FP7-ICT-611140 CARRE

Project co-funded by the European Commission under the Information and Communication Technologies (ICT) 7th Framework Programme

D.5.2. Query and Performance Enhancements

Z. Deng, E. Liu, A. Third, X. Zhao, Y. Zhao

October 2015

CARRE Contacts

Project Coordinator:	Eleni Kaldoudi	kaldoudi@med.duth.gr

DUTH Democritus University of Thrace	Eleni Kaldoudi	kaldoudi@med.duth.gr
OU The Open University	John Domingue	john.domingue@open.ac.uk
BED: Bedfordshire University	Enjie Liu	Enjie.Liu@beds.ac.uk
VULSK: Vilnius University Hospital Santariškių Klinikos	Domantas Stundys	Domantas.Stundys@santa.lt
KTU Kaunas University of Technology	Arūnas Lukoševičius	arunas.lukosevicius@ktu.lt
PIAP Industrial Research Institute for Automation & Measurements	Roman Szewczyk	rszewczyk@piap.pl

Disclaimer

CARRE is a Specific Targeted Research Project partially funded by the European Union, under FP7-ICT-2013-10, Theme 5.1. “Personalized health, active ageing & independent living”.

Document Control Page

Project	
Contract No.:	611140
Acronym:	CARRE
Title:	Personalized Patient Empowerment and Shared Decision Support for Cardiorenal Disease and Comorbidities
Type:	STREP
Start:	1 November 2013
End:	31 October 2016
Programme:	FP7-ICT-2013.5.1
Website:	http://www.carre-project.eu/
Deliverable	
Deliverable No.:	D.5.2
Deliverable Title:	Query and Performance Enhancements
Responsible Partner:	BED
Authors:	(in alphabetical order) Z. Deng, E. Liu, A. Third, X. Zhao, Y. Zhao
Input from:	All partners
Peer Reviewers:	G. Drosatos (DUTH), R. Ugodzinski (PIAP)
Task:	T.5.2. Query design and query performance enhancement
Task duration:	10 months: 1 January 2015 to 31 October 2015
Work Package:	WP5: Data management & visual analytics for empowerment
Work Package Leader:	BED – Enjie Liu
Due Date:	31 October 2015
Actual Delivery Date:	31 October 2015
Dissemination Level:	PU
Nature:	R & D
Files and format:	Deliverable report: 1 pdf file
Version:	02
Status:	☐ Draft
	☒ Consortium reviewed
	☒ WP leader accepted
	☒ Coordinator accepted
	☐ EC accepted

Document Revision History

Version	Date	Modifications	Contributors
v01.1	08 July 2015	Template and content	Xia Zhao
v01.2	19 Sept 2015	First daft	Enjie Liu, Zhikun Deng
v01.3	29 Sept 2015	Second draft	Enjie Liu, Youbing Zhao
v01.4	03 Oct 2015	Third draft	Allan Third
v01.5	08 Oct 2015	Forth draft	Zhikun, Enjie Liu
v01.6	25 Oct 2015	Ffifth draft	George Drosatos
v01.7	28 Oct 2015	Sixth draft	Zhikun Deng, Youbing Zhao
v01.8	29 Oct 2015	Seventh draft	Zhikun Deng, Enjie Liu
v02	31 Oct 2016	editing for uniformity	E. Kaldoudi

Table of Contents

List of Figures

Executive Summary

CARRE personalised patient empowerment and decision support services require presentation and analysis of a large volume of heterogeneous data and metadata as harvested from a variety of data sources including sensors, risk factors, PHR, decision support, etc. Without proper tools, it is almost impossible to achieve this goal.

Work Package 5, Task 5.2, “Data Management & Visual Analytics for Empowerment”, is proposed to meet this challenge. In Task 5.2 “Query design and query performance enhancement”, is to investigate general approaches in enhancing the RDF query performances. General guidelines are provided, considering there will be potentially a large number of users of the CARRE system, who will generate large dataset from sensors and PHR.

This document is a deliverable report of D5.2 “Query design and query performance enhancement” of WP5 in CARRE project. Some advices for the future implementation will be provided. Initial tests are conducted as a proof-of-concept. The actual effect of the improvement will be seen after finishing the task D5.3: Advanced visual analytics module.

About CARRE

CARRE is an EU FP7-ICT funded project with the goal to provide innovative means for the management of comorbidities (multiple co-occurring medical conditions), especially in the case of chronic cardiac and renal disease patients or persons with increased risk of such conditions.

Sources of medical and other knowledge will be semantically linked with sensor outputs to provide clinical information personalised to the individual patient, to be able to track the progression and interactions of comorbid conditions. Visual analytics will be employed so that patients and clinicians will be able to visualise, understand and interact with this linked knowledge and take advantage of personalised empowerment services supported by a dedicated decision support system.

The ultimate goal is to provide the means for patients with comorbidities to take an active role in care processes, including self-care and shared decision-making, and to support medical professionals in understanding and treating comorbidities via an integrative approach.

Terms and Definitions

The following are definitions of terms, abbreviations and acronyms used in this document.

Term	Definition
API	Application programming interface (API) is a set of functions and procedures that allow the creation of applications that access the features or data of an operating system, application, or other service
BPM	Beats Per Minute
BSBM	Berlin SPARQL Benchmark
CPU	Central Processing Unit
C#.NET	C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. C# programs run on the .NET Framework, an integral component of Windows that includes a virtual execution system called the common language runtime (CLR) and a unified set of class libraries
C#.NET MVC	Model View Controller
DSS	Decision Support System
EC	European Commission
eHealth	Electronic Health
EHR	Electronic Health Record
EU	European Union
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICT	Information and Communication Technologies
MedLinePlus	The National Institutes of Health's Web site for patients and their families and friends, http://medlineplus.gov.
OAuth	Open Standard to Authorization
Patient ID	Personal Identification Number
PHR	Personal Health Record
PubMed	Free search engine accessing primarily the MEDLINE database www.ncbi.nlm.nih.gov/pubmed/
RDF	Resource Description Framework - a standard model for data interchange on the Web.
SPARQL	RDF query language, that is, a query language for databases, able to retrieve and manipulate data stored in RDF
TTL	Time-To-Live
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language - a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable

1. Introduction

According to the DOW, the task will focus on:

- Considering cache mechanisms for reducing query response network latency and will implement the backend query function in a distributed environment, such as through cloud computing.
- Adaptations and optimizations at facilitating the process of graph visualization generation and supporting interactive operations is also part of this task.

From user perspective, query performance enhancement is needed due to the following factors:

- User experience (1 sec delay in page response can result in a 7% reduction in conversions) If an e-commerce site is making $100,000 per day, a 2-second page delay could potentially cost you $2.5 M in lost sales every year[1]. Although we are not building a e-commerce web system, and our concern is the user engagement. In which case, user experience is also an important feature for us.
- Large volume of visitors access to the repositories
- Increase user engagement via good customer experiences
- Limitation brought by the tools used in development

From technology point of view, query performance enhancement is needed due to the following factors:

- For triple stores, in typical application scenarios only relatively small parts of a knowledge base change within a short period of time.
- The majority of triples remain unchanged. Hence, most queries will return the same results even after the occurrence of changes on the knowledge base.

In the previous deliverable D2.1, four high – level visual analytics use cases have been identified, they are:

- UC_Vis_04 : The goal of this use case is to allow patients to understand their disease progression;
- UC_Vis_05 : The goal of this use case is to allow patients to understand their disease progression based on personal monitored data;
- UC_Vis_06 : The goal of this use case is to allow patients to understand their disease progression if they change their lifestyle;
- UC_Vis_07: The goal of this use case is to allow patients to understand their disease by comparing their personal state with current medical evidence.

The visual analytics will provide all necessary functions for the above cases.

2. Analysis of CARRE RDF data repositories

2.1. Status of CARRE RDF Data Repositories

Storage and management of RDF data is implemented using the open source version of Virtuoso Universal Server. More specifically, the OU team has installed the latest version of Virtuoso 62 (6.4) on a 6.6 CentOS (Santiago) machine with 8GB of RAM and a dual core 2.50GHz Xeon with 30GB expandable Hard Disk space. Virtuoso is both a database and a middleware engine3.

At the time of writing, the repository contains 3.47 million RDF triples occupying 1.1Gb, representing risk associations and multiple sensor data entries per day for approximately 40 users, some of whom have data

[1] https://blog.kissmetrics.com/loading-time/

[2] http://virtuoso.openlinksw.com/

[3] http://virtuoso.openlinksw.com/virt_faq/

covering a period of up to two years. The current server configuration is therefore more than sufficient for current load and for anticipated use during CARRE evaluation, with the ability to cache the entire database in memory whenever needed. The server is easily expandable in terms of both memory and storage beyond these relatively modest specifications should the requirements increase.

The CARRE repositories are divided by privacy concerns. The private repository stores data relating to individual patients, in a secure and access-limited fashion, with each patient's data in a separate and restricted RDF graph. All data from personal sensors, personal health records and any decision support services recommendations for an individual are stored and queried from that individual's private graph. This further limits the potential performance requirements, as there is never a need (nor, due to privacy concerns, an option) to query more than one private graph at the same time.

The public repository stores general medical knowledge relating to risk associations, evidence and observables, and is available for public querying without authentication, as it contains no personally identifying data for any patient and serves as a general-purpose resource for medical knowledge in a semantic format.

The RESTful API to access CARRE data is described in detail in D.4.1 and D.4.2, including the privacy and security features for private data. An authenticated and approved application can retrieve an access token per user from the API, which is used to authenticate all calls which access that user's private graph. API calls relating to public data may be performed without any token.

SPARQL4 is the standard query language for RDF data. The API allows arbitrary SPARQL queries to be submitted via an authenticated call, and also contains several helper methods which wrap specific common queries in an optimised way (for example, retrieving a daily summary of sensor readings for a user).

2.1.1. Public RDF

The CARRE public repository is used to store the risk associations that gathered by the project, and reported in D2.2. This repository is meant to be linked to the Linked Open Data cloud, and used as public knowledge. In the CARRE public repository, the data include:

- CARRE risk associations
- CARRE ontologies described in D2.4
- CARRE educational data

2.1.2. Private RDF

In CARRE, the private data repository stores the patient related data, which mainly come from the PHR,and the sensor data monitoring. The data comprises personal diary data which includes the biomarker records (with the necessary data of observables as indicated in the risk associations) and life style tracking data. The development of the private RDF repository is carried out by OU and the progress had been reported in D2.5, D4.1 and D4.2.

For demonstration purposes, there are currently 18 virtual patients defined by doctors based on data selected from real patients and were presented in Annex 1 of Deliverable 5.1. Three virtual patients (as described in D5.1) are selected from them as detailed use cases to be used in designing the visual analysis study to perform lifestyle management and risk assessment. In the following subsections we will list the basic information of thevirtual patients that are used in this deliverable.

Private data includes:

- Sensor data for bio-marker and activities tracking: Biomarker data and life style tracking data that are collected by sensors, mostly can be used at home.
- Personal Health Record (PHR) data:
    - Patient data will be retrieved from PHRs, Vivaport and HealthVault, by CARRE PHR data aggregator using PHR's APIs. Then data is mapped and converted to format used in OU

[4] http://w3.org/TR/sparql11-query/

CARRE RDF and inserted into Private RDF via common interfaces and common data exchange method used by RDF repository (SPARQL syntax and methods).

- CARRE PHR manual data entry system is another option to enter patient records. It is also used for anonymous data collection.

- Web lifestyle data based on patient's web searches: Web lifestyle data is particularly patient's intentions that are extracted from her online interaction with the web search engines (Google, Bing and Yahoo).
- Personalised decision support results: The DSS will communicate with OU CARRE RDF Repositories over SPARQL to retrieve RDF from both Repositories. For DSS service the CARRE semantic repository will be a RDF store accessible as a RESTful Web Service. The DSS runtime infrastructure will provide:
  - Framework and service to both Patient Application and Medical Expert.
  - Forecasting models and analytics based on the risk model fulfilled by data in Repositories.
  - Run-time decision based on the status of incoming data.
- Personalised educational data that used for patients' self-management: This data is recomended by the decision support system and is linked with the educational data from the public RDF. The educational data is stored in the public RDF. In the private RDF, we can store the personalised educational data that is in practise the links to the public RDF.

2.2. Query Performance Enhancement Requirements

This task anticipate the possible problems that may arise later when there are many users that will be involved in using, contributing and exploring the risk associations and the prediction of the disease progressions. Traditional approaches for posing queries against Linked Data retrieve and cache data in local indexes; however, the dynamicity and scope of Web data implies that results are often stale or missing.

We search for solutions from the two perspectives:

- RDF data repository query
  - Caching methods
  - Localisation method
  - Optimization of the query statements
  - Implementation related tool optimizations
- Web application query

2.2.1. RDF query optimization

CARRE data will be stored in RDF. The user application, such as explore the risk factors from the public repository, or explore their own disease progression, users has to retrieve data from the portal, and the portal has to fetch data from the RDF. In particular, a typical application is to show the risk associations and progressions in an interactive way to view graphs. The graphs are used to visualise data, and more importantly, the interrelation of the data. This makes our performance enhancement not only face problems with those involved in web data retrieve, but also with the problems if slowness of the graphic processing.

2.2.2. Why RDF is special?

Compared to querying data stored in a fixed relational database schema, querying a triple store is still usually slower by a factor of 2-20 (cf. e.g. BSBM results[5]). This shortcoming is due to the fact that columns in a

[5] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/

relational database are typed and may be indexed more efficiently. By using a triple store, this efficiency is lost to the flexibility of amending and reorganizing schema structures easily and quickly[6].

Link-traversal query approaches for Linked Data have the benefit of up-to-date results and decentralised execution, but operate only on explicit data from dereferenced documents. However, on the other hand, in many typical application scenarios, only relatively small parts of a knowledge base change within a short period of time in the triple stores. The majority of triples remain unchanged. Hence, most queries will return the same results even after the occurrence of changes on the knowledge base. In this case, caching is a suitable choice.

In the heavy traffic condition, and with the resources constrains on the Internet access, optimizing web application is demand[7]. A web application comprises of various modules and individual components with specialised function, which process a piece of information according to the code and provide output to other components.

Optimization of a web application improves the usability of for the user and enhances their experience, Optimization provides following advantages:

- Enhancing the speed of loading a web page
- Balancing the amount of data transfer to an optimal level
- Reducing the unnecessary processing by optimise the load on web servers

3. RDF query performance enhancement approaches

3.1. Caching methods

To obtain comparable high query performance with relational databases, diverse database technologies have to be adapted to confront the complexity posed by both RDF data and SPARQL queries. Database caching is one of such technologies that improves the performance of database with reasonable space expense based on the spatial/temporal/semantic locality principle.

However, existing caching schemes exploited in RDF stores are found to be dysfunctional for complex query semantics. Although semantic caching approaches work effectively in this case, little work has been done in this area. To improve SPARQL query performance with semantic caching approaches, approaches such as SPARQL algebraic expression tree (AET) based caching and entity caching can be used. The main idea is that successive queries with multiple identical sub-queries and star-shaped joins can be efficiently evaluated with these two approaches. The approaches are implemented on a two-level-storage structure. The main memory stores the most frequently accessed cache items, and items swapped out are stored on the disk for future possible reuse. Evaluation results on three mainstream RDF benchmarks illustrate the effectiveness and efficiency of our approaches.

Considering the above, it is important for applications that require extremely high performance to ensure that data is compact and related data is located contiguously where possible. This way, the utilisation of the cached data can be maximised. DBMSs have historically performed poorly at this task[8].

In addition, queries are often frequently issued, for example, when different users access the same information in a Semantic Web application. We can take advantage of this fact by caching query results, but also want to ensure that cached query results are selectively invalidated on knowledge base updates.

Caching of data also poses an extra issue in scenarios relating to healthcare, such as CARRE. Applications authorised to access the CARRE private repository must take care not to cache confidential data in insecure

[6] Martin, M., Unbehauen, J., & Auer, S. (2010). Improving the performance of semantic web applications with SPARQL query caching. In The Semantic Web: Research and Applications (pp. 304-318). Springer Berlin Heidelberg.

[7] http://www.examiner.com/article/web-application-optimization-techniques

[8] A. Ailamaki, D. J. DeWitt, M.D. Hill, D.A. Wood, 'DBMSs On A Modern Processor: Where Does Time Go?', Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

locations. In particular, this means that any use of client-side caching must be restricted only to data from the public repository.

3.2. Localisation methods

It is highly desirable to have a way of generating SPARQL queries that make use of data residing in disparate data sources served by multiple SPARQL endpoints. There have been presented some solutions to the problem of querying the data sources at multiple SPARQL endpoints in an integrated manner, namely, DARQ[9] and SemWIQ[10]. These approaches require either supplying of statistics about SPARQL endpoints or registration of SPARQL endpoints in a catalog, respectively. Both of them then use statistics about SPARQL endpoints which are supplied or in case of SemWIQ generated dynamically in order to determine where to send sub-queries. DARQ and even SemWIQ impose restrictions on the expressivity of the SPARQL constructs used. Only a few RDF data management systems, such as AllegroGraph[11], Stardog[12] or Virtuoso, use Reformulation-based query answering.

Automatic query satisfaction is not a trivial task. However, while a programmer may intuitively knows the most efficient manner in which to process a query. To achieve this, it requires significant insight and expertise. An automatic query optimiser can evaluate many different plans before settling on one with a low cost, and can do so without the input of a knowledgeable human. There are four steps to query optimisation[10]:

- Cast the query into internal form
- Convert to canonical form
- Choose candidate low-level procedures
- Generate query plans and choose the cheapest

3.3. Optimizing reformulation-based query answering in RDF

Reformulation-based query[13] answering is a query processing technique aiming at answering queries under constraints. It consists of reformulating the query based on the constraints, so that evaluating the reformulated query directly against the data (i.e., without considering any more the constraints) produces the correct answer set. We consider optimizing reformulation-based query answering in the setting of ontology-based data access, where SPARQL conjunctive queries are posed against RDF facts on which constraints expressed by an RDF Schema hold.

Joining can be an expensive operation, involving as it does two different tables[10]. There are a variety of algorithms, depending on the state of the data as regards sorting. This ranges from the very basic brute force algorithm, with a scaling of $O(n^2)$ with the size of the data being examined, to more useful techniques, such as merge, sort/merge, and hash joins. These are described below[14]:

- Nested Loop: It takes a pair of join inputs, such as tables, or outputs from another operator, and designates one the outer, and one the inner input. The inner input is then scanned for matches once for each item in the outer. This approach guarantees that all matches are found, and it work well for small datasets. However, in the case of CARRE project, the RDF stores will potentially be working with extremely large tables, this simple approach cannot be considered advisable. A more commonly utilised solution is the index nested loop join.

---

9 http://darq.sourceforge.net/

10 http://semwiq.faw.uni-linz.ac.at/

11 http://franz.com/agraph/allegrograph/

12 http://stardog.com/

13 Damian Bursztyn, Fran¸cois Goasdou´e, Ioana Manolescu, 'Reformulation-based query answering in RDF: alternatives and performance', HAL Id: hal-01174298 https://hal.inria.fr/hal-01174298

14 http://eprints.soton.ac.uk/267917/1/MiniThesis.pdf

- Merge and Sort/Merge: this types of joins assume that both inputs are sorted in order on the columns that are being joined on. Therefore, a simple scan of both inputs can perform a join in sequent, if the join is one to many. Merge join is always the fastest join if data is sorted correctly.
- Hash: A hash join performs a single scan over each input. It creates a hash table on the first input, with a pointer to the corresponding tuple on disk. When the second input is being scanned, it compares against that hash table to produce the joined output. This technique scales in linear fashion with the amount of data scanned, and does not require inputs to be sorted to work efficiently. However, it is likely to be slower than merge join.
- Join Minimisation: in essence the process of pre-calculating joins does not have to be performed at run time. Join pre-calculation is generally very attractive for read optimised disk-based systems. If a given join is performed regularly, a great deal of time can be saved by storing the completed join on disk.
- Implementation related tool optimizations

3.4. Implementation tool related optimization

In CARRE, Virtuoso is used for the RDF repository. However, as for every tool, there are particular issues and optimisations which apply to Virtuoso, that we need to take into account when generate queries to avoid decrease of the performance. For example:

- When repository sizes are large (do not fit in physical memory), or queries are complex or unoptimised, the server can suffer on a large number of page views.
- Query optimization – for example, converting a scalar subquery into a derived table with a GROUP BY (decorrelation) can improve Virtuoso performance[15].
- Clustering – Virtuoso configured on clustered systems has been shown to achieve excellent performance on very large scale RDF datasets[16].

For CARRE in particular, the clustering abilities of Virtuoso show that if the system should ever need to be scaled to a significantly larger userbase, performance could be maintained easily, since each user's *private* data is not only independent of every other user's data, but deliberately isolated from them. This architecture therefore supports a clear partitioning of data across cluster nodes.

4. Web application query performance enhancement mechanisms

Fetching something over the network could be both slow and expensive: large responses require many roundtrips between the client and server, which delays when they are available and can be processed by the browser, and also incurs data costs for the visitor. As a result, the ability to cache and reuse previously fetched resources is a critical aspect of optimizing for performance. Using resource caching[17] will enhance API performance, reduce the overhead on the server, and minimize the response size. Mechanisms will be looked from different aspects: server side and client side. Overview

Web application caching[18] aims to store locally generated data that possible to be reused in the future. The data are mostly stored closer to the end user. Caching can be implemented at different levels to improve web application performance. In general, caching help users to retrieve contents faster.

---

15 http://www.openlinksw.com/weblog/oerling/?id=1800

16 http://www.w3.org/wiki/LargeTripleStores#OpenLink_Virtuoso_v6.1_-_15.4B.2B_explicit.3B_uncounted_virtual.2Finferred

17 https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching?hl=en

18 http://docforge.com/wiki/Web_application/Caching

Benefits of web application caching[18]:

- Even small performance improvements from caching can drastically improve the user experience as the information they requested are not retrieved from the distant server, instead at a local server.
- It reduces workload of the service provider's web server, which reduces hardware and support costs.
- Caching could be done with very low-cost. Most web browsers and some Internet networking components, such as proxies, have common caching options. Leveraging these caches only requires a simple entry in the HTTP response header.
- The HTML5 standard incorporates web application caching through manifest files, which enable developers to implement application-specific caching techniques easier.

Drawbacks of web application caching[18]:

- It is sometimes requires extra logic with additional testing and debugging time to implement caching, therefore increase cost. .
- The best results of caching often emerged fromfrom purposed built -software or hardware. Thesecan be costly and harder to support. For example, with distributed systems a dedicated proxy cache sitting above the web application might be most practical.
- System administrators might need to be specially trained or experienced in particular caching scenarios to properly configure and support them.
- For client-side caching, privacy concerns may prevent the data being cached to a an insecure local device.

A wide variety of design patterns exist for caching within web applications. Each design pattern has its own set of benefits and limitations. The patterns can also be used in combination with each other. For those web pages that rarely change, they do not required to be retrieved on every new request. It is advisable to consider caching in the early stage in the request handling process.

4.1. Server-side caching

4.1.1. Caching proxy servers

In the simplest form of a web application, the system responds directly to every request from a web browser. It is beneficial to introduce a dedicated caching server, a proxy server, between the client and the web application server to avoid dynamic processing completely whereever possible. A proxy server can check the HTTP headers for cache hints and act accordingly. This frees the web server from all requests, but only to process the requests it actually needs to.

A proxy can often be introduced without additional changes to the web application if the web application had already responded to the requests with appropriate cache headers. Some proxies also add their own HTTP header content for tracking purposes, or to pass back information to the web application. Some proxy may allow dynamically generate requests where required, such as for user accounts and expired data.

In addition to caching, proxy servers may perform additional performance enhancement tasks, such as compressing data and optimizing documents. Systems that assist in such way are referred to as web accelerators. For example, this intermediary may multiple JavaScript files, minify and combine the files, and change the HTML of the main document to reference the new smaller JS file. This allows the web developers to focus on building page functionality while another system handles performance characteristics.

Sometimes caching proxy servers are most useful in distributed environments, where the proxy server can act as both a load balancer and a cache. For every request requiring dynamic content the request is passed to the next available web server.

4.1.2. Content delivery networks (CDN)

Content Delivery Networks refer to the geographically distributed networks, which can be very useful for storing static content close to the end user. For a service provider with multi-national presences, the so called Edge servers can reside around the world, closer to the end users; while the primary storage resides in several central locations. A web application can push any static content via FTP to the central server. Upon first request in a local region, the edge server requests the file from the central server. Subsequent requests near the edge server return the cached content until the Time-To-Live (TTL), expires. For example, if the TTL is set to 30 minutes, then the next request following the 30 minute mark is refreshed from the central server.

A Content Delivery Network might serve an entire web site, having the domain name point directly to the CDN. The content can be pushed to the CDN's upload server whenever the web application is updated. Alternatively, a web application might push just large static content, such as videos or images, to the CDN, while handling the primary web page requests dynamically.

4.1.3. Caching contents

In term of the contents, caching can be done for the whole or part of the contents on each page for web applications. Typically, there are two approaches: caching for the whole pages or caching for partial pages.

4.1.3.1. Whole page content

When a web server/web application accepts a request, that server might employ its own form of page caching. In general, the main problem to solve in caching whole page content is to decide when the content is out-of-date and needs refreshing, particularly in large systems where a single page is made of many smaller components. To completely avoid dynamically generating a requested web page more than once, the entire page may be cached to disk and served directly by the web server. When the page is first created, or updated later, the disk cache is written in a path accessible to the web server and subsequently being handled without any dynamic code.

This method has several important consequences[19]:

- All URLs from other pages must point directly to this file, or a path easily being processed by rewriting rules. Therefore, it is best to use a predetermined naming convention for all cached files so it's relatively trivial for other pages to generate the correct path.
- Page cache updates must be explicitly triggered when necessary. If missing page requests are all sent to the web application, then the first request might be one trigger. All future related updates by a content manager must also be applied.
- No content on the page can be generated dynamically during the page request. However, this can be overcome by AJAX. The client makes requests after loading a page, in order to update any additional dynamic content. However, this may also overcome the benefits of caching the entire page.
- Clean URLs might be lost without the use of rewriting. The URL structure should be considered in the context of the entire web application.

Another method of caching complete page content is for the web application to handle it directly on demand. All page requests still go directly to the web application, it then checks for a cached page. If the cached page exists and is not out of date, the page is sent immediately back to the client and no further processing is taken. The cache might be updated periodically or when the data becomes out of date.

This method also has several important consequences[19]:

- URLs do not need to be changed for cached or dynamic content. It is easier for bother users and the web application.
- The cached data may be stored at anywhere that accessible by the web application, such as remote flat file storage or databases.
- The web application may need to query several components to determine if the page as a whole is out of date. Alternatively, the other components may notify a core caching component when it is time

[19] http://docforge.com/wiki/Web_application/Caching#Whole_Page_Content

to refresh the page. For example, an administrative interface may clear the cache or notify a caching interface when a database record is inserted or updated.

4.1.3.2. Partial page content

It is often need and necessary to cache fragment of dynamically generated contents on a web page when possible. If a page is built from various components, and only some of those components are 'costly' to generate, then we can only cache those parts in order to improve overall performance.

If a web application is properly modularized during development, which make it possible that any individual component are cacheable. In the case of a simplified shopping cart, the products and categories might be easy to generate, while the user's cart summary on each page can be more expensive due to user management and cart calculations. On each page but the cart itself, the summary might be cached and updated upon needed. In CARRE, complex visualisations of the risk factor graph can be cached locally for performance, since all the risk data is public, but components of the same page referring to a user's private data should be generated dynamically, in order to avoid storing private data on an insecure device (such as a mobile phone).

4.2. Client-side caching

4.2.1. Web application level

Following are the possible caching approach that implemented in the client side.

- Automatically paging the query according to screen size, do not bring back all data at once.
- Paging through a query result is the process of returning the results of a query in smaller subsets of data, or pages. This is a common practice for displaying results to a user in small, easy-to-manage chunks.
- To use the Fill method to return a page of data, specify a startRecord parameter, for the first record in the page of data, and a maxRecords parameter, for the number of records in the page of data.
- Use local storage to cache common query results, and session storage to cache versatile query results.
- The main problem with HTTP as the main transport layer of the Web is that it is stateless. This means that when you use an application and then close it, its state will be reset the next time you open it. If you close an application on your desktop and re-open it, its most recent state is restored.

The state of the interface should be stored. Normally, this is done on the server-side, and you would check the user name to know which state to revert to. This is where local storage is needed, where a key on the user's computer needs to be read when the user returns.

If a Web service allows users with only a certain number of calls per hour but the data does not change all that often, the information should be stored in local storage and thus keep users from using up the quota. A photo badge, for example, could pull new images every six hours, rather than every minute.

This is very common when using Web services server-side. Local caching keeps user from being banned from services, and it also means that when a call to the API fails for some reason, there are still information to be displayed.

- Use client-side database instead of server roundtrips (e.g. Web SQL Database, IndexedDB).
- Use web workers for CPU-heavy operations in manipulate queries and results.
- Use web sockets to save bandwidth compared to XHR (AJAX), for example, in auto-complete queries.

The web has been largely built around the so-called request/response paradigm of HTTP. A client loads up a web page and then nothing happens until the user clicks onto the next page. All HTTP communication was steered by the client, which required user interaction or periodic polling to load new data from the server. However, all of these work-arounds share one problem: They carry the overhead of HTTP, which doesn't make

them well suited for low latency applications. Think multiplayer first person shooter games in the browser or any other online game with a real time component.

The WebSocket specification defines an API establishing "socket" connections between a web browser and a server. In plain words: There is a persistent connection between the client and the server and both parties can start sending data at any time.

- Use cache manifest for site to cache static resources from query results.

The main differences between the HTML5 cache manifest vs. the traditional HTTP headers[20]:

- for the cache manifest you need support in the browser
- for the HTTP headers you also need support in the browser of course but it's more universal
- you have more control over the caching with cache manifest
- your website or Web app can work correctly offline with no connection at all
- you can have two version of every resource - for offline and online usage

4.2.2. Web browser level

All the modern browsers have an aggressive cache policy with the purpose of reducing the network traffic and increase the performances. The last point is really important: to execute operations faster browsers read a set of information from the response header (for example, Etag, Cache-Control and Date), and, based on their values, it decides to take the data from the response or from the cache.

Default cache policy must satisfies a request for a resource either by using the cached copy of the resource or by sending a request for the resource to the server. The action taken is determined by the current cache policy and the age of the content in the cache. This is the cache level that should be used by most applications.

A cache policy defines rules that are used to determine whether a request can be satisfied using a cached copy of the requested resource. Applications specify client cache requirements for freshness, but the effective cache policy is determined by the client cache requirements, the server's content expiration requirements, and the server's revalidation requirements. The interaction of client cache policy and server requirements always results in the most conservative cache policy, to help ensure that the freshest content is returned to the client application.

Cache policies are either location-based or time-based. A location-based cache policy defines the freshness of cached entries based on where the requested resource can be taken from. A time-based cache policy defines the freshness of cached entries using the time the resource was retrieved, headers returned with the resource, and the current time. Most applications can use the default time-based cache policy, which implements the caching policy specified in RFC 2616[21].

The cache policy is used to determine if and how long to keep the cached the contents. Here are some widely used cache policies[22]:

- Always discard the information that will not be needed for the longest time.
- Least Recently Used (LRU) discard the least recently used items first, need to keep tracking what had been used.
- Most Recently Used (MRU) discards the most recently used items first. The idea is the older an item is, the more likely it is to be accessed.
- Least-Frequently used (LFU), counts how often an item is needed, those that are used least often are discard first.
- Pyramidal Selection Scheme (PSS) was designed by the authors of SLRU to reap the benefits of the latter while avoiding its high CPU overhead.

---

20 http://www.html5rocks.com/en/tutorials/speed/quick/

21 http://www.ietf.org

22 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.2906&rep=rep1&type=pdf

4.3. Useful tools

Here, we list two tools that will be used in the CARRE implementation.

4.3.1. Redis[23]

Redis is an open source (BSD licensed), in-memory data structure store, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets,sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

Redis is not a plain key-value store, actually it is a data structures server, supporting different kind of values. What this means is that, while in traditional key-value stores you associated string keys to string values, in Redis the value is not limited to a simple string, but can also hold more complex data structures.

Redis keys are binary safe, this means that you can use any binary sequence as a key, from a string like "foo" to the content of a JPEG file. The empty string is also a valid key. The Redis String type is the simplest type of value you can associate with a Redis key. It is the only data type in Memcached, so it is also very natural for newcomers to use it in Redis.

Before continuing with more complex data structures, we need to discuss another feature which works regardless of the value type, and is called Redis expires. Basically you can set a timeout for a key, which is a limited time to live. When the time to live elapses, the key is automatically destroyed, exactly as if the user called the DEL command with the key.

4.3.2. Memcached[24]

Memcached is an open source distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering. Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches. Its API is available for most popular languages.

Memcached is a distributed memory object caching system, generic in nature, but originally intended for use in speeding up dynamic web applications by alleviating database load. Memcached allows user to take memory from parts of your system where you have more than you need and make it accessible to areas where you have less than you need. Memcached also allows user to make better use of the memory.

5. Implementation

5.1. Workflow

As shown in Figure 1, visual analytic retrieves data from RDF and presents them a user-friendly way to help users understand the informed decision.

[23] http://redis.io/

[24] http://memcached.org/about

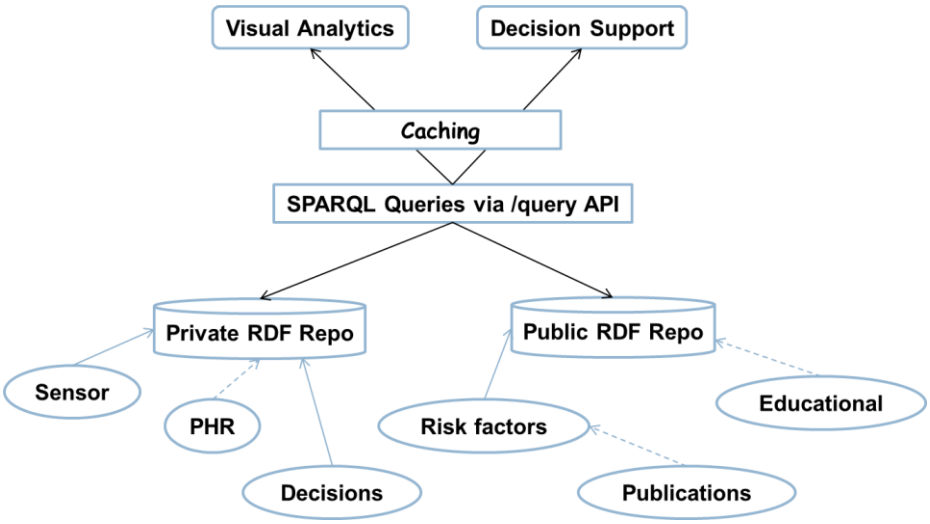

Figure 1. Query workflow

5.2. Client-side implementation

- Investigate and evaluate server side query parsing and optimisation (e.g. paging, filtering, ordering)
- Utilise suitable manner which communicate with server to make query more efficient (AJAX, WebSockets)
    - Partial diagram
    - Range does not changes very often
    - Relevant risks and values
- Build test suite of VM clusters and automated performance benchmark

5.3. Server side implementation

The following steps have been taken to optimise (or support future optimisation of) the repository implementation:

- The server specification (memory and storage) have been chosen to support best performance from the Virtuoso RDF store for current use, and predicted use through the life of the project, with scalability beyond the project designed as an option from the start.
- The choice of Virtuoso as the repository backend was in part based on its scalability to a distributed environment, if necessary. The logical design of the repository in terms of separate private RDF graphs per user also promotes scalability. In this respect, the privacy features of the repository also make it easier to improve performance.
- Common queries for the CARRE use cases are wrapped as methods in the CARRE Web API; this allows such queries to be optimised for performance without requiring any effort or duplication of work in application development. As visualisation and decision support services development continues, the relevant queries are optimised as much as possible by hand.
- Where necessary, for the same (common) queries, each RDF graph is saturated with the terms relevant to satisfying them. That is to say, inferences are precomputed to ensure that query results can most often be retrieved by database lookup rather than inference.
- Currently, where common queries relate to external Linked Open Data (such as, for example, looking up details of the units for a particular measurement), performance is good with queries against the

Bioportal[25] repository. There remains the option of installing a local copy of Bioportal and querying against that instead, allowing content to be synced in the background independently of user queries.

- The CARRE RESTful API is implemented using the Django framework[26], which includes support for Memcached. Thus, the results will be cached securely on the server and made available for repeated (authenticated) queries. This can improve performance on CARRE data, as well as minimising the need for any remote queries for long-lived external data, such as, e.g., units.

In addition to technical measures to improve query performance, it should also be noted that in practice, none of the most common visualisation or decision support queries for CARRE involve significant quantities of data or on-the-fly inference. The entire public risk factor association dataset can be downloaded in its entirety and cached locally without putting many demands on the client. This data also changes comparatively slowly due to the clinical scientific and review processes.

With regards to the personal data, the volume of data gathered and stored are anticipated to be large. The main use of visualisation is to calculate means or trends – an individual's step count values are less significant alone for risk calculation than the overall categorisation of a patient as, for example, "highly active" – categorisation that can be calculated offline on the server rather than on-the-fly. Where individual data must be retrieved for visualisation of, e.g., a patient's activity history, the amount to be retrieved is always bounded by a relevant time period (previous week, previous month, and so on) and so again, only ever involves limited quantities of data.

5.4. Test and validation

Primarily, Selenium is for automating web applications for testing purposes, but is certainly not limited to just that. Web-based administration tasks can also be automated as well.Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.

There are three projects of Selenium:

- Selenium WebDriver: To create robust, browser-based regression automation suites and test, and scale and distribut scripts across many environment.
- Selenium IDE: Creat quick bug reproduction scripts and create scripts to aid in automation-aided exploration testing.
- Selenium Grid: Enable running tasks on many servers at the same time, allows testing on multiple browsers and operation systems in parallel.

The tests can be setup in the above flow utilise Selenium Grid, which aims to test the performance under various user side configurations.

25 http://bioportal.bioontology.org

26 https://www.djangoproject.com

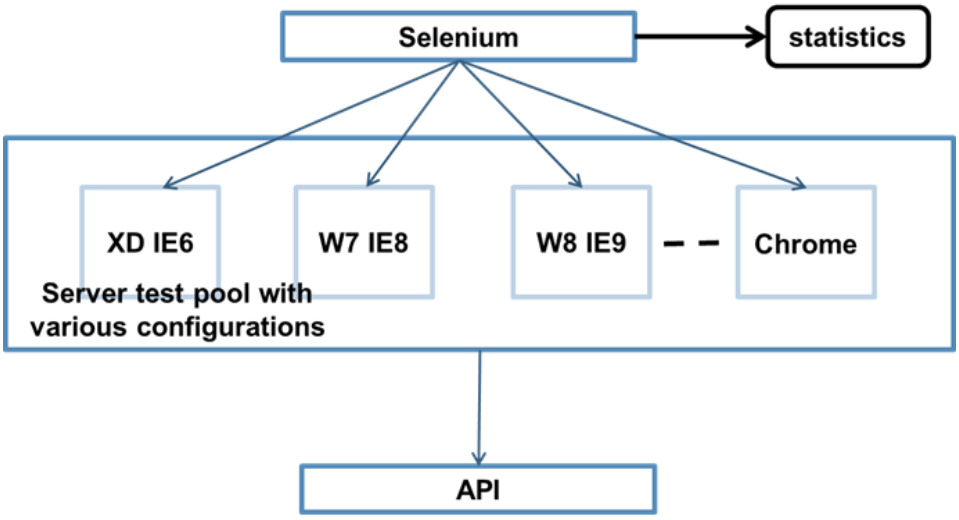

Figure 2. Testing workflow

5.4.1. Prelimary test

Currently, we have setup the visual analytic interface with functions that described in D5.1. In the next few months, along with implementation of D5.3, there will be more uses cases, for those performance could be improved if we adopte the approaches that are described in this document. We conducted a prelimary test on retrieving data from the RDF repository. In the firtst test was carried out when the network connect is set as Digital subscriber line (DSL) with the speed of 2MB/s. The broadband is set at the the relatively low level, with the aim to show clearer the difference of the performance. It worth note that, in the current data repository at the stage of the system development, CARRE repository had a low usage and with small scale of dataset, therefore, we purposely scaled down the broadband access to a low level. As shown in Figure 3, we cache the query data in the local storage.

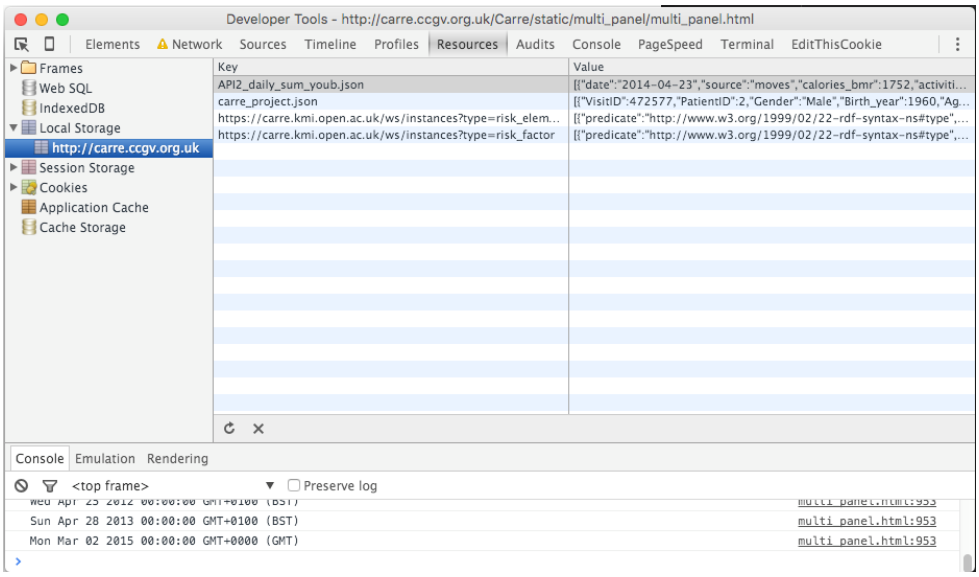

Figure 3. Screenshot of data that cached in the local storage

The following is code piece, which utilizes local-storage to cache the RDF query results by intercept jQuery AJAX call.

```
if (window.jQuery && window.location.hash == '#opt') {
  jQueryAjax = jQuery.ajax;
  // only the settings pattern is used
  jQuery.ajax = function (settings) {
    // if (settings.url && settings.url.indexOf('.json') >= 0
    if (localStorage.getItem(settings.url)) {
      var data = JSON.parse(localStorage.getItem(settings.url));

      if (settings.asyc === false && settings.success) {
        // if async is disabled
        settings.success(data);
      } else {
        var deferred = new $.Deferred();
        deferred.resolve([data]);
        // return promise so that outside code cannot reject/resolve the deferred
        return deferred.promise();
      }
    } else {
      // call the jQuery version
      var promise = jQueryAjax(settings);

      promise.done(function(data) {
        localStorage.setItem(settings.url, JSON.stringify(data));
      });

      return promise;
    }
  }
```

5.4.2. Preminary test 1 – loading time

Figure 4 shows the data retrieving without optimization and Figure 5 shows the data retrieving with the optimization. As can be seen, the page load time reduced from 3.73s to 2.02s.

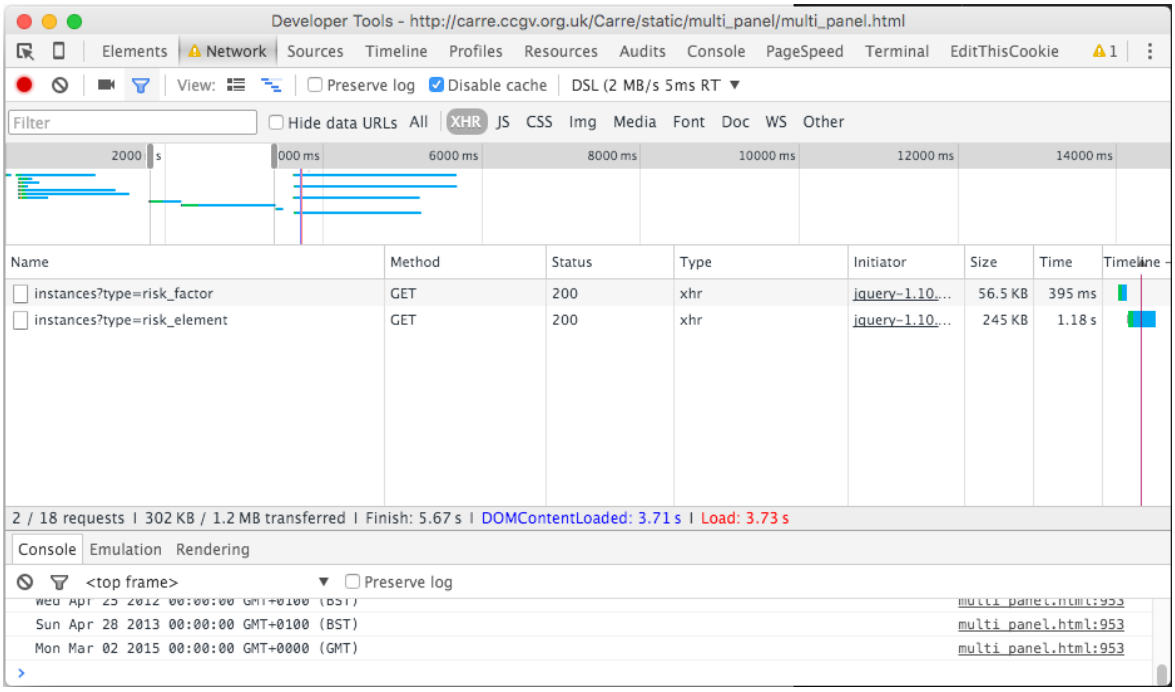

Figure 4. Loading time before optimization

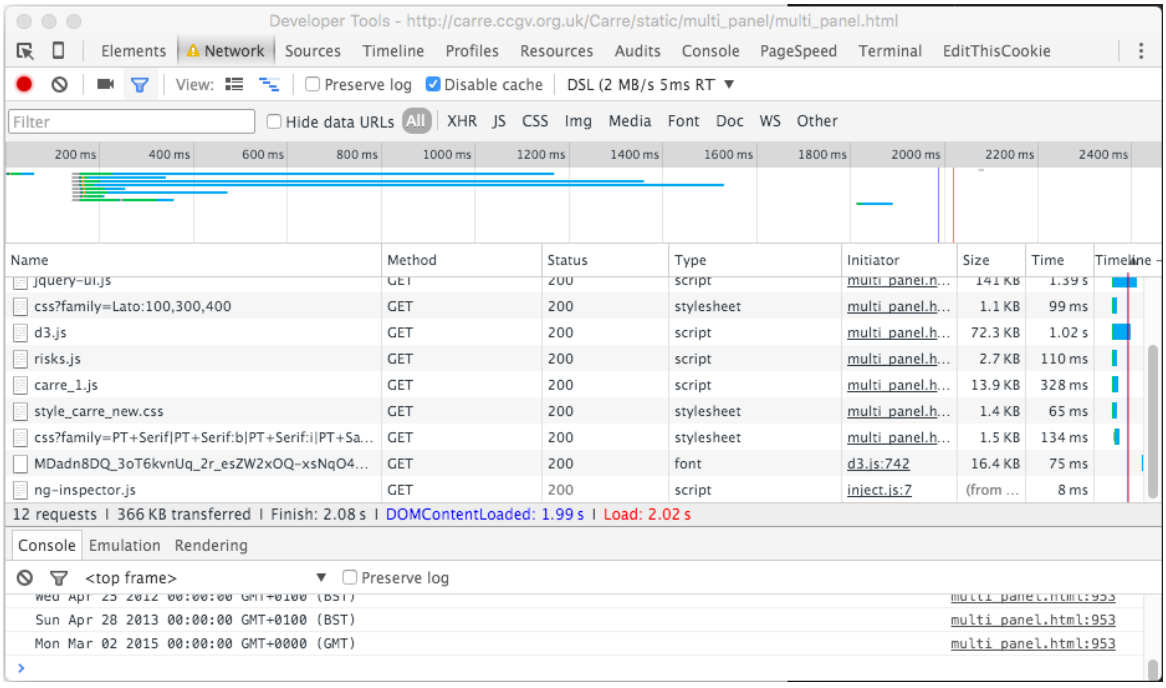

Figure 5. Loading time after optimization

5.4.3. Preminary test 2 – CPU usage

The second test is designed to test the CPU usages. Figure 6 and 7 shows CPU usage before and after optimization. As can be seen, CPU usage is significantly reduced after the optimisation.

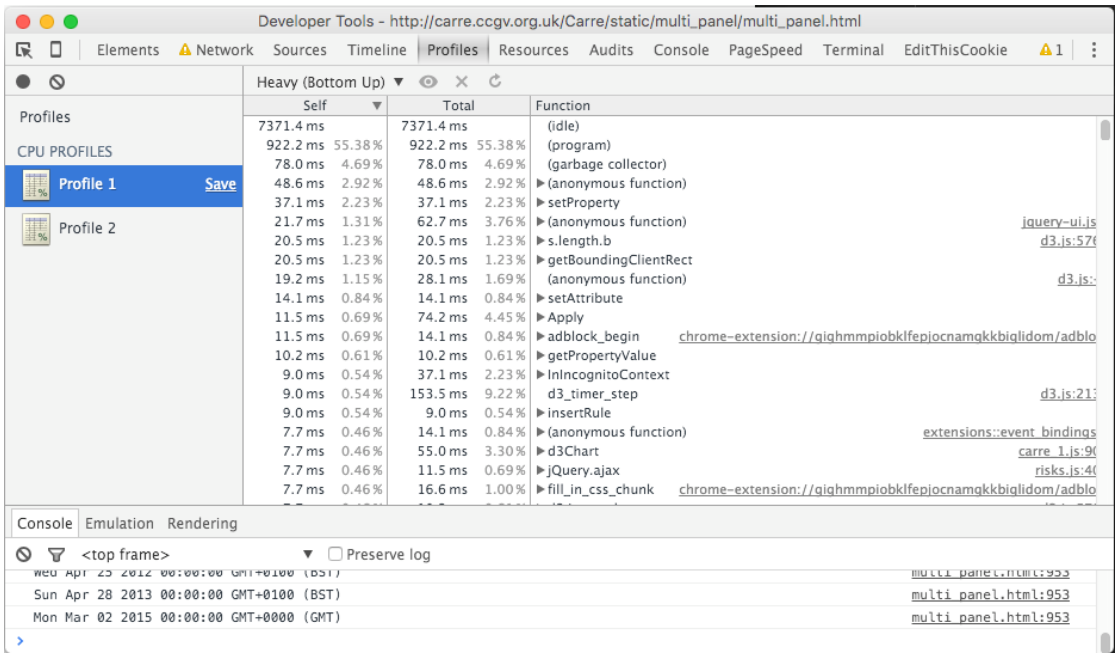

Figure 6. CPU usage before optimization

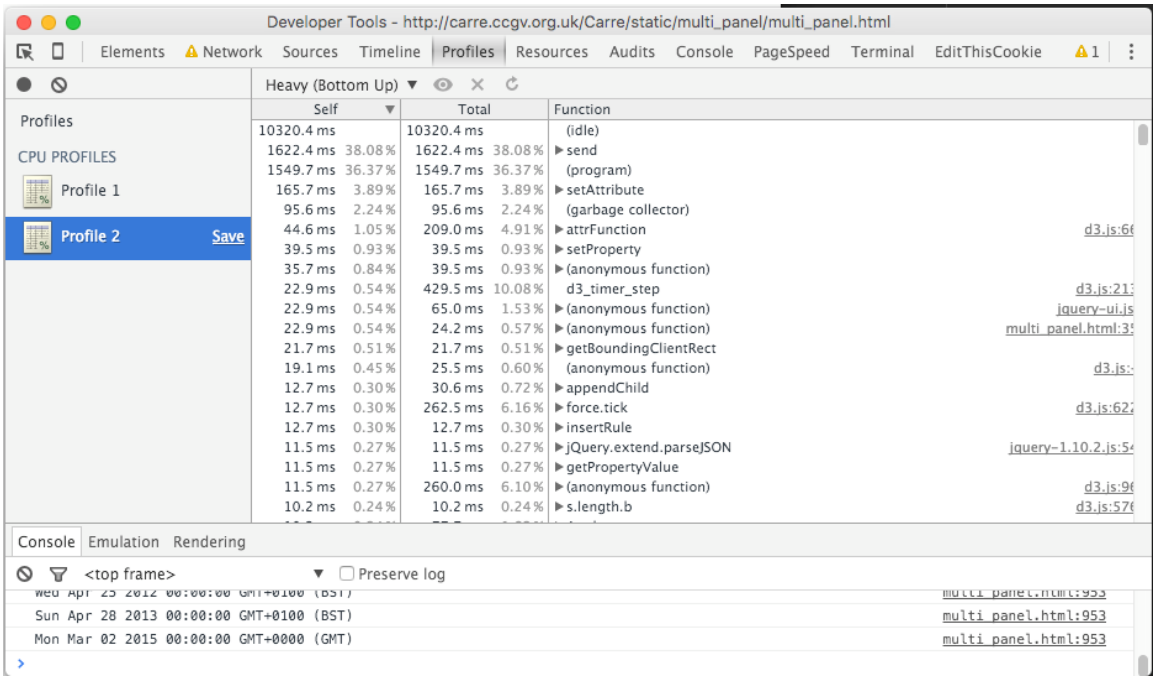

Figure 7. Applications for profile 2

6. Conclusion

This task is to anticipate the future possible technical challenges, such as when there are large number of user data and large number of user who access the CARRE system. Although, at the current stage and even at the end of the project, the actucal users and usage will not reach a significant volume, we still believe it is necessary to take into account of this factor.

The target of Work Package 5 in CARRE project is to provide effective data management and visual analytics tools to empower patients and medical experts to access, view, understand and analyse patients' health status and possible disease progressions.

In this deliverable report of D5.2, we mainly aim to provide some advice for the future implementation. Some advices for the future implementation will be provided. Initial tests are conducted as a proof-of-concept. The actual effects of the improvement can also be seen after finishing the task D5.3: Advanced visual analytics module. The actual effect of the improvement can also be seen after finishing the task D5.3: Advanced visual analytics module.