VFC for Wikis and Web Caching

Carlos Roque

Instituto Superior Técnico

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 many 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 to the web and wiki environment.
Additionally since users are not interested in all of wiki page content, we also describe a way of filtering wiki pages, so that they only contain content that is useful to the user.

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 an 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[32], 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[2], 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.

Also, there is also an increase in the usage of mobile devices in order to connect to the internet, which additional more problems for these devices since, the bandwidth provided in mobile connections is usually inferior to the one provided in network connections, resulting in more stress to traditional caches[31] that now are of particular use due to the appearance of these devices, which increases the inefficiencies given above.

[1] http://www.websiteoptimization.com/speed/tweak/design-factors/

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 of a wiki page), while reducing the bandwidth required for transmitting those updates to the client.
This is compounded by 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 they 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.

1.0.1 Shortcomings of current solutions

In the case of the web caches, most of the current solutions try to propose different algorithms for page replacement, ie. different ways of specifying when a page should be removed from memory, but since most 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 into attention the user behavior, certain patterns like users which 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 inside that common working set.

1.0.2 Proposed Solution

To both of these problems we propose a strategy based on Vector-Field Consistency[35], 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 like the retrieval and extent of visualization of said document, in order to know when caches should update their documents and the maximum staleness that a wiki user tolerates in a given wiki page.
Using this algorithm, we also plan of 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.
Also as discussed above, since not all content is interesting to the user we also study and develop a technique that allows a wiki to determine what is interesting to the user in a wiki page and to filter uninterested stuff when serving those pages to him, which preserves most of the user satisfaction rate, while keeping the transmitted information low, since less data needs to be sent to the client, helping particularly with mobile users.

1.1 Objectives

In this paper we develop a system that adapts the Vector-Field Consistency model to semantically enhance caching of web pages and to show updates to pages interesting to a wiki user.
In order to determine what is interesting we also develop a browser plugin to track user behavior and determine what is interesting to him in a wiki page and send that information to the respective system.
In the subsections bellow, we provide more detail on the objectives and enumerate the collected functional and non-functional requirements for both the web cache and wiki system.

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

1.1.1 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 defined above) 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;

Robustness Our cache system should provide means to be used either alone or in a distributed fashion in order to improve balance load and reduce the probability of a single point of failure and by doing that being robust enough to be used in middle sized cache systems as intended;

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 VFC, our cache system will inherit the simplicity of VFC, while providing a powerful web cache scheme.

1.1.2 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 defined above) 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 and the extent of a web page change that a user is wiling to tolerate as a measure of wiki page importance;
- Within a page article, use information about the user behavior on that page in order to determine the article blocks that are important to a user and send update to only those blocks.

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

1.1.3 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:

- Can determine how much a user likes a certain page block, and transmit that to the wiki engine, given a set of page blocks;
- Provide a way for the user to control the consistency zones and specifying their consistency requirements, both for a cache and wiki;
- Control whether a filtered page is used or not, because a user may want to retrieve a faithful full rendering of the page without removal of uninterested blocks;
- View the classification of each block in terms of user preferences and to change or specify that the guess is wrong;
- Propagate changes of the bookmarked pages to the web cache server;

1.2 Document Road-map

We begin by enumerating our objectives in the section bellow (Section 2), then we study the related work (Section 3) in both web caching and wikis, to try to frame under taxonomies current existing systems and study they advantages and disadvantages, together with existing methods to determine user preferences in web pages.
Then we briefly describe VFC and its current uses, followed by a description of our architecture (Section 4) and finally the tests we plan to do to validate, evaluate and compare our solution to some of the other solutions studied in the related work (Section 5). The document closes with the conclusion.

2 Related Work

In this section 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.
We also discuss about existing strategies to break web pages into semantic blocks, features that are commonly used to determine user preferences and ways of gathering all the information in order to determine which blocks the user likes most.
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 matters;

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[40][33].

Centralized architecture

This architecture (Fig. 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 an small set of clients.

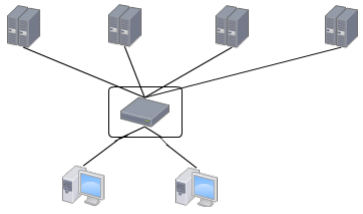

Fig. 1. Diagram of an centralized cache

Hierarchical architecture

The second approach (Fig. 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 bellow it, so that the document is available to the lower levels.

The Adaptive Web Caching or Top-10 prefetch[25] 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.

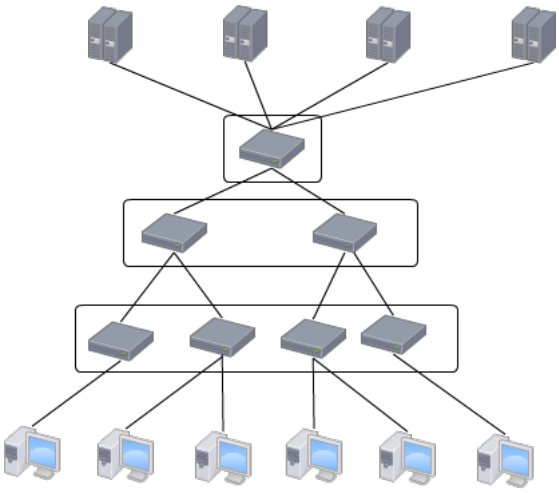

Fig. 2. Diagram of an hierarchical cache

Cooperative Distributed Architecture

In the third approach (Fig. 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 and that specifies the URL´s that the cache server is responsible for.

The Cache Array Routing Protocol[38] is a 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.

Hybrid Architecture

In the fourth approach (Fig.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, and upon receiving a request a cache server contacts first the servers at the same level as it, and only if

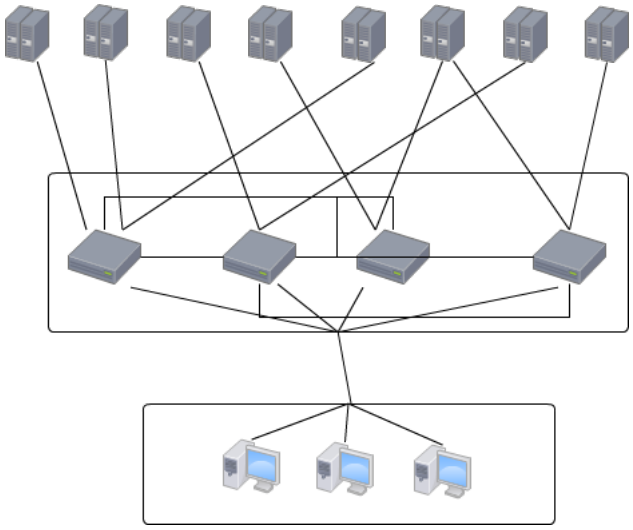

Fig. 3. Diagram of an cooperative distributed cache

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 an round trip time below a certain threshold, even if another neighbor has the document.
The Internet Cache Protocol[41] used by Squid [6] among others, is an example of an protocol based on an 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.

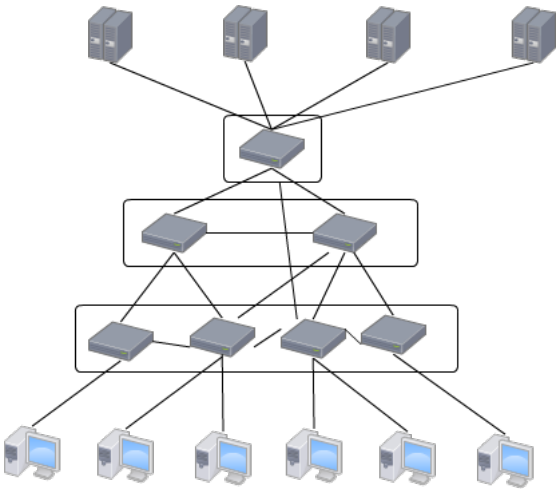

Fig. 4. Diagram of an hybrid cache

2.1.2 Models of web proxies

In the world of web proxies, there are two distinct models for web proxies[1][15], that influence the chosen consistency and algorithms.

[6] Squid is a open source proxy/cache server.

2.1.3 Forward proxy

The first model of proxy and the most used is the forward proxy model(Fig.5, where an proxy is sitting directly in front of a client and acts as an intermediary between an client and an a server, and hides the client from the server enabling the client to be directly served by the proxy if it has caching capabilities.

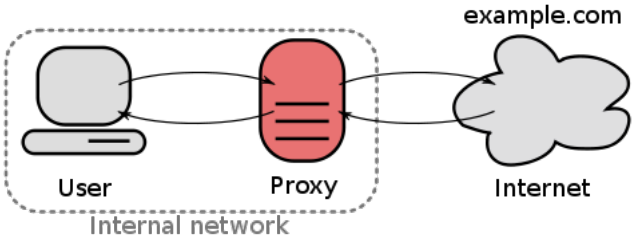

Fig. 5. Diagram of the forward proxy model

2.1.4 Reverse proxy

The second model of proxy is the reverse proxy model(Fig.6), and is associated with an 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 an 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 an 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 an mobile device or not; it also allows an provider to exploit the location of a client, in order to redirect the request to the proxy that is closer to the client.
This model also allows for blocking or filtering of requests in case of attacks to an determined web server, and depending of the proxy software to distribute the processing of an 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.

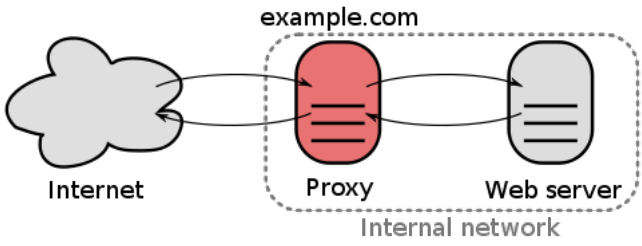

Fig. 6. Diagram of the reverse proxy model

2.1.5 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[32][8].

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[16] 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, 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 a 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, it is 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, it is 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, an 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[10].

While verifying if an 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, an 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 an conditional request which avoids the transmission of an full response, using only an response code of 304 (not modified), if the values match.
If the origin web server cannot be contacted the http protocol, specified that the web cache may serve the cached response, if it adds an 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 an 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[26], 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, 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, if the cache control headers allow it.

Consistency by Invalidation

The second strategy of consistency on the cached documents is consistency by invalidation[21][24]. 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.

2.1.5.0.1 Version 2.1

Another variation of the cache consistency based on invalidation is the use of leases[19] 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 if 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.

2.1.5.0.2 Version 2.2

On the other side it was later observed that there was a potential problem in the lease protocol as described above[44], 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, so that 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 cache leases, and only modifies a document when all responses are received, or when the granted leases expire, if some cache server cannot be contacted.

2.1.5.0.3 Version 3

In the paper by Yu[45], 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.

2.1.5.1 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[7] and cloud computing under an http server, the consistency by invalidation might find an 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 an 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.

2.1.6 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[28], that classify the strategies into:

[7] Content Distribution Networks

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 a 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 it 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 an 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 an 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 paragraphs, 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.

Recency based strategies

Pyramidal Selection Scheme

This is a strategy[5] 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 ΔT_{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.

LRU-Min[8]

This strategy[4] is based on the Least Recently Used algorithm, and modifies it in order to include an 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.

Frequency based strategies

LFU-DA[9]

This strategy[6] is based on frequency and tries to reduce some disadvantages associated with an pure frequency based strategy, by introducing an 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 an 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).

α-aging

This strategy[46] uses the notion of virtual ticks or periods that may have an 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 an tick, the strategy, uses the following aging function in all objects in cache:

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

Where α is an tunable value between zero and one, and that makes the algorithm ranging from an LRU to an LFU behavior.
Finally in the case of an tie, the strategy uses LRU in order to decide which object to remove.

2.1.6.1 Frequency-Recency based strategies

LRU*

LRU*[11] 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 removed. If not, the counter is decreased by one and placed at the start of the LRU list. This is

[8] Least Recently Used - Minimal
[9] Least Frequently Used - Dynamic Aging

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.

Hyper-G

This strategy[3], combines LRU, LFU and the size of objects in order to provide alternative mechanisms in the event of an 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 an tie then the largest object is removed. Until there is enough space to place the new object.

Function based strategies

GD-Size

The GD-Size[10][9] 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.

GDSF

The GDSF[11][6] 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 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.

Random strategies

Harmonic

This strategy[20] uses an weight random function, in which the probability of an object being selected for removal is equal to the inverse, of its cost.

[10] GreedyDual-Size
[11] GreedyDual-Size with Frequency

Machine Learning based strategies

NNPCR[12]

This is an strategy[13] based on machine learning specifically on the usage of an neural network as a way to give an 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 log with an hour long in order to train the neural network, and a second log with an hour long from a different day in order to verify the neural network results.

SVM-GDSF[13]

This strategy[31] uses an 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 an 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 way1, with W_i being the injected parameter:

$$K_i = F_i \times \frac{C_i}{S_i} + L + W_i \tag{1}$$

C4.5-GDS [14]

This strategy[31] uses an 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 an 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 an number between zero and one, that is the probability of an object being reused.
This number (P_i) is calculated in an accumulated way into an parameter W_i, at each request as follows:

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

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

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

2.1.6.2 Analysis

After studying some cache replacement algorithms, and following the results of numerous comparative tests between them[34], 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[31] may improve upon them.
Also given the rise of large objects being transmitted thought the web like movie or audio files[15], 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.

[12] Neural Network Proxy Cache Replacement
[13] Support Vector Machine Greedy Dual-Size with Frequency
[14] C4.5 - Greedy Dual-Size
[15] `http://www.websiteoptimization.com/speed/tweak/average-web-page/`

2.1.7 Commercial cache servers

In this subsection we describe the available cache server software in respect with the parameters studied above (table 1), 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.

Web Cache	Consistency Strategy	Cache Replacement Strategy	Architecture	Proxy Model	HTTP Version
Squid	Consistency by Validation (HTTP)	LRU or GDSF	Single or Hybrid (ICP)	Forward and Reverse Proxy	1.0 (1.1 on work)
Polipo	Consistency by Validation (HTTP)	LRU	Single	Forward Proxy	1.1
nginx	Consistency by Validation (HTTP)	LRU	Single	Forward and Reverse Proxy	1.1
Varnish	Any (Consistency by Validation (HTTP) by default)	LRU	Any	Reverse Proxy	1.1

Table 1. Comparison of existing web cache servers

squid

Squid[16] 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 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 or GDSF 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. an 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.

polipo

Polipo[17] 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

[16] `http://www.squid-cache.org`
[17] `http://www.pps.jussieu.fr/~jch/software/polipo`

like Tor[18] 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.

nginx

Nginx[19] 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 a 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.

Varnish

Varnish[20] 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 a 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 an 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 an consistency by invalidation protocol.

2.2 Wiki Systems

In this subsection, 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:

Enterprize wikis That are used within an enterprize[29] 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 enterprize or information that the owner wishes to share with others;
- **enterprize-wide wikis or pedias** That hold information about the enterprize as a whole and are meant as internal encyclopedias.

Educational wikis These wikis[17] 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;

[18] 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
[19] `http://nginx.org/en`
[20] `https://www.varnish-cache.org/`

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 a education institution;

Subject wikis These wikis are meant to contain books, materials or exercises about a given subject[21]

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; [22]

Specific-purpose public wikis These wikis are dedicated to one specific purpose or issue. [23]

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 only edits or creates something about what he or she knows, that only changes one section or small paragraphs at once, and that finds attractive the fact that a page is easily editable.
The other type of editor is the professional or wiki-savy 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, and when a page is changed they read what was changed and revert the change if they did not approve it, or correct the new edit, or approve the edition and do nothing [7].
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.

2.2.3 Classifying wikis by architecture

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

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.

[21] for example. `http://en.wikiversity.org/wiki/Wikiversity:Main_Page`
[22] for example `http://en.wikipedia.org/wiki/Main_Page`
[23] for example the wikis in `http://www.wikia.com/Wikia`

Peer to peer architecture

One example of a peer to peer architecture for wikis is presented the paper by Morris et al.[27] 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 a 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[24] in order to listen for document requests and answer them.

When a document is created, the peer generates a new JXTA advertisement,[25] 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).

Hybrid architecture

This architecture is described in Urdaneta et al.[37]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.

[24] A JXTA pipe is a way that JXTA has to connect specific peers together so that they share some data
[25] A JXTA advertisement is a way to tell peers about resources and network services available

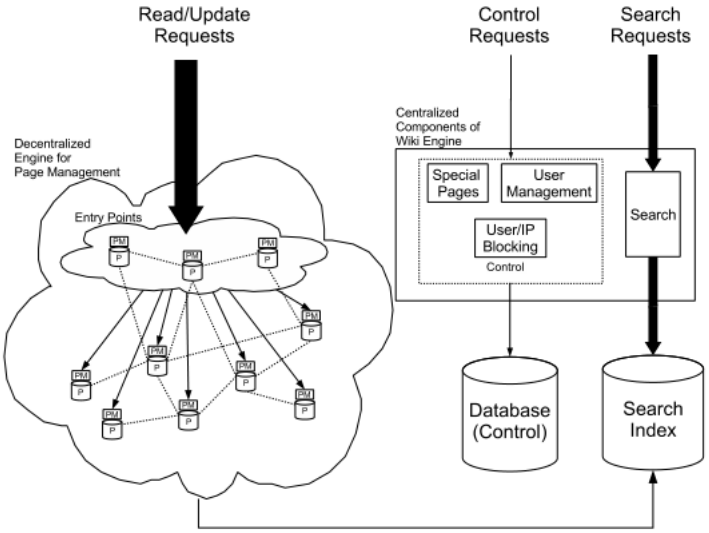

Fig. 7. Wiki Hibrida

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:

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

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. α, β 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.

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[26], 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 datacenters 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 User preference extraction

This section analysis the user preference extraction applied wiki pages, in order to extract information relative to the user preference of certain wiki page blocks. Therefore we which features can we use to classify a certain web block and methods of obtaining a page block score given a set of features.

2.3.1 User preference features

In this section we discuss some features based on implicit evidence collected from the user by indirectly observing his or her behavior when browsing a certain website, meaning that the user website contains functions that collect information about what the user does, when he is browsing and processes that information, in order to collect the features for a certain website block[12]. [27]
So in the paper by [23], these features are classified according to two different axis, the scope (or to what they apply) and the behavior category; the first axis (scope) is divided into Segment (some part of the webpage), Object (the webpage) and Class. The second axis is divided into:

Examine That is the observation of some element;
Retain That is the storage of some element, like printing some text or bookmarking a page[36];
Reference That is the quotation or reference of some element in other webpage;
Annotate That is the act of commenting or classify some element;
Create That is the creation of a bookmark or a wiki entry;

So in the following paragraphs we examine the most used features, and categorize them.

Examine

The paper by Kellar and Watters[22] points the following features:

- Usage of the bookmarks, back button of the browser;
- Usage of select, copy, cut and paste browser functions;

[26] see for example `http://www.wikia.com/Wikia` that allows users to create wikis about their favorite themes
[27] See for example the Curious Browser

– Usage pages in history and the usage of certain toolbars like the ones from google or yahoo, as a way of collecting information about whether the user liked of one page as a whole or not and if that page was important to the task he was doing.

In the paper by Shapira [36] additional features are described, specifically:

– The relation of mouse movement and the reading time;
– The relation of scrolling and reading time;
– The reading time normalized by the page size;
– The number of links visited on a page (also refereed in the paper above);
– The number of links visited relative to the number of existing links on the page;
– Whether or not the user performed some actions, like selecting or copy text or images on the page.

Annotate

As an example of feature belonging to the annotate there is the paper by Golovchinsky et al.[18], where it is used the annotation of documents using a pen and/or a tablet in order to extract certain features, specifically:

– The circle of a word or paragraph;
– The underline of certain phases;
– The creation of sidenote marks along the text;
– Comments made to the text;

The authors brake the text into words, and consider that the more two different words were underlined the more weight, that word should have. The authors conclude that these annotations were more conclusive proof to the usefulness of a certain block, than the other traditional features.

2.3.2 Machine learning strategies

In order to given a set of significant feature find if a certain page block is interesting to a user or not, we have identified three machine learning strategies, commonly used in this problem by the work in the existing literature[43] that are Support Vector Machine[14], C4.4[30] and Naive Bayes[42]. It is shown in that paper that C4.4 has the highest precision but it has the tendency for over-fitting, while Naive Bayes has the worst performance due to the fact that most of the features used in the user preference extraction are somehow dependent from each other.

2.3.3 Overview

In this section we have described some commonly collected user navigation features and tried to categorize them, finally, we described some of the used strategies in order to merge the features and produce a percentage of user interest in a page block, so that the browser plugin can extract the preferred blocks from a wiki page.

2.4 Vector-Field Consistency

The Vector-Field consistency (or VFC) algorithm[35][39] is an algorithm that unifies consistency enforcing with locality awareness, using the concept of vector-fields.
Where there is an pivot or center point that represents an important object for the system or observation point and generates consistency zones around that object, where we place the objects according to an set of dimensions defined in a vector called the VFC vector.
Each consistency zone has a given consistency degree (Fig. 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 an range that goes from the previous consistency zone (or pivot, in the case of the first zone), to an certain maximum VFC vector value.

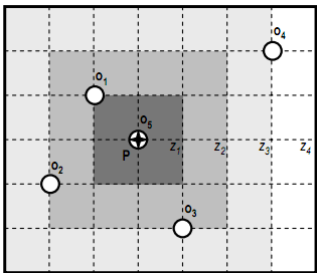

Fig. 8. Example of a consistency zone

In order to position a 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, a object or pivot may also belong to more than one view (Fig. 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 an view can be associated with an user, an game field layer, etc.

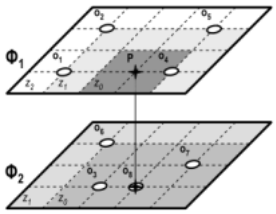

Fig. 9. Example of multiple views

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 an game object could have had, sequence the maximum number of lost updates of that object and value the difference between replica contents.

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 accessability.
In the wiki we are going to use an centralized architecture as the wiki architecture, and in the user interests component we are going to use an wiki specific page breaking algorithm and C4.5 to join the user collected features.

3.1 Web Cache

Since our work is related to two main parts, the web cache and the wiki, in this subsection, 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 an use case of an request using the normal HTTP consistency, the registering of a user and it´s consistency definitions and bookmarks including the exchange network messages, and then an request using VFC.
Finally we describe some relevant algorithms and data structures.

3.1.1 VFC Model adaptation

In the VFC model, we have used an 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 an 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 a 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 an 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 an 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 an javascript file that is used by the current HTML document, and it´s distance is going to be equal to the distance of the current document.

Also, since an pivot should typically be the only element with an distance of one, when we are parsing an 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 an distance of zero).

Also only an bookmark may become an pivot, so if an user makes an request to an document that is not linked to an pivot, then the document isn't cached, even if later (by parsing more pages) it is found that the document did have an connection to an pivot; so that the algorithm can provide an fast way of deciding what to cache.

If an document needs to be revalidated 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 bellow.

If an 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 an 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 bellow.

If an document is linked to more than one pivot (they may be from multiple users), then the document as an VFC vector for each connection that is associated with an 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[28], 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 then 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;

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 it´s fullest, while keeping the wanted properties and qualities of user preference awareness.

3.1.3 Base Component Model

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

Server side

As an architecture for the proxy layer (Fig. 10), 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 bellow.

Port Management

This blocks contains all the ports, that are components that can handle client and/or server connections, in an given protocol (for now there is only ports for HTTP 1.1).
Also, it can be any number of ports, that have an unique identifier and may have only inputs or outputs or both, so that an user may configure an port to receive special requests from an web application or an port that is specific for an certain type of server content like objects with an large size.
We use the single-threaded pro-actor pattern to manage the connections because it can support more clients than a thread based solution (since it does not reach any CPU limits), and therefore reduces the server overhead and does not damage the experience of the cache users, which increases the user satisfaction. The components of this block are described below:

Port or Connection Port This component manages the connections 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 subcomponents:
- **Client layer** This component is the client side of the port and processes, client requests and sends responses, and has subcomponents to parse the HTTP request into an generic proxy request and an proxy response to an HTTP response.
- **Server layer** This component is the server side of the port and processes, server responses and sends requests, and has subcomponents to parse the an proxy request to an server request and an server response to an proxy response.

[28] see the description of squid above

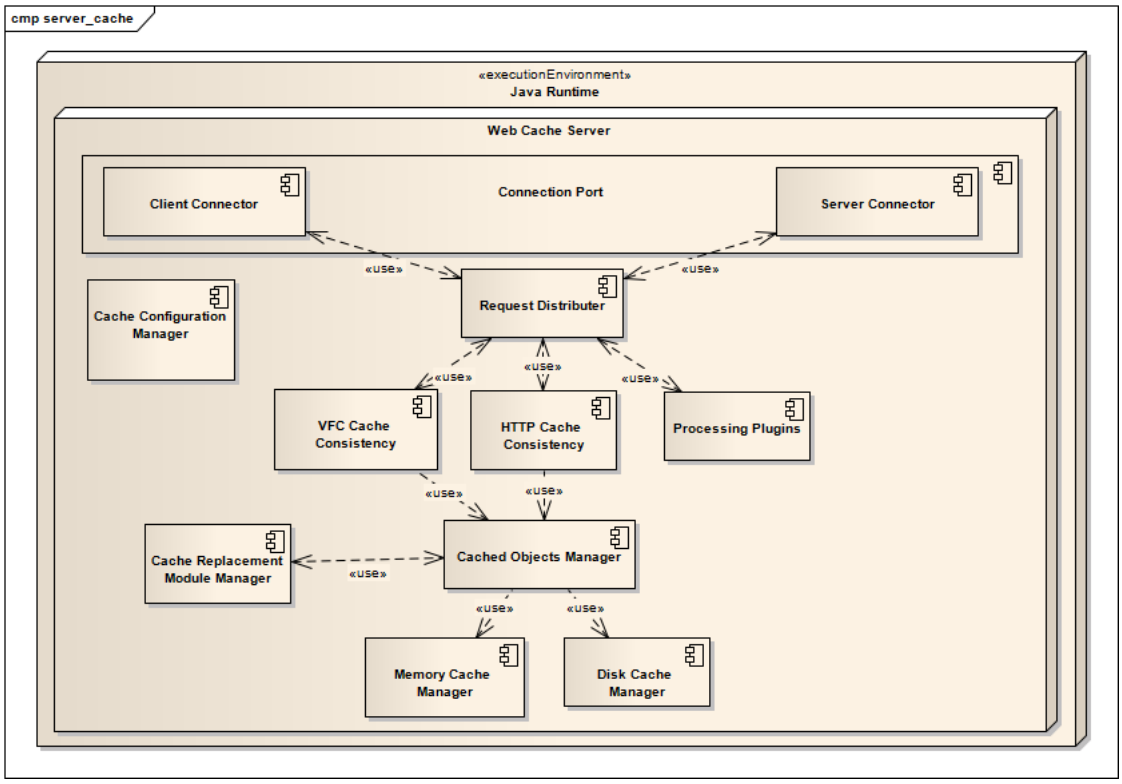

Fig. 10. Architecture of the web proxy/cache server

Request Processor

This block receives proxy requests from an client connection, and using an chain of responsibility distributes the requests for processing by the VFC consistency module or the HTTP consistency module, it may also distribute them through aditional modules that may store the message, produce an log of the requests or transform either the request or response.
If an processing model 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.
With this approach we are able to deal with all kinds of environments and to adapt to more environments which is essential in a ISP or large enterprize network. Finally, the request processor subcomponents 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 an web server;

VFC Cache Consistency This component manages the VFC protocol and return 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 an request with or without an 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 an configurable period of inactivity.

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;

Memory Cache Manager This component manages a memory cache, that stores documents in memory, using a content-addressed system[29], 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.

Client side

As an architecture for the proxy layer (Fig. 11), we have a model that combines an traditional MVP[30], with an component to change the requests made by the browser, and an main component that is represented by an button in the browser and allows the browser user to configure it´s user name, the consistency zone definitions and the bookmarks he wants to consider as pivots. The rest of the blocks are described bellow.

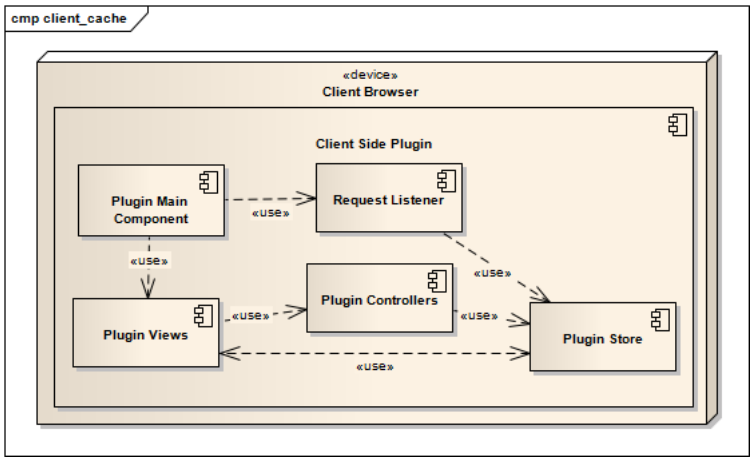

Fig. 11. Architecture of the web proxy/cache client plugin

Plugin Views

This component is responsible to present an user interface in order for the user to configure it´s user name, to view the cache generated user name (using an hash of the given user name, browser version and an 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 an pivot is updated and to configure it´s bookmarks/pivots to send and optionally adds all of it´ current bookmarks and/or open tabs.

[29] Also known as CAS
[30] Model View Presenter

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 it´s name, or the consistency zone definitions, or any of the other configurable parameters.

Plugin Store

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

3.1.4 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 it´s related operations), and when the cache receives an VFC request.

HTTP request use case

We an client browser makes an request to the cache (Fig. 12), the following steps are done, considering that the page is not in the cache, and an server request must be made:

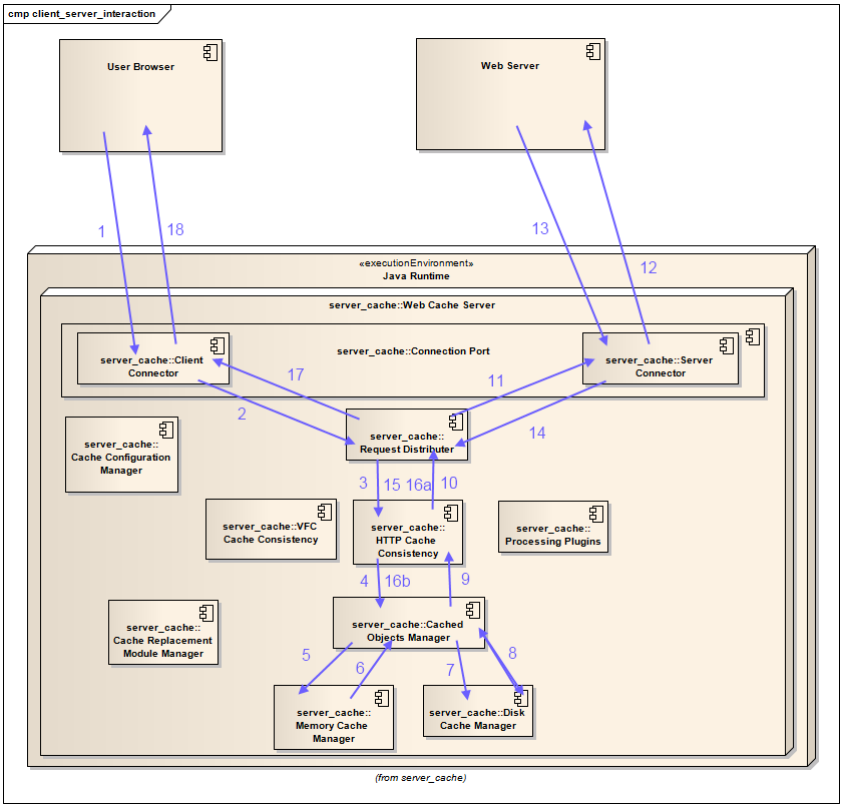

Fig. 12. Use case involving an HTTP request.

1. The client makes the connection;
2. The client part of the port that receives the connection parses the request, and extracts the headers, the cache control options, the pragma options, the requested ranges and if the connection is persistent or not, and builds an 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, uses distributes the request along the chain of responsibility containing the HTTP Cache Consistency, that processes the request;
5. The HTTP Cache Consistency component, sends an 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 an failure response to the request;
9. The HTTP Cache Consistency, transforms the client request into an server request and places it in an queue for sending it to the original web server;
10. The Request Distributor, retrieves the server request and passes it to an compatible Server Connector, using an queue;
11. The Server Connector, retrieves the request from the queue and sends it to the server;
12. Upon receiving an response, the Server Connector, associates the response to the proxy request and handles it to the Request Distributor, by placing it in a queue;
13. The Request Distributor, retrieves the request from the queue, and distributes 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 and handles it to the Client Connector;
16. The Client Connector answers the client and sends it the response.

VFC user registration use case

When an client browser makes an request to the cache (Fig. 13), in order to register an user and it´s information the following steps are done:

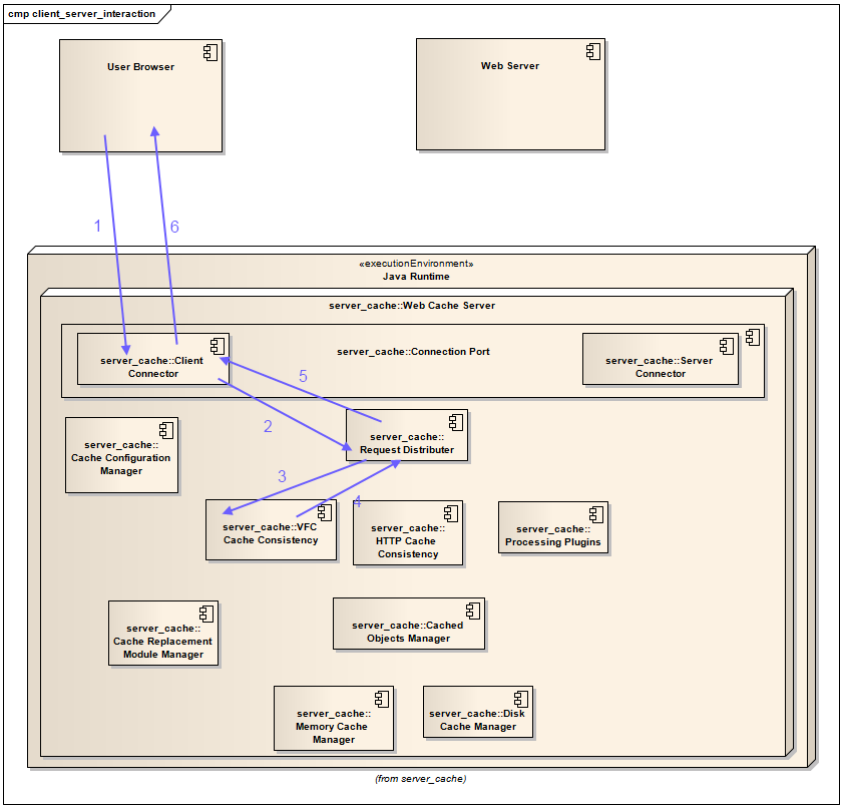

Fig. 13. Use case involving an user registration.

1. The user browser plugin makes an request, to register the user and it´s data, that is an regular HTTP request;
2. The Client Connector, sends the data to the Request Distributor, that places the request in the chain of responsibility;
3. The VFC Cache Consistency, handles the request and processes the received data, adapting the user pivots, or the user consistency zones if they change;
4. The VFC Cache Consistency, sends an 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 an set of bookmarks or consistency zone definitions that may create new pivots and respective consistency zones or change the existing ones.

VFC request use case

We an client browser makes an request to the cache using VFC (Fig. 14), the following steps are done, considering that the page is not in the cache, and an server request must be made:

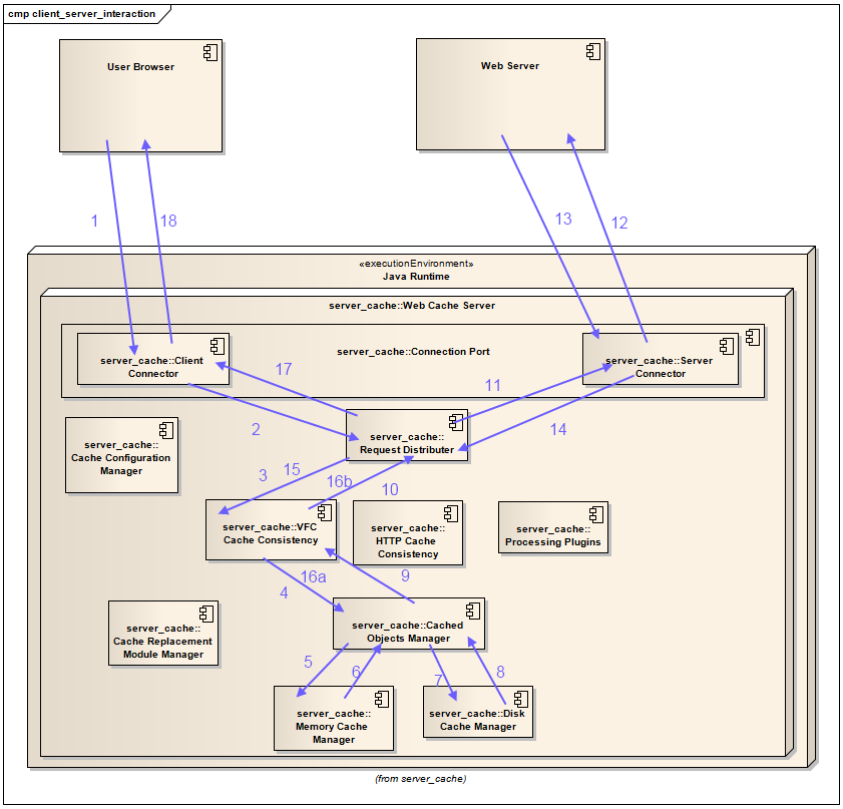

Fig. 14. Use case involving an VFC request.

1. The client makes the connection;
2. The client part of the port that receives the connection parses the request, and extracts the headers the requested ranges and if the connection is persistent or not, and builds an 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, uses distributes the request along the chain of responsibility containing the VFC Cache Consistency, that processes the request;

5. The VFC Cache Consistency component, determines the VFC Page corresponding to the request and sends an 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 an failure response to the request;
9. The VFC Cache Consistency, transforms the client request into an server request and places it in an queue for sending it to the original web server;
10. The Request Distributor, retrieves the server request and passes it to an compatible Server Connector, using an queue;
11. The Server Connector, retrieves the request from the queue and sends it to the server;
12. Upon receiving an response, the Server Connector, associates the response to the proxy request and handles it to the Request Distributor, by placing it in a queue;
13. The Request Distributor, retrieves the request from the queue, and distributes it to the chain of responsibility, allowing the HTTP Cache Consistency to process it;
14. The VFC Cache Consistency parses the received page if the page is an html request, adds the parsed links as shadow VFC Pages to the consistency zones where the parent is inserted and 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. It also adds one to the recency value of all pages;
15. The Request Distributor, retrieves the request from the queue and handles it to the Client Connector;
16. The Client Connector answers the client and sends it the response.

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 it´s current pivot is within the consistency zone boundaries and if it is adds one to it´s frequency and one to the recency of all VFC Pages. If not sends an 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

TODO

4 Solution Assessment

As for a comparative evaluation we will use a qualitative, a quantitative, and a comparative evaluation of our cache and wiki architecture. In particular we will evaluate our cache replacement algorithm, and try to perform an evaluation of user satisfaction related to the identified important blocks.
In the qualitative evaluation we will evaluate:

- The correctness of the cache behavior, in particular if our cache presents the correct page without unexpected errors;
- Whether or not our cache is able to fail gracefully if there are expected errors, in particular if the connection to a web server is broken during the transmission of data;
- The correctness of the VFC protocol application, in particular the correct generation of consistency zones around all pivots and under all identified cases;
- The correctness of the VFC consistency zone enforcement, in particular if pages are assigned to the correct consistency zone and updated accordingly;
- The correctness in absence of the browser plugin, in particular, if the cache is able to fallback to standard HTTP protocol consistency;

We will also try to evaluate the behavior of our cache in a distributed environment.
In the Quantitative evaluation we will try to evaluate:

- Compared to a solution without caching:
  - The network usage, to determine if our cache is able to reduce the load between the clients and web servers, specially if there are lots of clients (ISP scenario);

- The time it takes for a request to be satisfied, and if the VFC usage effectively gives similar results to a solution without caching;
- The correctness of the response, if our cache server ir always able to produce the same (or acceptable) response as a browser would do alone.

– Compared to other caches:
  - The number of hit-rates, to see if our solution has a better hit-rate ratio than other cache server, also taking into account user´s interests and respective weighting;
  - The number of miss-rates, particularly if not only the number of miss-rates is reduced but also if the hit-rate is inversely proportional to the miss-rate (which gives that most requests are *correctly* served from cache), also taking into account user´s interests and respective weighting;
  - The network usage, in particular if our solution wastes less bandwidth than other cache servers;
  - The time it takes for a request to be satisfied, and if the time is lower than the average of the other cache servers;
  - The used disk space, to check if our solution requires less disk space than others;
  - The memory usage, the same with the memory.

– Individually using various workloads:
  - The used disk space, to check if the disk space usage is always low, without abnormal workloads;
  - The number of miss-rates;
  - The number of hit-rates;
  - The memory usage;

In the cache replacement strategy evaluation we will measure against other cache replacement algorithms (eg. LRU, PSS and LRU*):

– The hit-rate, to check if our solution provides a better hit-rate;
– The miss-rate;
– The used cache space, to check if our solution wastes less disk space;
– The time used in cache replacement, to see if our solution requires less overhead and therefore is able to provide more speed to transfers;

We will also try to measure savings in space, bandwidth and time from obeying to VFC criteria defined by the clients.
In order to identify the user satisfaction and block division algorithms related to the identified important blocks, we will try to do tests with users, and record the feature vector for each site they visit and their classification and comments about the identified important blocks and the identified blocks in order to check if the penalty values are always low or nonexisting.

5 Conclusion

We now conclude this report by summarizing what was done and presenting the final conclusions. So, in this report we discussed some web cache and wiki engine architectures and explored some advantages and disadvantages of each.
After that we analyzed in detail the consistency protocols used in web caches; the various types of page replacement strategies and their advantages and disadvantages, including the analysis of some of the most used page replacement algorithms and ended with the analysis of some existing cache servers.
For the wiki systems we analyzed the types of existing wikis and the user types more common in public wikis.
After that we analyzed the existing algorithms to divide a web page into blocks, which features are most commonly used to extract implicit user information and techniques used to gather all the information collected from the user in order to determine which blocks of a page he enjoys most.

References

1. *Web protocols and practice: HTTP/1.1, Networking protocols, caching, and traffic measurement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
2. Ghaleb Abdulla. *Analysis and modeling of world wide web traffic*. PhD thesis, 1998. AAI9953804.
3. Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Edward A. Fox, and Stephen Williams. Removal policies in network caches for world-wide web documents. *SIGCOMM Comput. Commun. Rev.*, 26(4):293–305, August 1996.
4. Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. Technical report, Blacksburg, VA, USA, 1995.
5. Charu Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the world wide web. *IEEE Trans. on Knowl. and Data Eng.*, 11:94–107, January 1999.
6. Martin Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich, and Tai Jin. Evaluating content management techniques for web proxy caches. *ACM SIGMETRICS Performance Evaluation Review*, 27(4):3–11, March 2000.
7. Susan L. Bryant, Andrea Forte, and Amy Bruckman. 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, pages 1–10, New York, NY, USA, 2005. ACM.
8. LY Cao and M.T. Oezsu. Evaluation of strong consistency web caching techniques. *World Wide Web*, 5(2):95–123, 2002.
9. Pei Cao and Sandy Irani. 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, pages 18–18, Berkeley, CA, USA, 1997. USENIX Association.
10. Vincent Cate. Alex-a global filesystem. In *Proceedings of the 1992 USENIX File System Workshop*, number 7330, pages 1–12. Citeseer, 1992.
11. C Chang and AJ McGregor. The LRU* WWW proxy cache document replacement algorithm. 1999.
12. Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *Proceedings of the 6th international conference on Intelligent user interfaces*, IUI '01, pages 33–40, New York, NY, USA, 2001. ACM.
13. Jake Cobb and Hala ElAarag. Web proxy cache replacement scheme based on back-propagation neural network. *J. Syst. Softw.*, 81(9):1539–1558, September 2008.
14. Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.
15. Brian D. Davison. A web caching primer. *IEEE Internet Computing*, 5(4):38–45, July 2001.
16. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext transfer protocolHTTP/1.1, June 1999. *Status: Standards Track*, 1(11):1829–1841, 1999.
17. Andrea Forte and Amy Bruckman. Constructing text:: Wiki as a toolkit for (collaborative?) learning. In *Proceedings of the 2007 international symposium on Wikis*, WikiSym '07, pages 31–42, New York, NY, USA, 2007. ACM.
18. Gene Golovchinsky, Morgan N. Price, and Bill N. Schilit. From reading to retrieval: freeform ink annotations as queries. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 19–25, New York, NY, USA, 1999. ACM.
19. C. Gray and D. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. *SIGOPS Oper. Syst. Rev.*, 23:202–210, November 1989.
20. Saied Hosseini-Khayat. *Investigation of generalized caching*. PhD thesis, St. Louis, MO, USA, 1998. UMI Order No. GAX98-07761.
21. John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6:51–81, February 1988.
22. Melanie Kellar and Carolyn Watters. Using web browser interactions to predict task. In *Proceedings of the 15th international conference on World Wide Web*, pages 843–844, Edinburgh, Scotland, 2006. ACM.
23. Diane Kelly and Jaime Teevan. Implicit feedback for inferring user preference: a bibliography. In *ACM SIGIR Forum*, volume 37, pages 18–28. ACM, 2003.
24. Byoung-Hoon Lee, Sung-Hwa Lim, Jai-Hoon Kim, and Geoffrey C. Fox. Lease-based consistency schemes in the web environment. *Future Generation Computer Systems*, 25(1):8 – 19, 2009.
25. E.P. Markatos and C.E. Chronaki. A top-10 approach to prefetching on the web. In *Proceedings of INET*, volume 98, pages 276–290, 1998.
26. J.C. Mogul. Clarifying the fundamentals of HTTP. *Software: Practice and Experience*, 34(2):103–134, 2004.
27. J.C. Morris. DistriWiki:: a distributed peer-to-peer wiki network. In *Proceedings of the 2007 international symposium on Wikis*, pages 69–74. ACM, 2007.
28. Stefan Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4):374–398, 2003.
29. Erika Shehan Poole and Jonathan Grudin. A taxonomy of wiki genres in enterprise settings. In *Proceedings of the 6th International Symposium on Wikis and Open Collaboration*, WikiSym '10, pages 14:1–14:4, New York, NY, USA, 2010. ACM.
30. Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52:199–215, 2003. 10.1023/A:1024099825458.
31. Feng Qian, Kee Shen Quah, Junxian Huang, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. Web caching on smartphones: ideal vs. reality. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 127–140, New York, NY, USA, 2012. ACM.
32. Michael Rabinovich and Oliver Spatschek. *Web caching and replication*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.
33. P. Rodriguez, C. Spanner, and E.W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *Networking, IEEE/ACM Transactions on*, 9(4):404–418, 2001.
34. Sam Romano and Hala ElAarag. A quantitative study of recency and frequency based web cache replacement strategies. In *Proceedings of the 11th communications and networking simulation symposium*, CNS '08, pages 70–78, New York, NY, USA, 2008. ACM.
35. Nuno Santos, L. Veiga, and Paulo Ferreira. Vector-field consistency for ad-hoc gaming. *Middleware 2007*, pages 80–100, 2007.
36. Bracha Shapira, Meirav Taieb-Maimon, and Anny Moskowitz. Study of the usefulness of known and new implicit indicators and their optimal combination for accurate inference of users interests. *Proceedings of the 2006 ACM symposium on Applied computing - SAC '06*, page 1118, 2006.
37. Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. *A Decentralized Wiki Engine for Collaborative Wikipedia Hosting*, page 156163. Citeseer, 2007.
38. V Valloppillil and K W Ross. Cache array routing protocol v1.1. *1*, 1998.
39. Luís Veiga, André Negrão, Nuno Santos, and Paulo Ferreira. Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency. *Journal of Internet Services and Applications*, 1(2):1–21, August 2010.
40. Jia Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review*, 29:36–46, October 1999.

41. D Wessels and K Claffy. Application of internet cache protocol (icp), version 2. *RFC Editor United States*, (2187), 1997.
42. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
43. Xin Yang, Peifeng Xiang, and Yuanchun Shi. Finding user's interest blocks using significant implicit evidence for web browsing on small screen devices. *World Wide Web*, 12:213–234, June 2009.
44. J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Using leases to support server-driven consistency in large-scale systems. In *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on*, pages 285–294. IEEE, may 1998.
45. Haobo Yu, Lee Breslau, and Scott Shenker. A scalable web cache consistency architecture. *SIGCOMM Comput. Commun. Rev.*, 29:163–174, August 1999.
46. Junbiao Zhang, Rauf Izmailov, Daniel Reininger, Maximilian Ott, and Nec U. S. A. Web caching framework: Analytical models and beyond. In *Proceedings of the 1999 IEEE Workshop on Internet Applications*, WIAPP '99, pages 132–, Washington, DC, USA, 1999. IEEE Computer Society.