EUROPEAN COMMISSION

Community research

Climate Information Portal

CLIPC Milestone (N°: 17)

Caching for visualisation

File name: MS17 CLIPC Caching for visualisation.docx

Dissemination level: PU (public)

Author(s): *Alessandro D'Anca (CMCC), Sandro Fiore (CMCC), Laura Conte (CMCC)*

Reviewer(s): Wim Som de Cerff, Maarten Plieger

Release date for review: 31/05/2016

Final date of issue: 31/05/2016

Revision table			
Version	**Date**	**Name**	**Comments**
1	May 2016	Concept	Final integration of caching system and visualisation components

Abstract

Description of the caching system adopted in the CLIPC project for speedup the retrieval of outputs and its internal features and components.

Project co-funded by the European Commission's Seventh Framework Programme (FP7; 2007-2013) under the grant agreement n°607418

Content

Introduction

The objective of the developed caching system is to allow the CLIPC users to take advantage of a fast responsiveness and a real time retrieval and visualisation of the requested indicators. With the aim of avoiding the replication of the processing tasks already performed, the caching module represents an intermediate layer between the user client, which requests the execution of certain operations on a set of input data, and the processing backend system, able to perform the requested job, to produce the outputs and make them available to the user for downloading or visualisation. Therefore, being an intermediate component, the caching system, developed to support the processing activities in the CLIPC project, has been modeled and implemented in order to act as an interceptor with respect to the upper layer eventually forwarding the received inputs to the underlying layer in a transparent manner.
Moreover, the caching system architecture has been designed to be independent of the specific analysis engine running in the back-end: icclim[1] or Ophidia framework[2][3] represent data processing services which will take advantage of the developed mechanism.
In the following of this document the caching system features will be analyzed in terms of objectives and architecture. The developed software modules and the steps performed for the integration in the CLIPC back-end processing system will be described and future improvements will be presented.

1. Caching System Definition

The caching system developed in the context of the CLIPC project has been implemented as a middle-layer able to speed-up the retrieval of the results of the users' computations.
More in detail, the concepts of "cache hit" and "cache miss" are the bases which drove the development phase. In this perspective, the caching module is able to simulate to the upper layers (the client of the request) the execution of a particular job by intercepting the incoming request and by replying with the correct results without involving the underlying layers designated to the real execution of the task: this represents the behaviour in the "cache hit" case. On the other hand, during the "cache miss" case, the caching system transparently forwards the incoming request parameters to the back-end in order to start the execution of the job and the computation of the needed analysis.
The following figure depicts the caching system behaviour when a "cache hit" or a "cache miss" occurs.

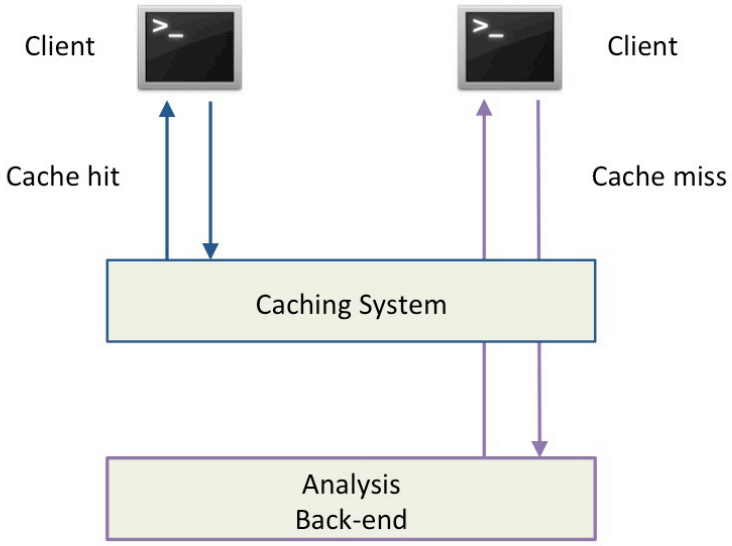

Figure 1. High level representation of a "Cache hit" and a "Cache miss"

In order to implement this mechanism, the caching module has been developed as a back-end software component which manages the output repository fed by all the processed jobs submitted by the users. By querying a "caching catalog", the developed module is able to send back to the client the result if it is already available in the cache as output of a previous performed computation (cache hit); if not, it triggers the data analysis pipeline needed to compute the expected climate indicator (cache miss). In this case the result is both sent back to the client and stored in the caching catalog.

2. Caching System Architecture

The Caching System has been developed taking into account the pre-existing back-end environment in which the module would has been integrated. The back-end system relies on WPS[4] for managing the incoming requests so the WPS specification has been used as interface between the client requests and the underlying processes. PyWPS[5] has been used to implement the WPS specification in a Python environment and the processes, representing the execution algorithms, are coded in Python language. So, in order to ease the integration phase, the same Python language has been chosen to implement the caching system. MySQL[6] has been chosen as DBMS for hosting the caching catalog able to collect the information related to the processed jobs.
The next figure shows the interactions between the involved components and the flow of the operations in a common real use case:

1. the caching system, included in the PyWPS context, uses the request input parameters as search key for querying the caching catalog;

2a. a cache hit triggers the delivering of the found results to the client by the caching service;

2b. a cache miss triggers the back-end processing needed to compute the expected climate indicator;

3b. input data are taken from the data repository;

4b. at the job completion, the results are sent back to the client and the caching service extracts from the request input/output the needed information for enabling a cache hit for similar incoming requests.

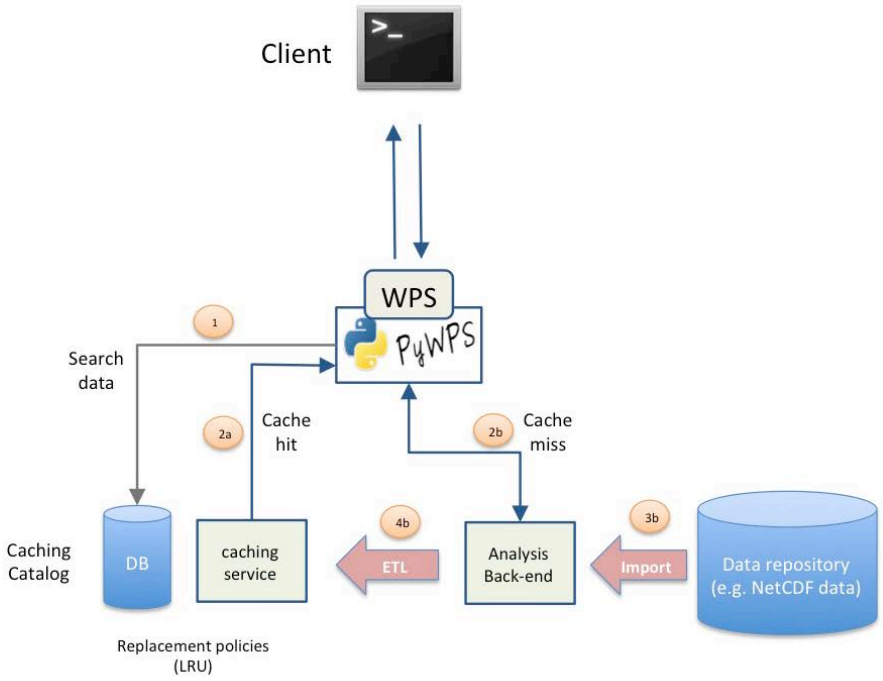

Figure 2. Flow of the operations and interactions between the caching system and the analysis back-end

In order to properly manage the disk space devoted to the caching system, a replacement policy has been adopted; specifically, a LRU (Least Recently Used) policy has been implemented with the aim of erasing from the caching catalog the least used entries and the related information. By using a predefined (and configurable) threshold, only a limited set of requests and associated outputs will be stored in the caching catalog.

3. Caching System Installation

The development of the caching system has been driven by the aim of easing the installation in a pre-existing processing system.

Concerning the caching catalog, as stated in the previous section, it has been implemented exploiting a MySQL database table; it contains the following information:

- *id*: unique identifier (primary key) of the related entry;
- *query*: used to store the input parameters which uniquely identify the input request;
- *result*: contains the output associated to the stored request;
- *status*: could be used for distinguishing between requests currently under processing and ready;
- *last_request*: timestamp related to the last accessed date;
- *hit_count*: counter which represents how many times the associated result has been accessed; it could be useful for statistical purposes.

The first step concerns on the installation of a MySQL DBMS on the target host and the import of the ddl (sql source file) related to the caching catalog.

The software module named *clipccache.py* contains the python class and the functions able to manage the different caching functionalities. Specifically, the following structures has been implemented:

- a python class named *clipccache* which manages the connection to the MySQL caching catalog, the attributes related to the user's request and the LRU threshold statically configured as a global class variable;
- a method named *cache_search* which implements the searching phase by querying the caching catalog; in case of a cache hit, it updates the *last_request* timestamp with the current date;
- a method named *insert_new* which implements the ingestion of a new entry in the caching system catalog in case of a cache miss.

The second step of the installation regards the deploying of the caching service on the target host environment simply by copying the *clipccache.py* file in the PyWPS *processes* directory.

Finally, in order to enable the caching mechanism for a WPS process, the related python file needs to be modified adding few lines of code. Specifically, the two methods *cache_search* and *insert_new* need to be called respectively at the beginning and at the end of the process.

4. Integration of the Caching System in CLIPC/climate4impact

In order to make the caching system even more compliant with the CLIPC-climate4impact[7] environment and to ease the integration in the pre-existing processing environment, few modifications have been performed.

The climate4impact system relies on a PostgreSQL DBMS so the caching system has been adapted in order to store the caching information in a PostgreSQL table. The use of standard SQL speeded up this process so no query has been modified. The new PostgreSQL python drivers have replaced the MySQL one and a new database table has been created in the PostgreSQL DBMS.

In addition, other improvements have been performed to better integrate the caching system in the climate4impact environment concerning the separation of the users space and the management of the output files deletion. Regarding the former, the system has been modified in order to store the caching information on a per-user basis to prevent a user to access data outside from his own basket. Concerning the latter, a new routine has been implemented for checking the existence of the output file extracting its path from the information stored in the caching catalog. This new feature manages any potential file deletion made in the user's output basket.

5. Conclusions & future work

The caching system to support the CLIPC project activities represents an enhanced software module developed with the aim of improving the performance of the processing layer by speeding up the retrieval of the results already available by previous job submissions. A service based on cache hits and cache misses has been implemented and easily deployed in the CLIPC analysis back-end; a LRU policy has been also developed in order to preserve the disk space occupancy.
As future improvements, the caching system could be made aware of the running processes distinguishing between ready and in progress jobs. Furthermore, a common repository could be setup and managed for storing the results of the processes without differences among the users space still keeping the separation of the users' basket. Finally, a procedure for checking the coherence between input data and results could be implemented in order to manage any potential update of the input files.

References

[1] icclim - http://icclim.readthedocs.io/en/latest/
[2] Ophidia Framework - http://ophidia.cmcc.it/
[3] Ophidia Framework - S. Fiore, A. D'Anca, D. Elia, C. Palazzo, I. Foster, D. Williams, G. Aloisio, "Ophidia: A Full Software Stack for Scientific Data Analytics", proc. of the 2014 International Conference on High Performance Computing & Simulation (HPCS 2014), July 21 – 25, 2014, Bologna, Italy, pp. 343-350, ISBN: 978-1-4799-5311-0
[4] WPS - OGC 05-007r7, OpenGIS® Web Processing Service, Version: 1.0.0 at http://portal.opengeospatial.org/files/?artifact_id=24151
[5] PyWPS - http://pywps.org/
[6] MySQL - https://www.mysql.com/
[7] Climate4impact - https://climate4impact.eu