.html
- http://www.bearcave.com/barcelona/index.html
- http://www.bearcave.com/bookrev/revindex.htm
- http://www.bearcave.com/finance/
- http://www.bearcave.com/finance/portfolio_equations/
- http://www.bearcave.com/finance/etf_portfolio/etf_portfolio.html
- http://www.bearcave.com/finance/factor_models/index.html
- http://www.bearcave.com/cae/index.html
- http://www.bearcave.com/dac_paper/dac_paper_final.html
- http://www.bearcave.com/cae/chdl/index.html
- http://www.bearcave.com/software/vhdl/index.html
- http://www.bearcave.com/cae/cascade_mult.html
- http://www.bearcave.com/misl/misc.htm
- http://www.bearcave.com/misl/misl_tech/index.html
- http://www.bearcave.com/bookrev/linked/index.html
- http://www.bearcave.com/misl/misl_tech/rdf_query_languages.html
- http://www.bearcave.com/misl/misl_tech/nlp.html
- http://www.bearcave.com/misl/misl_tech/detecting_authorship.html
- http://www.bearcave.com/misl/misl_tech/publishing.htm
- http://www.bearcave.com/misl/misl_tech/msdrm/index.html
- http://www.bearcave.com/misl/misl_tech/gnutella.html
- http://www.bearcave.com/misl/misl_tech/peer2peer.html
- http://www.bearcave.com/random_hacks/permute.html
- http://www.bearcave.com/misl/misl_tech/transmeta.html
- http://www.bearcave.com/misl/misl_tech/asynch.html
- http://www.bearcave.com/misl/misl_tech/demise_of_sun.html
- http://www.bearcave.com/software/dlang/Cplus_plus_and_d.html
- http://www.bearcave.com/software/btp/freesoft.html
- http://www.bearcave.com/misl/misl_tech/email_virus.html
- http://www.bearcave.com/misl/misl_other/tirza.html
- http://www.datanucleus.org/products/accessplatform.html
- http://www.datanucleus.org/products/accessplatform_3_2/jdo/guides/tutorial_mongodb.html
- http://db.apache.org/jdo/api30/apidocs/
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/annotations/package-summary.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/metadata/package-summary.html

- http://db.apache.org/jdo/api30/apidocs/javax/jdo/identity/package-summary.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/identity/CharIdentity.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/Constants.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/spi/Detachable.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/annotations/Element.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/metadata/ExtensionMetadata.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/annotations/FetchGroup.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/annotations/InheritanceStrategy.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/JDOCanRetryException.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/JDOHelper.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/JDOOptimisticVerificationException.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/JDOUnsupportedOptionException.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/metadata/Metadata.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/metadata/PackageMetadata.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/annotations/PrimaryKey.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/datastore/Sequence.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/annotations/Serialized.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/spi/StateManager.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/identity/StringIdentity.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/metadata/ValueMetadata.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/overview-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/overview-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/cache/infinispan/entity/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/cfg/beanvalidation/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/dialect/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/cache/infinispan/timestamp/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/cfg/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/dialect/resolver/package-summary.html

- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/ejb/criteria/expression/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/ejb/criteria/predicate/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/ejb/instrument/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/entities/mapper/relation/component/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/entities/mapper/relation/lazy/proxy/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/exception/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/query/impl/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/query/property/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/strategy/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/tools/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/envers/tools/reflection/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/id/insert/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/jmx/package-summary.html
- http://ant.apache.org/manual/Tasks/javadoc.html
- http://docs.oracle.com/javase/7/docs/api/overview-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/awt/im/spi/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/lang/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/lang/management/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/nio/charset/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/rmi/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/security/acl/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/util/logging/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/java/util/zip/package-summary.html

- http://docs.oracle.com/javase/7/docs/api/javax/annotation/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/javax/crypto/interfaces/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/javax/imageio/metadata/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/javax/imageio/stream/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/javax/lang/model/element/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/javax/management/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/javax/management/remote/rmi/package-summary.html
- http://docs.oracle.com/javase/7/docs/api/javax/naming/ldap/package-summary.html
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
- http://www.bearcave.com/software/c++types.html
- http://www.bearcave.com/unix_hacks/python.html
- http://docs.oracle.com/javase/7/docs/api/javax/management/remote/rmi/package-summary.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/cfg/package-summary.html
- http://www.datanucleus.org/products/accessplatform_3_2/jdo/guides/tutorial_mongodb.html
- http://db.apache.org/jdo/api30/apidocs/javax/jdo/metadata/ExtensionMetadata.html
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/org/hibernate/ejb/criteria/expression/package-summary.html
- http://www.bearcave.com/misl/misl_tech/transmeta.html
- http://wiki.squid-cache.org/
- http://wiki.squid-cache.org/SquidFaq
- http://wiki.squid-cache.org/SquidFaq/InterceptionProxy
- http://wiki.squid-cache.org/SquidFaq/SecurityPitfalls
- http://wiki.squid-cache.org/SquidFaq/ToomanyMisses
- http://wiki.squid-cache.org/SquidFaq/RAID
- http://www.squid-cache.org/Doc/config/cache_dir/
- http://en.wikipedia.org/wiki/Fiber_channel
- http://wiki.squid-cache.org/SquidFaq/SquidProfiling
- http://wiki.squid-cache.org/SquidFaq/ContentAdaptation
- http://wiki.squid-cache.org/SquidFaq/WindowsUpdate

- http://wiki.squid-cache.org/BestOsForSquid
- http://wiki.squid-cache.org/ZeroSizedReply
- http://wiki.squid-cache.org/ConfigExamples
- http://www.squid-cache.org/Versions/v2/2.7/cfgman/http_port.html
- http://www.squid-cache.org/Versions/v2/2.7/cfgman/maximum_object_size_in_memory.html
- http://www.squid-cache.org/Versions/v2/2.7/cfgman/memory_replacement_policy.html
- http://www.squid-cache.org/Versions/v2/2.7/cfgman/logformat.html
- http://www.squid-cache.org/Versions/v3/3.0/cfgman/
- http://old.squid-cache.org/Doc/Hierarchy-Tutorial/tutorial-9.html
- http://www.squid-cache.org/Versions/v2/2.7/cfgman/memory_replacement_policy.html
- http://www.squid-cache.org/Versions/v2/2.7/cfgman/maximum_object_size_in_memory.html
- http://www.squid-cache.org/Versions/v2/2.7/cfgman/http_port.html
- http://jpivot.sourceforge.net/api/overview-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/excel/package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/mondrian/
  script/package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/navigator/
  hierarchy/package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/olap/model/
  package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/param
  /package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/olap/
  navi/package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/table
  /package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/table
  /span/package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/test
  /olap/package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/util
  /package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/olap/model/
  impl/package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/mondrian/
  package-summary.html
- http://jpivot.sourceforge.net/api/com/tonbeller/jpivot/chart/
  package-summary.html
- http://jdbforms.sourceforge.net/UsersGuide/html/index.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch02.html

- http://jdbforms.sourceforge.net/UsersGuide/html/ch03s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch04s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch05s06.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch07s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch08s06.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch10s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch11s02.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch12s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch14s02.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch15s05.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch15s08.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch16s04.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch16s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch17s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch17s05.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch17s08.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch18s02.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch19s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch19s05.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch20s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch20s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch22s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch22s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch22s04.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch26s02.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch26s05.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch27s04.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch27s09.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch31s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch31s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch32s02.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch32s05.html
- http://jdbforms.sourceforge.net/UsersGuide/html/apb.html
- http://jdbforms.sourceforge.net/UsersGuide/html/apc.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch02s02.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch07s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch07s04.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch11s03.html

- http://jdbforms.sourceforge.net/UsersGuide/html/ch26s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch14s01.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch32s04.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch32s03.html
- http://jdbforms.sourceforge.net/UsersGuide/html/ch04s01.html
- http://jdbforms.sourceforge.net/apidocs/overview-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/config/package-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/dom/package-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/event/datalist/package-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/event/eventtype/package-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/servlets/reports/package-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/util/package-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/validation/package-summary.html
- http://jdbforms.sourceforge.net/apidocs/org/dbforms/xmldb/package-summary.html
- http://icu-project.org/apiref/icu4j/overview-summary.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/package-summary.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UCharacter.EastAsianWidth.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UCharacter.HangulSyllableType.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UCharacter.JoiningType.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UCharacter.NumericType.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UCharacter.SentenceBreak.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UCharacterEnums.ECharacterCategory.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UProperty.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UProperty.NameChoice.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/UCharacter.DecompositionType.html

- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/package-summary.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/AlphabeticIndex.Bucket.LabelType.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/AlphabeticIndex.Bucket.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/AlphabeticIndex.Record.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/AlphabeticIndex.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/ArabicShaping.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/ArabicShapingException.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/Bidi.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/BidiClassifier.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/BidiRun.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/BreakIterator.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/CanonicalIterator.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/CharsetDetector.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/AnnualTimeZoneRule.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BasicTimeZone.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BuddhistCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/ByteArrayWrapper.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BytesTrie.Entry.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BytesTrie.Iterator.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BytesTrie.Result.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BytesTrie.State.html

- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BytesTrie.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/BytesTrieBuilder.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Calendar.FormatConfiguration.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Calendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CaseInsensitiveString.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CharsTrie.Entry.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CharsTrie.Iterator.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CharsTrie.State.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CharsTrie.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CharsTrieBuilder.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/ChineseCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CompactByteArray.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CompactCharArray.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CopticCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Currency.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/CurrencyAmount.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/DangiCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/DateInterval.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/DateRule.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/DateTimeRule.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/EasterHoliday.html

- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/EthiopicCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Freezable.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/GenderInfo.Gender.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/GenderInfo.ListGenderStyle.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/GenderInfo.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/GlobalizationPreferences.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/GregorianCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/HebrewCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/HebrewHoliday.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Holiday.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/IllformedLocaleException.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/IndianCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/InitialTimeZoneRule.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/IslamicCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/JapaneseCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/LocaleData.MeasurementSystem.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/LocaleData.PaperSize.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/LocaleData.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/LocaleMatcher.LanguageMatcherData.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/LocaleMatcher.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/LocalePriorityList.Builder.html

- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/LocalePriorityList.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Measure.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/MeasureUnit.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Output.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/OverlayBundle.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/PersianCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/RangeDateRule.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/RangeValueIterator.Element.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/RangeValueIterator.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Region.RegionType.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/Region.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/RuleBasedTimeZone.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/SimpleDateRule.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/SimpleHoliday.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/SimpleTimeZone.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/StringTokenizer.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/StringTrieBuilder.Option.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/StringTrieBuilder.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/TaiwanCalendar.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/TimeArrayTimeZoneRule.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/TimeUnit.html

- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/TimeUnitAmount.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/TimeZone.SystemTimeZoneType.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/util/TimeZone.html

## .3 List of Bookmarks

- http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
- http://www.bearcave.com/softnotes.htm
- http://docs.jboss.org/hibernate/orm/4.1/javadocs/overview-summary.html
- http://docs.oracle.com/javase/7/docs/api/overview-summary.html
- http://jpivot.sourceforge.net/api/overview-summary.html
- http://jdbforms.sourceforge.net/UsersGuide/html/index.html
- http://jdbforms.sourceforge.net/apidocs/overview-summary.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/text/package-summary.html
- http://icu-project.org/apiref/icu4j/com/ibm/icu/lang/package-summary.html
- http://www.bearcave.com/finance/