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 a lot of people, and therefore increase the use of the network, regardless of the level of interest each one has on those modifications.
In this paper, we describe how the current web and wiki systems perform caching and manage replication, and offer an alternative approach by adopting Vector-Field Consistency and the use of some preferences of the user, to the web and wiki environment.

1 Introduction

In today's Internet environment, there is an increasing number of users geographically dispersed and a large amount of those users uses the web, some of them exclusively, to do their daily work and to look for information or for recreational activities like social networking, online gaming, etc. This poses many problems to the internet service providers that have almost unlimited requests for providing more bandwidth to their clients, placing the providers in a difficult situation since their network may not be ready for the increase of traffic and the redesign of that network may have prohibitive costs.

So in order to continue satisfying the demands of the users, many ISPs use one or more layers of cache servers in order to reduce the bandwidth required for some of the most common protocols (like HTTP), but due to the dynamic nature of the requirements needed for a cache, those caches have more and more problems to satisfy the client´s needs (that prefer a fast response above all),[1] mainly related to the number of unnecessary pages (frequently dynamic), cached and updated/invalidated and to the fact that many web pages have a lot of uninteresting things to the user, that are highly dependent on the user itself, also 40% of all requested objects from a client are not cacheable without resorting to special strategies(Rabinovich & Spatschek 2002) and many objects cached are useless since even if 90% of all web accesses are made to the same set of servers, there is a significant percentage of those pages that are accessed only once(Abdulla 1998).

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

1.1 Shortcomings of current solutions

In the case of the web caches, most of the current solutions try to propose different algorithms for page replacement, i.e. different ways of specifying when a page should be removed from memory, but many pages are non-cacheable according to the most commonly used consistency method,

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

because of many 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.

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

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

1.2 Proposed Solution

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

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

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

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

2 Architecture

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

2.1 Web Cache

Since our work is related to two main parts, the web cache and the wiki, in this chapter, we describe the cache architecture, starting by the VFC adaptation and the description of the cache replacement strategy.

2.1.1 VFC Model Adaptation

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.1.2 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[6], there are two values important for the document removal/replacement algorithm:

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

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

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

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

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

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

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

2.2 VFC-Wiki

For the VFC-Wiki we are going to discuss the adaptations for VFC, relative to the VFC used for the web cache.

[6] see the description of squid above

2.2.1 VFC Adaptations

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

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

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

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

3 Implementation

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

3.1 VFC-Cache

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

So in this subsection we are going to present the used libraries and the interface followed by the user registration component and finally the algorithm used by our cache replacement policy, that hooks on ehcache, as explained below.

3.1.1 Relevant Libraries

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

Netty[7] This is the library used for handling user and server connections and parsing the received HTTP messages, extracting all of the information contained in the headers, including the HTTP status line;

ehcache[8] This is the library that is used to handle cached pages in memory and persist them to disk, also offering an way to search for pages given their URL and to provide hooks (using the Policy interface[9]) so that we can use our custom VFC replacement policy;

slf4j and logback[10] These are the libraries used to handle logging in our cache and allow us to see which cache consistency method is being used.

htmlparser[11] This is the html parser library used, to parse the html pages and extract the links from it.

3.1.2 User registration

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

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

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

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

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

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

3.1.3 Page Replacement Algorithm

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

So our replacement strategy, for all pages in the input vector that have VFC information, sees if the page distance to its pivot is lower than six and the frequency is greater than 6 and then adds the value of the distance/(frequency+1) to the score of that page. It also sees if the page recency is greater than six and if the frequency is less than 6 and then adds the value of recency/(frequency+1) to the score of that page.

After this the page with the lowest score is selected to be removed.

3.2 VFC-Wiki

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

Also, XWiki allows us to receive events and store arbitrary information along with the wiki pages and the plugin listens to the creation, change or removal of XWiki documents, so according to the event we do one of the actions, mentioned in the paragraphs, below.

So our plugin listens to the following events:

Document created This event is fired after a document (this includes a user) is created, and if a new user is created, the VFC information relative to the user is parsed and his or her watched pages are placed in the consistency zones defined by the VFC information of that user;

Deleting Document This event is fired before a document (or user) is removed and if a user is removed then the VFC information relative to that user stored in the wiki pages is removed;

Updating Document This event is fired before a document (or user) is updated and if it is a document that is updated, then the list of children of the document is stored by the plugin.

Document Updated This event is fired after a document (or user) is updated and in the case of a user the documents that have some connection to that user, by virtue of VFC are placed in their new consistency zones, if the VFC information of the user is updated, in the case of updates to the list of watched files, then the pages that are no longer related to the user will be removed from the consistency zones of the user and the new ones will be placed under user control. If a document is updated, then the VFC information of the removed children (or links) that was created by virtue of that link is removed and in the cache of new children, the VFC information is added to the new children in a recursive way.

3.3 Summary

In this section we have addressed some details of the implementation of both wiki and the web cache. Also, for the web cache we presented some of the major libraries that were used and why they were used.

4 Evaluation

After discussing the architecture, the algorithms and the relevant implementation details, we are going to evaluate the project in qualitative and comparative terms. In the comparative terms, we are going to compare VFC-Cache with other caches, like Squid and Polipo, in different configuration scenarios.

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

The tests were executed with the client and cache server all running in the same machine. All of the quantitative and comparative tests, were done using an automated tool and a list of visited urls (in order to simulate what a real user would do) together with a list of bookmarks selected from those urls. Then, several features where measured for each of the tested caches.

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

4.1 Quantitative Evaluation

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

4.1.1 Automated Tests

In this subsection we are going to compare the VFC scheme with the regular http scheme using the developed base cache, focusing on the number of cached documents, but before presenting any graphics we would like to highlight that the pages used for the quantitative and comparative tests are listed in the appendix. So in the number of cached documents (Figure 1), we see that one of the main objectives of VFC, that is to reduce the number of unimportant cached pages was successfully achieved, with a ratio of 0.26 : 1 pages from VFC to standard http, meaning 26.08% of pages or in overall less 74%. Which means that we have more space to store more interesting pages, using the same memory space as other caches.

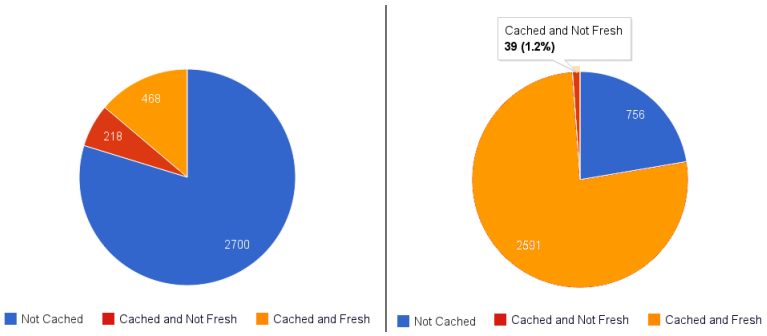

Fig. 1. Number of cached pages, VFC on the left and standard http on the right

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

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

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

4.1.2 Fifa 1998 tests

In this subsection, we are going to present a synthetic test using the fifa 1998 website access log data. For this test we have considered the logs of July 15th of 1998, then we have filtered the first 80000 entries of that log (after a conversion to the common log format) and then we have selected users number 55, 60, 79, 111, 210, 245, 179, 366, 381, 388, 465, 595, 623, 638, 746, 998, 1509 and

1548 which we gave as common interest the fact that they are all interested in french pages and hence the their set of pivots involve pages that begin with the url "/french/*".

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

So given the results presented in figures 2, we notice that although the number of wasted pages is high it is close to the number of not wasted pages, even if the number of wasted bytes measured is higher than the number of not wasted bytes.

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

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

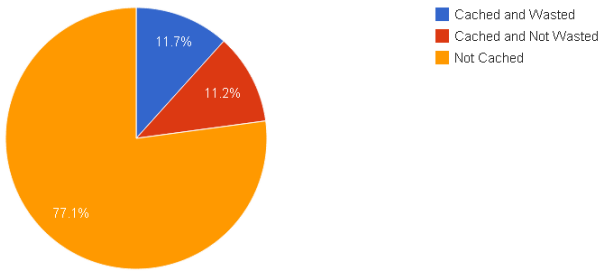

Fig. 2. Results of the fifa 1998 test in number of pages

4.2 Comparative Evaluation

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

So in the number of cached documents (Figures 1 (presented in the quantitative tests) and 3) and given that in the second figure, the first graphic (counting from the top left of the figure) is from Squid v2 using GDSF, next to the right we have Squid v2 using the LRU Heap, next to the right we have Squid v2 using the LFU-DA, in the next row, we have Squid v2 using LRU, then Squid v3 using GDSF, then Squid v3 using LRU Heap, in the next row we have Squid v3 using LFU-DA and finally Squid v3 using LRU.

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

In terms of latency, the VFC algorithm can reduce the load for the most used pages, in spite of behaving almost like a direct cache for the pages that are not bookmarks. Also our cache is unable to parse CSS files or javascript files which can force the browser to transfer more pages, particularly images that are used in the background of the page or as the background of buttons or other visual elements.

This is noticeable since in average most pages used in the test and in the bookmarks, retrieved at least two elements without the usage of the VFC algorithm, being mostly images and pieces of HTML pages, used in ads, that are produced using javascript. In terms of memory usage and cpu usage this is where most differences are and these can be attributed to the fact that most of the commercial caches are fine tuned to extract the most performance out of the operations they do.

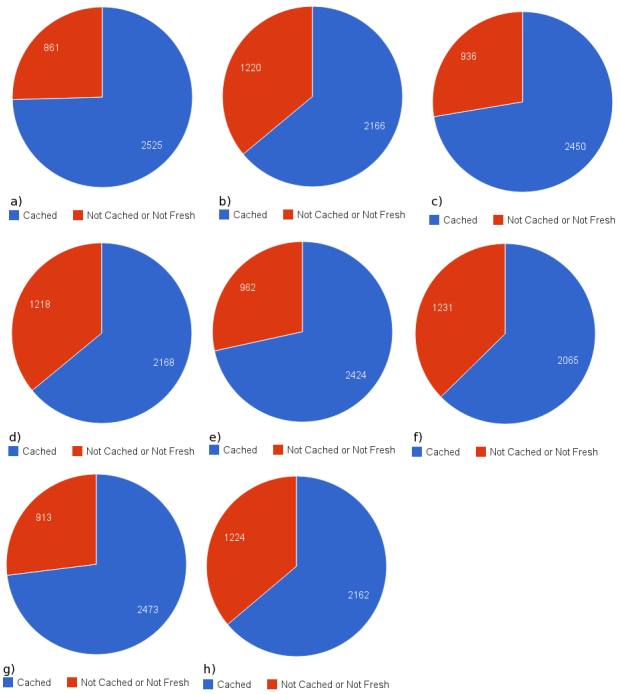

Fig. 3. Cached pages of the other caches, where a) is Squid2 with GDSF, b) is Squid2 with LRU Heap, c) is Squid2 with LFU-DA, d) is Squid2 with LRU, e) is Squid3 with GDSF, f) is Squid3 with LRU Heap, g) is Squid3 with LFU-DA and h) is Squid 3 with LRU

Also in the memory aspect, all of the other caches were coded in C or C++, while our implementation was coded in java, so the java virtual machine, simply allocated a predefined amount of memory independent of its usage or not by the program.

4.3 Summary

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

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

5 Related Work

In this part 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.

6 Vector-Field Consistency

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

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

Each consistency zone has a given consistency degree (Fig. 4), and consistency zones are ordered so that as one moves away from the pivot the objects placed have less in common to the given pivot (ie. are more distant).

The consistency zones are shaped like a cube (or square, depending on the number of dimensions or features one wants to monitor), where each consistency zone is defined by a range that goes from the previous consistency zone (or pivot, in the case of the first zone), to a certain maximum VFC vector value.

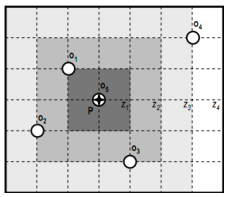

Fig. 4. Example of a consistency zone

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

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

In VFC, an object or pivot may also belong to more than one view (Fig. 5), where each view has its own set of pivots and therefore one pivot or object may generate more than one consistency zone or belong to more than one consistency zone, in the case of pivots and objects respectively, where a view can be associated with a user, a game field layer, etc.

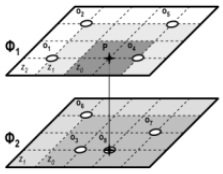

Fig. 5. Example of multiple views

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

7 Conclusions

At this paper we have presented the the architecture of our solution, where we analysed our adaptation of the VFC algorithm to the cases of both the wiki and the web cache.

Next we provied some details about the implementation of our solution, where for the web cache we presented some of the major libraries that were used and why they were used and some remarks about the registration of users and the page replacement algorithm. In the case of the wiki, we described the actions performed when the various events related to wiki users and documents happen.

After this we discussed the evaluation tests, that where divided into quantitative and comparative tests, some of them involving other caching solutions.

Finally we presented some of the related work, where we explained the VFC algorithm, that we intent to use in our cache solution and finished with an overview of the past usages of the VFC algorithm.

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

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

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

Possibility of cycles in the graph of pages Since we track, the links between pages it is possible that while following links, the resulting graph is a cyclic graph, which might pose some implementation problems;

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

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

Freeform of the HTML We also have a problem with the HTML pages, since usually HTML is not as strict as XML and some browsers also tolerate some semantic errors within HTML, which makes that our cache also has to behave like a browser while parsing pages and be as forgiving as the most relaxed browser in terms of HTML semantic, adding to the processing time and memory amount since some of those errors might require some transformation to the DOM tree, which voids the use of a more lightweight approach like SAX;

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

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

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

References

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

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

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

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

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