Cache Clearing System

Babovic, Zoran; Jovic, Darko; Cakarevic, Vladimir; Milosavljevic, Ivan; Stevanovic, Marija; Minic, Predrag; and Milutinovic, Veljko

Abstract—*In the introductory part, this paper defines the general environment for this research and defines the terms of interest. Then we define the research problem: How to implement a system which clears the cached content from the production web sites. The essence of the proposed solution is presented in the part three: The main part of the system, CacheBot, consists of two multithreaded applications, called Hard Daemon and Soft Daemon, which are in charge of actually deleting or updating the contents on the web servers. The fourth part defines advantages that we made compared to an existing solution, as well as the conclusion.*

1. *Introduction*

THE Dow Jones Cache Clearing System is responsible for clearing the cached content associated with pages, sections and collections on the production sites (WSJ.com, Barrons.com). Content is cached in order to improve response times and resource utilization when serving pages requested by browsers. However, as a news site, it is important that the content is as current as possible. In order to achieve the optimal trade off on how often to clear cached content so as to maximize response times and conserve resources without unduly compromising the currency of information on the pages, complex business rules have been developed and implemented to dictate when, how often and in what manner to clear cached content.

2. *Problem Statement*

This part describes requirements and expectations established for this project. Required features were to make Hard Clear Cache – part of the system that is in charge for removing the files immediately; Soft Clear Cache – part of the system that replaces file with a newly generated file; Flexible configuration of URLs for soft and hard clear cache; guaranteed delivery of Cache Clearing. Any cache clearing requests should never become lost. The system had to be able to resubmit failed requests based on configurable parameters or properly log any persistent problems. System also has to be able to work in a number of abnormal conditions, to eliminate all duplicate cache-clearing requests from the message queue (as we have many of duplicate requests).

All of these requirements had to be designed in a simple robust way that emphasizes reliability, easy maintenance and high performance over flexibility and abundance of features. [1]

3. *Proposed Solution*

In order to clear cached content, the Cache Clearing System must be able to locate the exact content to be cleared. Content is identified by its content ID which is then resolved to one or more URL paths and file names. It is this URL path and file name that are cached on disk and used to serve all subsequent requests from user browser sessions. Once the Cache Clearing System determines that a particular cached content item is to be cleared (i.e., an editor has published an updated version of the content), it will either:

-Delete the cached URL path(s) and file name(s) associated with the content item from the disk. The next time a user requests this content item via a browser session, the absence of that URL on the disk will cause the Cache Clearing System to retrieve a fresh copy of the content from the one of the production rails, and generate a new URL path and file name on the production rail on which the content was originally requested. This method is referred to as Hard Cache Clearing.

-Proactively obtain a fresh copy of the content from the one of the production rails, generate a new URL path and file name and distribute that URL to all production rails. This method is referred to as Soft Cache Clearing.

In Figure 1 we can see the architecture of the Cache Clearing System. CacheBot is the main part of the system, and consists of two multithreaded applications, [2] called Hard Daemon and Soft Daemon, which are in charge of actually deleting or updating the contents on the web servers. Requests for Hard Clearing are inserted into CLEAR_CACHE_QUEUE table by CacheManager and read periodically by the Hard Daemon, while requests for soft clearing are defined in SOFT_CLEAR_CACHE table and read periodically by the Soft Daemon.

Web servers are grouped to serve requests from browsers faster. These groups are called *Elementary Targets* (or just *Targets*), and each web server in the target is called *Rail*. Every rail is

Manuscript received July, 2006. This work was supported in part by the Dow Jones Company.

Babovic Zoran, Jovic Darko, Cakarevic Vladimir, Milosavljevic Ivan, Stevanovic Marija, Minic Predrag, and Milutinovic Veljko are with Faculty of Electrical Engineering, University of Belgrade, Serbia.

defined by the unique name and the URL. Elementary targets can also be grouped to form higher-level targets.

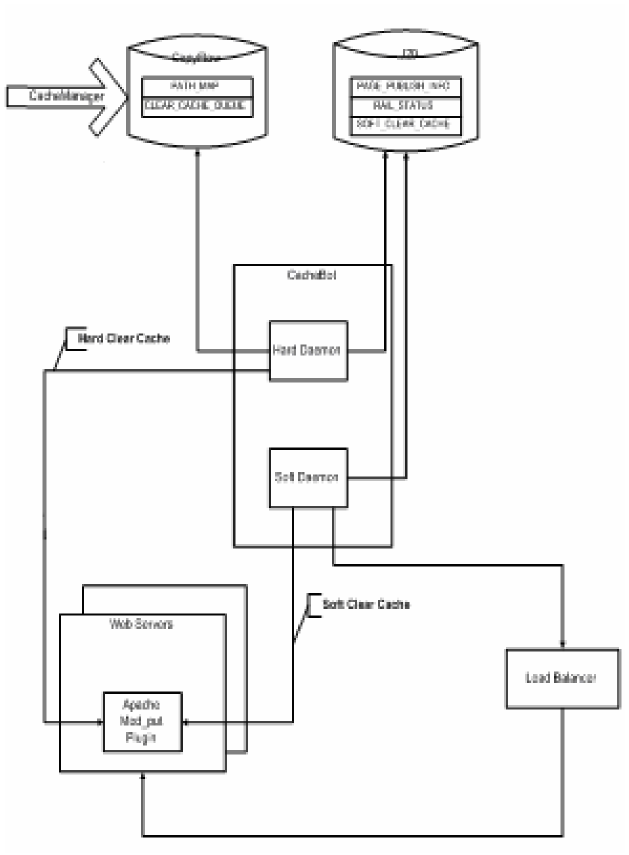

Figure 1: Cache Clearing System Architecture

Every request for cache clearing has corresponding target, which means that it needs to be executed on all rails belonging to the specified target. There are exceptions to this rule: if request has a target of higher level assigned and one or more elementary targets which could be soft cleared.

Every Apache web server contains mod_put plug-in, which makes possible to delete or update web content from the server with HTTP PUT and DELETE requests. Hard Daemon uses only HTTP DELETE requests for clearing the content, while Soft Daemon uses both PUT and DELETE requests for updating the content on the web server.

In Figure 1 is also shown a load balancer, which Soft Daemon targets to generate a new content that will be delivered to required rails.

3.1 Hard Cache Clearing

The Hard Clear Cache daemon provides the following functionality:

1. Fetches requests for Hard Cache Clearing from CLEAR_CACHE_QUEUE table,
2. Eliminates duplicate requests if they occur,
3. Checks the syntax of requests to ensure that they are well formed and valid,
4. Resolves partial URLs from requests by rules defined in PATH_MAP table,
5. Eliminates duplicate URLs which may occur during resolution phase,
6. Deletes contents from the appropriate rails by sending HTTP DELETE requests,
7. Deletes processed requests from CLEAR_CACHE_QUEUE table.

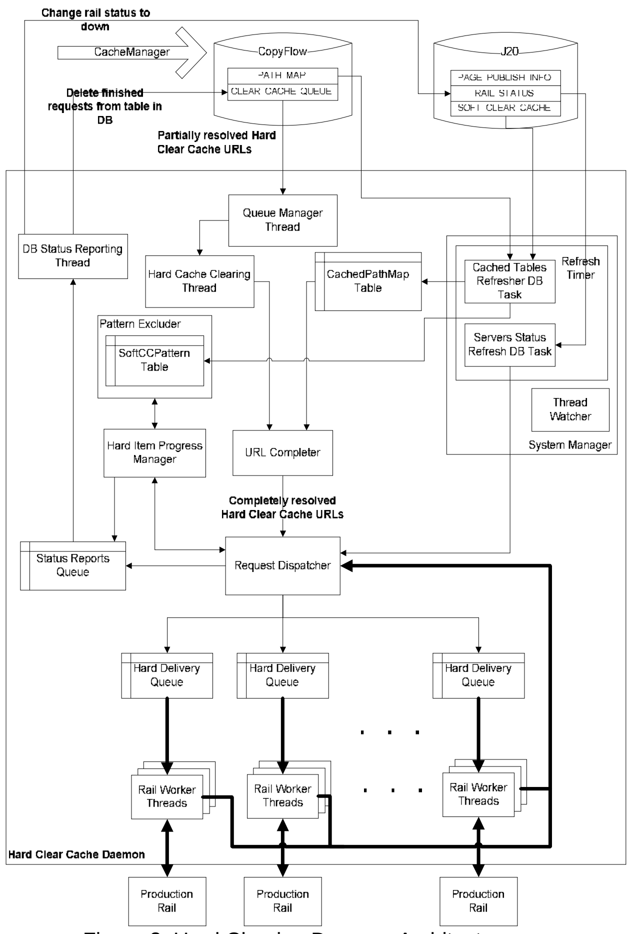

Figure 2: Hard Clearing Daemon Architecture

Hard Daemon fetches requests from the CLEAR_CACHE_QUEUE table periodically, with the configurable time interval. This is done by the Queue Manager Thread. If some requests are fetched from the table, Hard Daemon does not wait for the next time interval, but reads immediately from the database table in case that there are new requests available. If there are no requests in the table, Hard Daemon waits for the next time interval to check the CLEAR_CACHE_QUEUE for new requests. Duplicate requests (requests which have the same content id, content type and target) will not be fetched, but will be deleted on original request completion. For each fetched request, Hard Cache Clearing Thread creates object called Hard Item, which encapsulates request data (URL, target…). After requests are fetched, partial URLs are resolved based on the request’s content type and target, which are paired with path and extension fields from the PATH_MAP table. One partial URL can be resolved to several resolved URLs, which are generated by concatenating path, content id and extension. In this phase it is possible that some partial URLs are resolved to same URLs. These duplicates are also eliminated by the CacheBot. URL resolving is done by the URL Completer component.

After URL completing, Hard Item with completed URLs is passed to Request Dispatcher. It uses Hard Item Progress Manager for determining on which rails request should be executed and excluding URLs which are configured for Soft Cache Clearing (Pattern Excluder). Hard Item Progress Manager is also responsible for tracking of request execution and handling of duplicates of currently processing requests.

After Item processing in the Hard Item Progress Manager, for every resolved URL an object called URL Request is created and assigned to Hard Item. Request Dispatcher takes list of URL Requests from the Item and dispatches them to appropriate rails (i.e. inserts them in corresponding delivery queues).

There is a configurable number of Rail Workers (running in separate threads) dedicated to each rail. Those workers take URL Requests from the Delivery Queue and execute them on the appropriate rails. Rail Workers can handle uploading (used in Soft Cache Clearing) and deleting of contents from the rails by using HTTP PUT and DELETE requests. In case of Hard Cache Clearing only deleting is performed. Depending on the success of request execution, Rail Workers notify Request Dispatcher and Hard Item Progress Manager of the execution status.

If rail worker detects that rail is down, it notifies Request Dispatcher which creates rail status report for updating RAIL_STATUS table.

Status of the rail is kept in the RAIL_STATUS table, and there are two possible states of the rail: UP and DOWN. Internally, CacheBot has two more states which are not stored in the RAIL_STATUS table, but only logged into a log file. These states are: SLOW and DELAYED. When the rail is DOWN, requests are not sent to it. Rail can be set to DOWN state by either CacheBot or administrator. CacheBot sets rail state to DOWN when it determines that server is not accessible (gets timeout during connection attempt), while administrator just changes the value in the RAIL_STATUS table for particular rail. On the other hand, rail can be set to UP only by administrator, by changing the state to UP in the RAIL_STATUS table. Rail statuses are periodically read by the Server Status Refresher DB Task which then updates the rail statuses in the CacheBot governed by the following rules:

- The state of the rail will be changed to DOWN if the rail state is not already DOWN and the time of last change (recorded in the CacheBot) is before than the change time stored in the table.

- The state of the rail will be changed to UP if the state is DOWN and the stored time of the last change is before the change time from the table.

SLOW state represents the situation when the average execution time of the configurable number of last requests on particular rail is greater than the configurable threshold.

DELAYED state represents the situation when number of requests waiting for execution in the queue for particular rail is greater by the configurable threshold than number of requests in the queue for the rail of the same elementary target which has the minimal number of requests in it.

All database reports are inserted into the Status Reports Queue and there is a DB Status Reporting Thread which will take tasks from the queue and execute them.

3.2 Soft Cache Clearing

The Soft Cache Clearing Daemon provides the following functionality:

1. Fetches requests for Soft Cache Clearing from SOFT_CLEAR_CACHE table, but only if they have been updated since the time of the last fetching. This occurs at regular time intervals,
2. Checks the syntax of requests to ensure that they are well formed and valid,
3. Communicates with the Apache Web server (via HTTP routed through the Load Balancer) in order to force the generation of an updated content file on a single production rail. New content is retrieved by inserting timestamp in the URL for which new content is needed and sending HTTP request with the new URL to the load balancer,
4. Uploads updated content on the appropriate rails via HTTP PUT requests,
5. Deletes generated page (with a timestamp added) from the rails via HTTP DELETE request.

In Figure 3 we can see the internal architecture of the Soft Daemon. The architecture is slightly different from the Hard Daemon architecture. Rail Worker Threads, Request Dispatcher, DB Status Reporting Thread, Rail Status Refresher Task, and System Manager, function in the same way as described in Hard Cache Clearing section. New elements are Soft Cache Clearing Thread, Soft Queue Manager Thread, and Soft Item Progress Manager. Soft Daemon fetches URLs for Soft Clearing from the SOFT_CLEAR_CACHE table by Soft Queue Manager Thread, but only if they have been updated since the time of the last fetching. This is checked by including PAGE_PUBLISH_INFO table in the fetch query.

The data needed for page update is stored in an object called Soft Item. Fetching is done periodically with the configurable time interval. Soft Queue Manager Thread behaves in the same way as in Hard Daemon, but reads cache clearing requests from different database tables.

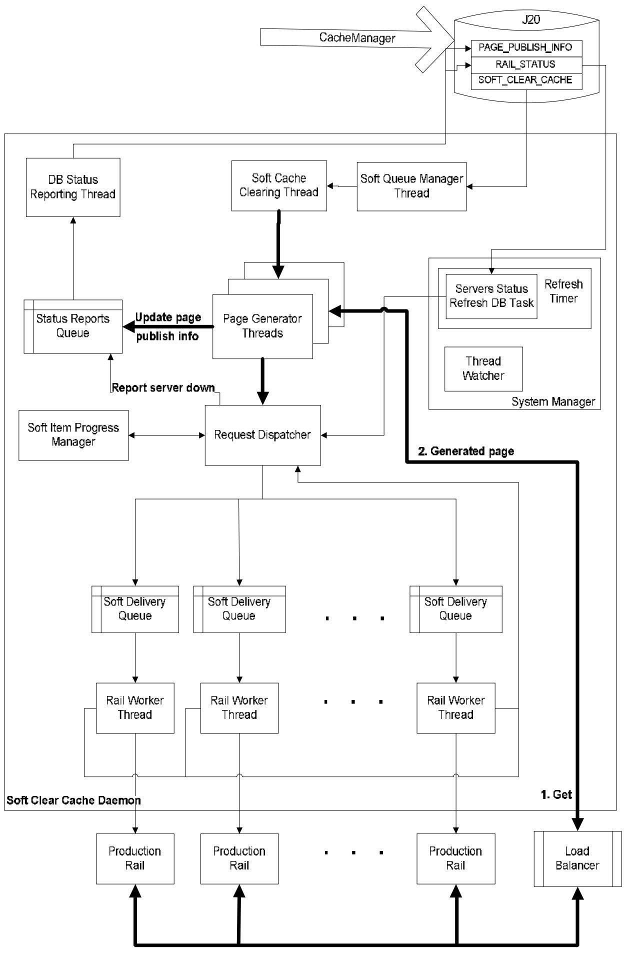

Figure 1: Soft Clearing Daemon Architecture

Page Generation Task is created for each Soft Item. This task executes GET HTTP request through Load Balancer and saves received page. It calls Request Dispatcher to dispatch generated page to rails. If an error occurs during page generation, the corresponding PAGE_PUBLISH_INFO table record will be updated so the page updating will be tried in the next iteration.

In order to get a new content for specific URL, Page Generation Task inserts a timestamp into the URL and targets Load Balancer which will recognize this type of URL and will generate a new page. Timestamp is in a form yyyyMMddHHmmssSSS and is inserted before the file extension.

After the page has been created, it is stored in the Soft Item object and passed to Request Dispatcher for dispatching to required rails. Request Dispatcher then calls Soft Item Progress Manager to eliminate possible duplicates and inserts page update request to all rails. Request is considered to be a duplicate if the request with the same URL and target is currently waiting to be processed on one or more rails. In case that duplicate has been detected, content of the original request is replaced with the new one, so rail will receive the latest version of the generated page and the old content will be deleted. Soft Daemon can store generated page content in a file or in a memory structure.

Each Rail Worker Thread will update page on its rail via HTTP PUT request. When some page is updated, Rail Worker deletes temporary generated page(s) from the rail, if there is any, via HTTP DELETE request with a URL which contains -*.* (delete extension) at the end. This will delete the generated page with a timestamp in a URL.

Request Dispatcher behaves in the same way as in Hard Daemon. This means that rail states and their behavior is the same as described in Hard Cache Clearing section.

All database writes are done through Status Reports Queue by DB Status Reporting Thread as in Hard Clearing.

The System Manager works in the same manner as in Hard Daemon.

4. CONCLUSION

In this part we will summarize facts that we gained in this project and also, we will make a comparison with one existing solution, but currently used system will not be described now.

Main problem with the current solution is that each thread has to wait as long as the last working server finishes with certain page. In our solution this problem is resolved by independent threads. This means that servers do not wait each other, which, obviously speed up the whole system. Weakness of this solution is bigger memory consumption, which is being recompensed with the fact that there is no possibility of overhead situations (overhead in the current solution is the thread information on which server it has to be executed).

In the current solution of the problem, pages are being updated all at the time, with a certain time stamp. The page is updated only if it is changed since the last updated time. This means that the superfluously updating is being eliminated.

Page tracking is optimized; primarily by decreasing the data base utilization (one table is even excluded from the data base used for our solution).

Servers are being set to UP and DOWN state only by the administrator, which implies that pages are being sent even to the DOWN rails, in case that administrator did not notice and change server state changes. Our solution gives much more reliable system. Beside administrator, which works in the same way, the server itself sends the message to the data base so it can be updated with the new server state. This change is now known through the whole system and the pages will not be sent to these servers.

Having in mind those advantages that we gained in this project, we think that all the requirements specified in the problem statement are satisfied, but also, some extra advantages are scored.

REFERENCES

[1] Horton, I., “Beginning Java 2,” *Wrox Press Ltd*, Birmingham, 2000.

[2] Oaks, S., Wong, H., “Java Threads 2nd Edition,” *O’Relly & Associates*, January, 1999.