On the feasibility of distributed systems for interactive visual analysis of omics data

Yehia M. Mokhtar Farag

June 1st, 2014

Master Thesis
Department of Informatics
University of Bergen

Preface

Omics is a set of high-throughput technologies used in the biological field to discover and understand the roles, relationships and behaviour of biological molecules. The phenomenal rate for high-throughput omics data production requires effective visualization and analysis tools to process the data and make it biologically meaningful. Most existing tools are either standalone applications that require sufficient computing resources for the data processing and analysis (which may not be always available) or non-interactive web tools which limit the data exploration. In addition, some of these tools require advanced programming skills, which can be a barrier for many biologists.

To overcome these limitations this master thesis investigates the feasibility of developing a web distributed system for omics data that supports interactive visualisation of omics data analysis. This includes developing a prototype to address the essential development challenges and to outline the main limitations of such a system.

The work in this study was carried out under the supervision of Dr. Kjell Petersen, Group leader at CBU at the University of Bergen and Dr. Harald Barsnes, Post doctor at the Proteomics Unit at the University of Bergen (PROBE).

Chapter 1 introduces the main topics and concepts related to this study, while Chapter 2 gives an overview of the motivation for the study. The main goals are defined in Chapter 3. Chapters 4, 5 and 6 explain the development stages, starting with the technology selection and the system design, and ending with a discussion of the main development challenges. Chapter 7 and 8 include the main results and an evaluation of the developed system. Finally Chapter 9 summarises the study stages and provides the overall conclusion.

The source code for a prototype application is available at:

http://diva-omics.googlecode.com

Dedication

"This work is dedicated to the memory of my great father, and to my kind beloved mother."

Acknowledgment

First and foremost, I have to express my gratitude towards my project supervisors, Dr. Kjell Petersen, and Dr. Harald Barsnes, for their guidance and advice. I also extend my sincere thanks and appreciation to my family and friends for their support throughout my study and for Dr. Hala Mokhtar, Mr. Tamer Eldeeb and Ms. Vanessa Armendariz for their help and support during my master study.

Contents

List of Figures

List of Tables

Chapter 1

Background

"Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning."

Albert Einstein

1 Background

The role of Bioinformatics in science and industry has become increasingly prominent and is growing in significance every day. The impressive revolution in computing capabilities, together with advancements in molecular biology research have led to the increase of data analysis capabilities and the gathered knowledge. This has had a great impact on the way of thinking about life sciences and biotechnology. This study is concerned with one of the main bioinformatics aims which is developing tools, methods required for biological data analysis. The purpose of this chapter is to provide an introduction to the main concepts and topics making up the background for this thesis.

1.1 Molecular Biology

Biology is the natural science focusing on life and different living organisms. Doing biological studies aim to increase the ability to understand the biological information of different living organisms and to use this gathered knowledge in ways that help people lead better and healthier lives.

Molecular biology is the branch of biology that deals with the biological activity at the molecular level. The central dogma of molecular biology is a regulatory mechanism that explains the dynamic interaction among DNA, RNA and proteins, see Figure 1 [1].

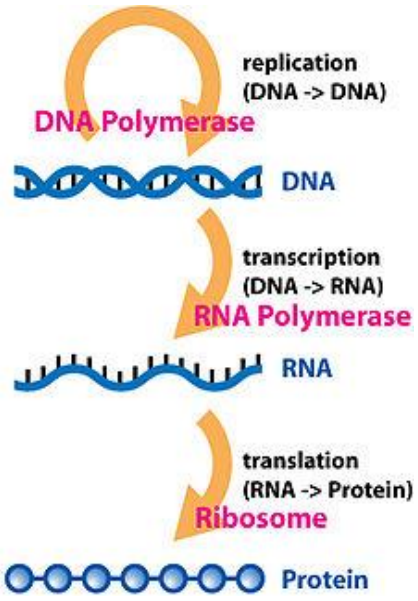

Figure 1: The central dogma of molecular biology [Wikipedia].

The central dogma is illustrated by three major steps. First, the replication step; where two identical replicas from one original DNA molecule are produced, this step is important for transferring the stored genetic information from one cell generation to the next. Second, the transcription step where DNA code is converted into a complementary strand of RNA. And finally, a translation step where the RNA is subsequently translated into the corresponding proteins.

Genes are DNA sub sequences that have the capability to generate specific RNA molecules. They are considered as the fundamental physical and functional unit of heredity [2]. Proteins are biological macromolecules consisting of one or more polypeptide chains (chain of amino acid residues) and performing a large number of functions within living organisms.

The regulatory mechanism for the central dogma three main steps depends mainly on the interactions between the different genes and proteins in the living cells, where specific proteins promote the activity of specific genes [3]. The identification of the regulator-target relationships between proteins and the activity of specific genes requires measuring of proteins abundance (protein quantification) and arranging the sets of differentially expressed genes (normally assessed by measuring the quantity of the produced mRNA [3]) simultaneously. Here comes the role of high-throughput omics techniques.

1.2 Omics Techniques and Datasets

Omics as a term refers to a set of biological fields and high-throughput technologies all ending in -omics; as in genomics, proteomics, transcriptomics and metabolomics. These technologies are used to understand and discover the roles, relationships and behaviour of different molecules and using different applications and databases to handle and store this information. The omics technologies are not only concerned with the relevant molecules, but also try to reveal the roles of given system conditions and provide the most comprehensive views of the studied biological system by measuring as many of the relevant molecules as possible at the same time. The most relevant omics technologies for this thesis are:

- **Genomics:** The study of genomes (the whole set of organism DNA) and gene functions, including efforts to explore entire DNA sequences and fine-scale genetic mapping for organisms [2].

- **Transcriptomics:** The study of the produced RNA transcripts at any one time. Mainly focused on the effect of different biological or environmental factors on the transcript patterns.
- **Proteomics:** The study of proteins, and their functions, including proteins structure analysis.

1.2.1 Omics Data

The high-throughput omics techniques have provided the ability to simultaneously measure a large number of samples or conditions, thus producing massive amounts of data. This data has to be handled and stored efficiently, either in genome-scale omics repositories such as *Ensembl* (http://www.ensembl.org) or in dedicated databases such as *ArrayExpress* (www.ebi.ac.uk/arrayexpress) for experimental gene level data or as *UniProt* (www.uniprot.org) for the protein level. In both cases the omics data have to be readable for analytical resources and tools. The generated omics data is therefore converted into computer readable formats. This provides the facilities for tools to handle the data and generate biologically meaningful results.

1.2.2 Omics Datasets

Through this thesis the omics datasets (the input data) will be represented as a two dimensional table with a given number of columns and rows, from hundreds to thousands. Each column represents a different sample or condition and each row represents a specific molecule, gene or protein, depending on the experiment type. The index of each row and column maps a specific measurement value to specific gene or protein as shown in Figure 2.

ORF	Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6
YHR051W	0.2	0.04	0.19	0.41	0.76	2.18	2.5
YGR145W	0	-0.23	0.25	-0.89	-1.09	-1.69	-2.18
YCR005C	-0.49	-0.14	0.1	-0.38	-0.84	1.06	2.28
YMR318C	0.04	-0.04	-0.12	-0.43	-0.49	-1.47	-2.12
YLR339C	0.32	0.46	0.3	-0.2	-0.67	-1.79	-2.56
YHL021C	-0.45	0.12	0.65	0.99	0.8	2.98	2.51
YLR267W	-0.12	0.12	0.28	-0.1	-0.47	1.66	2.69
YGR148C	0.2	0.11	0.06	-0.29	-0.27	-1.84	-2.64
YHR128W	0.24	0.08	0.23	-0.71	-0.67	-1.89	-2.32
YLR196W	0.25	0.08	0.28	-0.86	-0.92	-1.56	-2.74

Figure 2: Excerpt of a sample omics dataset.

The huge amount of data produced in omics research normally requires different bioinformatics techniques to analyse and generate meaningful results. This brings us to the importance of omics data analysis and visualisation.

1.3 Bioinformatics

Bioinformatics is the use of computer science and information technology into the field of (molecular) biology. Bioinformatics has three main aims [4]:

i. Organise the huge amount of the biological data in accessible ways.
ii. Develop tools, methods and resources to efficiently analyse these data.
iii. Use the tools to generate clear understandable results.

1.3.1 Omics Analysis and Visualisation

One of the main aims for bioinformatics is to analyse the multi-dimensional omics dataset generated in high-throughput omics studies of biological systems. Although there are many approaches and methods that can be used in omics analyses, the approaches can be classified into two main categories: unsupervised and supervised [5]. The first category is an exploratory approach, where no prior assumption is made about the data. It is trying to explore a dataset and identify groups with similar behaviour as in clustering and principle components analysis. In the second category the focus is on extracting important variables for group separation or for finding correlations between data samples (column groups in our case). It assigns some knowledge about the functions and data behaviour before performing the analysis and uses it to define a role that can be used to assign genes/proteins/molecules into groups.

The generated results from the different analysis approaches are often represented as text and numerical formats. This makes knowledge discovery a challenging task without using efficient visualisation methods simplifying the data complexity.

There are different visualisation methods that can be used to extract the important information from the generated results, and represent it in understandable manner. The main difference between these methods relates to the analysis method applied to the raw data. In the following, an overview of the different visualisation methods and the related analysis is provided.

Hierarchical Clustering is an unsupervised method that is used to order, visualise and group omics data according to the expression patterns, where distance measurements showing similarity might be biologically related. Results from hierarchical clustering are normally represented as dendrograms combined with a heat-map.

Tree or Dendrogram View is one of the most effective and powerful graphical representations of the clustering methods [6] based on illustrating the clustering arrangement of the produced clusters (Figure 3a).

Heat-map is one of the most elegant graphical representations of omics data [6]. The dataset measurement values are represented as coloured blocks where the similarity between values is represented using similar colour (Figure 3a).

Principal Components Analysis (PCA) is an unsupervised statistical approach that simplifies the datasets by reducing the data dimensionality [7]. PCA tends to explain the whole dataset using two or sometimes three principal components. The principal components are relatively independent variables resulted from transforming the potentially correlated descriptors, and have major contribution to the whole dataset, i.e., is able to explain most of the information in the dataset [5]. The results of PCA are usually visualised in scatter plots revealing data clusters and outliers [8].

Scatter Plot is one of the visualisation methods that are used to represent genes and proteins after being processed by PCA analysis (Figure 3b).

Rank Product is a supervised simple non-parametric approach used to evaluate differential expression for regulated genes in replicated microarray experiments. Although the rank product statistic approach was originally created for microarrays [9], the approach has become widely accepted as an analysing approach for most omics data types. The rank sorts the genes/proteins/molecules in descending order after performing multiple comparisons between the values based on user input such that the largest positive difference is giving rank 1.

Profile Plot/Line Chart is a graphical representation for expressions in omics data, where the visualisation of expression levels is done across all samples. These types of plots can be very useful to examine the relative pattern as in gene expression data (Figure 3c).

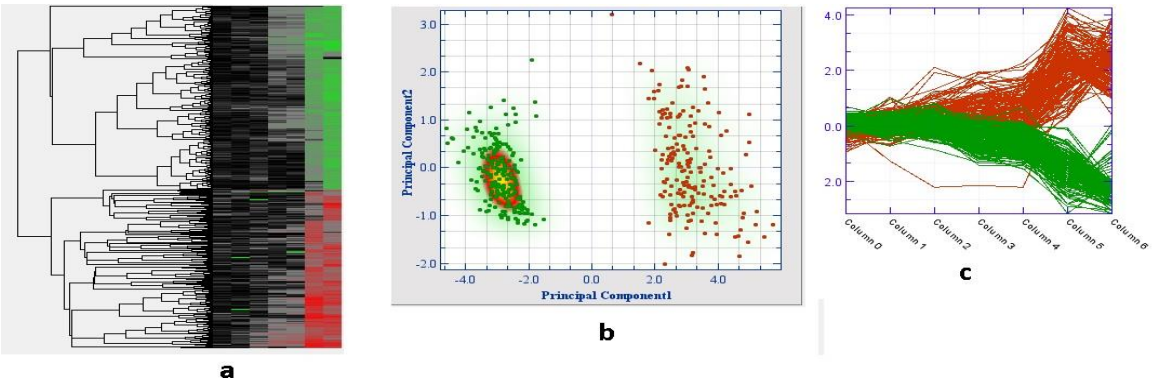

Figure 3: Different visualisation methods for omics data from J-Express application: a) Dendrogram-Tree along the left side of the heat-map visualisation, b) Scatter plot generated from PCA analysis, and c) Profile plot.

1.3.2 Omics Tools

Although there are many web and desktop tools that perform omics analysis, mostof the tools that support interactive visualisation are limited to stand-alone tools or Java applet plugins [10]. While there are limitations and challenges facing standalone tools, the interactive visualisation gives it a major advantage over most web-based solutions. The development of new technologies such as HTML5 and JavaScript provides the opportunity to overcome the interactive visualisation limitations and allow the development of distributed interactive visual analysis systems for omics data that adds all the benefits of using distributed systems.

1.3.3 J-Express Standalone and J-Express Modularized

J-Express is a standalone Java desktop application developed at the University of Bergen for analysing gene expression data form microarray experiments [11]. The system supports different analysis methods as well as multi interactive visualisations which can be applied at either the full dataset or sub-datasets.

J-Express Modularized is an omics data analysis software developed based on J-Express Standalone after refactoring its code into a multi-tier architecture design. This new design separates visualisation, computation into independent layers, thus supporting distributed solutions (Figure 4), in addition to removing Java specific user interface components from the computing layer to increase the code usability in different environments such as web and desktop.

This new system design consists of three main layers: (i) an application logic layer for processing the raw data and generate results objects; (ii) a presentation layer for optimising

the generated results and their visualisations it in the requested way; and (iii) a central manager layer for maintaining the interactive selection between different modules in the presentation layer. In particular, the architecture focuses on the separation between central manager, presentation and the application logic layers. This increases the modularity and isolation between different modules, and also makes the system design easier to understand and thus simpler to maintain, modify and extend, thereby satisfying the requirements of flexibility, extensibility and maintainability.

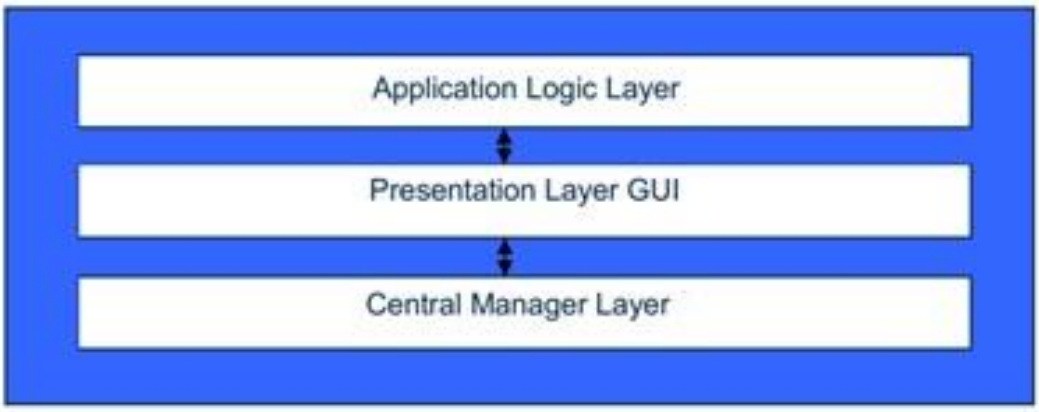

Figure 4: General module architecture design for J-Express Modularized.

J-Express Modularized currently supports the analysis and visualisations of four selected modules from J-Express Standalone: (i) hierarchical clustering; (ii) rank product statistics; (iii) principal components analysis (PCA); and (iv) profile plot/line chart.

Chapter 2

Distributed and Interactive

2 Distributed and Interactive

As a consequence of the large size of omics-scale data generated by high-throughput biological studies, there has been an interest in the development of omics analysis tools that support interactive visualisation. The main challenge is to develop a system that is able to analyse this scale of data efficiently while supporting interactive exploration of the data to provide biological insight.

2.1 Omics Tools Limitations

There are many different omics tools that perform data analysis and support interactive visualisation of the generated results. Such tools are mainly faced by three limitations: (i) being standalone desktop applications; (ii) being distributed-based systems with non-interactive visualisation; and/or (iii) having poor user-friendliness.

2.1.1 Standalone Desktop Application Limits

One of the major limitations for an omics standalone desktop application is that it has to be installed locally on the user's machine which does not always have sufficient computing resources for performing data processing and analysis of large biological data sets.

Additionally, by being a standalone application, the presentation layer and the logic layer are located in the same machine where the data lives. Placing the data files in the local machine will limit data sharing and knowledge transfer between different users (Figure 5).

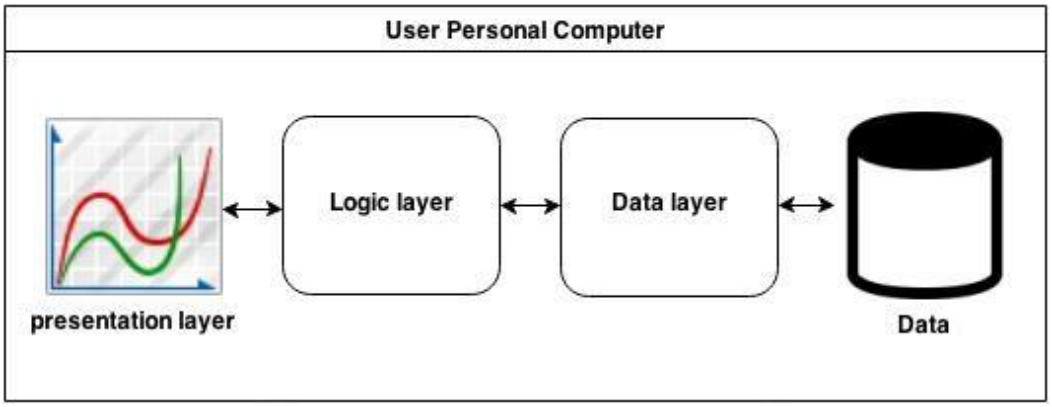

Figure 5: Standalone desktop application architecture – all layers placed on the user's PC.

An additional limitation for standalone desktop applications is the application setup process. The users (biologists) may face setup difficulties if they do not have the technical experience

to set up the application and check the setup requirements. Finally, being a standalone desktop application makes the software maintenance and update more complicated for system developers.

2.1.2 Non-Interactive Distributed Systems

The second challenge faced by omics-data analysis tools is being distributed systems with non-interactive visualisation. These distributed systems perform the data analysis and visualise the results as non-interactive static images. By interactive visualisation of omics data we mean systems that support interactive multiple coordinated views. In omics analysis interactive visualisation is considered a key element for information extraction from multi-dimensional datasets. The multiple coordinated views for the analysis result improve the exploration of the data, and make relations and knowledge discovery much easier tasks [12].

2.1.3 User-friendliness

The third challenge for the omics analysis tools is being easy to use. This includes different programming and computational skills required by the biologists to perform the data analysis. Although, it is still unclear how to develop an easy to use system, considering some specifications can make the system easier to use by biologists.

For an omics analysis tool to be easy to use, it needs to be efficient and provide the users with an easy-to-navigate Graphical User Interface (GUI). The system should avoid requiring programming skills by users, and at the same time support all functionalities in a clear and consistent way. The tools also have to avoid complications that may cause disturbance for the analysis process, and have to deal with the data size, visualising the generated results in a comprehensible way.

2.2 Distributed Interactive Visual Analysis System (DIVA)

In response to these limitations, this thesis addresses the feasibility of developing omics data analysis tools that solves the stated challenges. A proto-type of a *Distributed Interactive Visual Analysis System (DIVA)* will be developed to demonstrate the feasibility of such a system and outline its main limitations.

Browser based web distributed systems will be considered as the way to overcome the limitations of desktop standalone applications. The system will support interactive

visualisation of the results using state of the art web technologies. The main specifications and considerations for the developed system will be highlighted in the following chapters.

2.2.1 A Browser Based Graphical User Interface

A browser based GUI will be developed as the interface for the system. This increases the portability and accessibility, in addition to supporting the sharing of computing resources and providing a low threshold for biologists without programming skills. At the same time it allows the user to focus on the analysis process without dealing with issues such as hardware requirement, system upgrades, scalability, etc.

Using a web distributed solution will reduce the computing capabilities required for the user local machine since the analysis process will be done on the server-side, while the client-side will mainly be responsible for the interactive visualisation of the generated results. A distributed web solution will also enhance the system capabilities by making it possible to store and share the processed results between users which enrich the analysis process.

2.2.2 Interactive Visualisation

The system will support interactive visualisation to support the omics analysis. To avoid the limitation mentioned for non-interactive distributed systems a special layer will be developed to manage the user's selections and coordinate the multiple analysis results visualisations updates.

2.2.3 J-Express Modularized

The system will use the logic layer of J-Express Modularized as integrated components to perform the omics data analysis for the four selected modules (Profile plots, Principal component analysis (PCA), Hierarchical clustering, and Rank Product). Custom modules will be developed to maintain the result visualisation and to handle the interactivity between the different modules. Figure 6 illustrates the DIVA general architecture as a web distributed system.

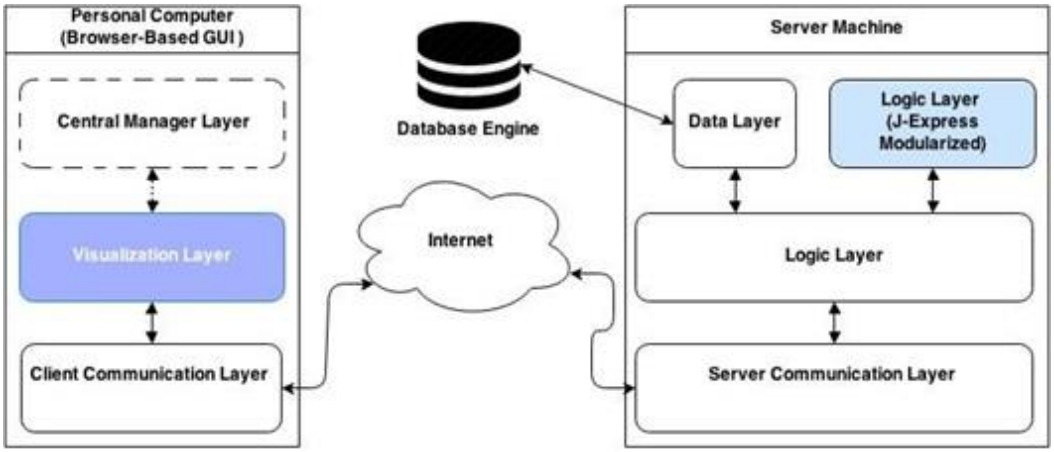

Figure 6: DIVA general architecture.

2.2.4 System Requirements and Limitations

The developed system should be a biologist-friendly (easy to use) system with a browser based interface that handles stored datasets, and fulfills the following requirements.

a. Functional Requirements

- *Interactive Visualisation (Multiple Coordinated Views):* Support omics data (genes/proteins/ molecules) selections and interactively coordinate multiple analysis visualisations updates.
- *Groups and Sub-Datasets Support:* Allow the user to interactively decide which part of the dataset to use (through creating customized colored groups and/or sub-datasets).
- *Support Different Omics Analysis:* Support hierarchical clustering analysis, principal component analysis (PCA) and rank product analysis and interactively visualise the generated results, in addition to support profile-plot visualisation.
- *Store Updated Datasets:* Allow storing updated datasets with new added groups, and sharing it between different users.
- *Export Datasets:* Allow datasets exporting as text files (full or sub datasets) and store it on the client local machine.

b. Non Functional Requirements

- *Performance:* Maintain good/reasonable response speed for user's selections and function invocations.

- *Efficiency:* Efficient memory and CPU usage. Especially on the client-side to provide acceptable performance on (resource-limited) browsers.

- *Usability:* Easy to use for users without any special training.

- *Maintainability/Extensibility:* Flexible design to allow future feature modifications.

c. System Limitations

As the developed system is an initial prototype, the focus will be on the main goal of inspecting the feasibility of developing distributed interactive visual analysis systems. The developed system therefore will have the following limitations:

- A file system (instead of a database) will be used to store and retrieve different datasets and computing results.
- Users are not allowed to upload their own datasets text files (only stored datasets in the file system are available to use).
- The system supports general anonymous users only (no user registration or login required).
- The system supports only tab-delimited datasets text files as input file types for datasets.
- No normalization methods are added to fix the missing measurements in the datasets files.

2.2.5 Challenges Overview

A browser-based Distributed Interactive Visual Analysis System has to analyse large scale omics data while providing interactive visualisation of the generated results on the client-side (which is resource limited). In DIVA we can classify the main system functionalities into two main categories: (a) computations required for different analysis processes (profile plot, PCA, ranking analysis, and hierarchical clustering analysis) and (b) computations required to maintain the result visualisation and coordinate the interactivity between the different visualisation components. The main challenges addressed by DIVA are related to category (b), whereas category (a) functionality of performing the omics analysis and generating results was not the main challenge in DIVA, since J-Express Modularized component was available and has all the required logic to perform the data analysis.

Therefore the main challenge we are concerned with is how to distribute the data and workload across the different layers of the system to achieve both goals. Offloading the entire

workload and data information to the server-side increases the network traffic which is likely to cause a big performance hit. On the other hand, reducing the networking calls will increase the load on the limited client.

The goal is to find the balance between increasing network calls and the workload on the client-side to optimise and maintain efficient interactive visual analysis system for large scale omics data.

Chapter 3

Thesis Goals

3 Thesis Goals

One of the major goals for bioinformatics is to maximise the knowledge which can be extracted from large-scale omics data. Visualisation is considered to be a key element as images and graphs play an important role in outlining and demonstrating the relationships and trends in this type of data.

The purpose of this thesis is to discuss the feasibility of developing a distributed interactive visual analysis omics system demonstrating how selected modules from the standalone J-Express Modularized application can be converted into a web based distributed system maintaining the original application's interactivity and functionality.

A distributed system is considered as the main architectural design, where the client-side will mainly be responsible for managing the interactive visualisation of the generated results, and the server-side is responsible for the remote processing and storage of the data files in addition to sharing the data resources between the users.

A prototype system of an interactive visual data analysis of omics data will be implemented as an online service to non-technical life science researchers. This provides the opportunity to address the main challenges and limitations of such systems, as well as outlining the additive benefits to the omics-data analysis process achieved by the use of distributed systems.

The main goal for this thesis can thus be summarised as:

Investigate the feasibility of developing a web distributed system that supports interactive visualisation of omics data analysis based on converting selected modules from J-Express Modularized, and discuss the main benefits and limitations of such a system.

Chapter 4

Technology

4 Technology

In order to achieve the main goal of the thesis, a prototype of a web distributed system for omics data analysis is developed, providing a realistic evaluation of how feasible it is to develop such a system. This chapter introduces the technologies and methods considered during the development process.

4.1 Technology Selection

DIVA is a web distributed system that supports interactive visualisation for results generated by omics data analysis and J-Express Modularized will be used as the main data analysis component. Thus there are some specifications and requirements to consider when selecting the technologies and development tools:

1. The client-side will be a browser-based client responsible for data visualisation and interactivity between different visualisation modules. The client development technology therefore has to support a browser environment, and be compatible with the data generated by the logic layer at the server-side.
2. The server-side has to support easy integration with J-Express Modularized components.

4.1.1 Client-Side Technologies

There are different client-side technologies that can be used to provide the required rich interactive visualisations and to maintain connectivity with the server. Here is a short description of the technologies considered.

a. Java Applets

A Java applet is a small Java application launched within the browser and then executed using a local Java Virtual Machine (JVM) on the client-side. The applets provide interactive features to web applications which is not always (easily) available for HTML.

b. HTML5

HTML5 is the latest standard of the Hyper Text Mark-up Language (HTML), and supports rich content such as animations, graphics, and music without the need for additional plugins.

This is the language used to develop complex web applications with cross-platform support where the application can run on PCs, Tablets, and Smartphone etc.

c. Ajax

Ajax stands for Asynchronous JavaScript and XML. It is a group of client-side web development techniques allowing the client-side to communicate with the server-side asynchronously (in the background) without interfering with the existing display.

d. Google Web Toolkit (GWT)

GWT is a web development toolkit developed by Google. GWT has a component which allows developers to use plain Java code during the development phase, and then translate the code into JavaScript in the deployment phase (Figure 7). This tool provides developers with the main benefits of using plain Java code in client-side development, e.g., re-using existing code without having to worry about data compatibility. This simplifies the client-side development and provides all the benefits of using JavaScript. In addition, GWT supports different visualisation plugin components. Finally GWT supports Java server-side development and has special protocol (GWT Remote Procedure Calls (RPC)) to organise the client-server communication.

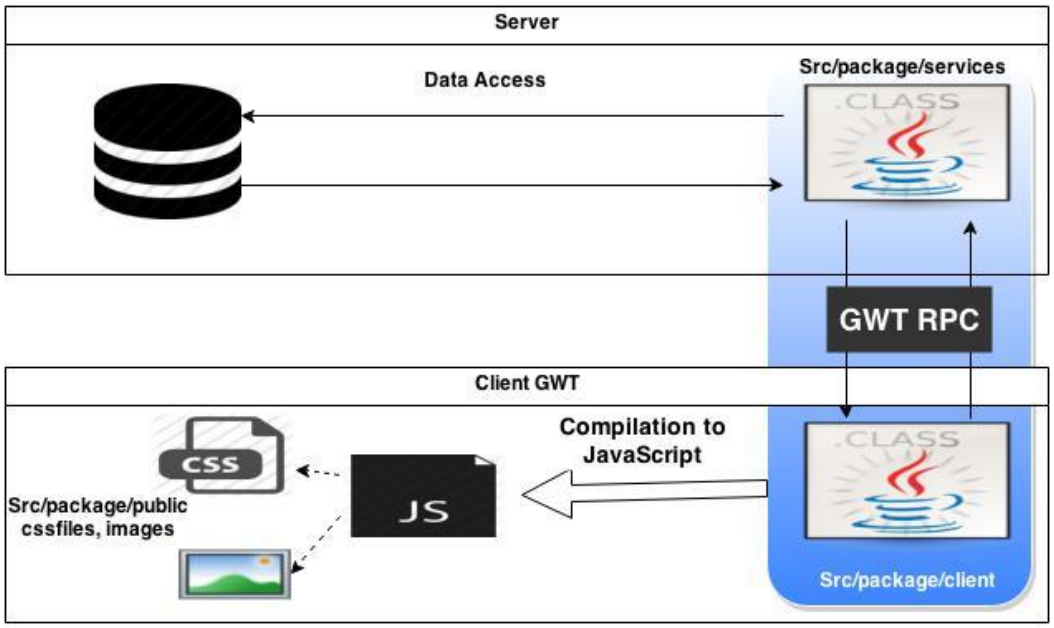

Figure 7: GWT architecture overview.

e. GWT Remote Procedure Calls

GWT provides a special mechanism that allows direct methods execution of the server-side from the client using Hypertext Transfer Protocol (HTTP) requests across the network. GWT RPC is servlet based technology and supports direct transfer of the Java objects between the client and server. The RPC mechanism is represented in the system as three main components: (i) a remote service (server-side servlet); (ii) client code to invoke that service; and (iii) Java data objects which will be passed between client and server. Figure 8 (based on graph from Tutorialspoint [13]) shows the GWT RPC Architecture.

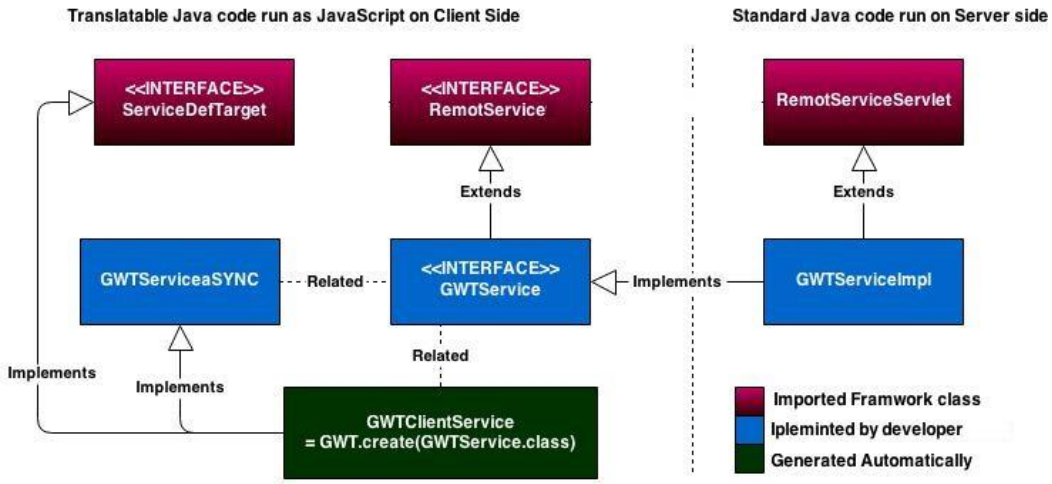

Figure 8: GWT RPC architecture in DIVA system (based on graph from Tutorialspoint).

f. GWT Add-ons

Third-party add-ons are considered as one of the most important features for GWT. They provide easy to integrate ready components. Using different GWT add-ons reduce the developing process time and enhance the overall quality of the system.

Table 1 provides a comparison of the considered client-side development technologies.

	Java Applets	HTML5 / JavaScript/	GWT	Description
Easy to use (i.e., biologist – friendly)	☐	☒	☒	For Java applets, users have to install the Java plugin for the browser. A manual enabling of Java Applet may also be required.
Quick start up	☐	☒	☒	JavaScript is always faster than Java applets since Java applets have to start the Java virtual machine.
Multi-threading	☒	☒	☐	Java Applets and HTML5 (through web workers) have support for threads however JavaScript and GWT are single threaded.
Rapid application development	☐	☒	☐	JavaScript requires less amount of code for the same task compared to Java.
Native code execution	☒	☐	☒	Code compiling is done before web publishing for Java applets, and GWT. This is not the case for JavaScript.
Accessibility and portability	☐	☒	☒	Some browsers especially mobile browsers do not support Java Applets [14].
Reuse of server-side code	☒	☐	☒	If server-side code is already implemented in Java, Applets and GWT are good choices.
Compatibility with J-Express Modularized generated results	☒	☐	☒	The development is done in Java in both Applets and GWT, making it more compatible with the generated results and allowing code re-use at the server-side.

Table 1: Java Applets, HTML5/JavaScript, and GWT technologies comparison.

4.1.2 Server-Side Technologies

For the server-side development, we need technologies that allow an easy integration with J-Express Modularized, which means that Java support is required. In addition the selected technologies should support easy communications between client and server, and maintain the server lifecycle management (life and death of server module instances, loading, initializing the classes, and making server module instances eligible for garbage collection). Finally it has to support multi-threading and to maintain the concurrency between the different sessions.

a. Servlets

Servlets are Java component-based technology developed for building interactive web-based applications, and have access to the entire family of Java APIs and a library of HTTP-specific features such as statues code, request and response headers, and cookies [15].

b. Servlets Containers

Servlet container is a piece of software (usually a component of web servers), that provides the required information for servlet hosting and is responsible for managing security, concurrency, servlet lifecycle management, transaction, deployment, and other services.

4.1.3 Technology Decision

This section outlines the decisions made for the technologies adopted in DIVA.

a. Client-Side

From Table 1, we can see that Java Applets, HTML5/JavaScript, and GWT technologies all support the main functionalities required for developing the client-side of DIVA. However, there are some advantages to using GWT that we cannot find in the others. GWT and HTML5/JavaScript support easy to use tool development where no pre-setup is required by users as for Applets. This gives a major advantage and satisfies one of the main required specification for the developed system (biologist-friendly), also the application portability is higher when using GWT or HTML5/JavaScript.

GWT has two major advantages over HTML5/JavaScript. First the client code is written in Java which increases the server code usability and makes the generated results from the

server-side fully compatible with the client-side (both written in Java). Secondly, GWT has the RPC support which reduces the effort required to organize high volumes of network traffic between the client and the server. This feature is highly desirable in DIVA to organise the distribution of computation across the client and the server. For these reasons the decision was made to use GWT for developing the client-side.

b. Server-Side

It is obvious that Java should be used for the server-side development due to its compatibility with J-Express Modularized. In addition, the Java GWT remote service servlet will be adopted as the main communication layer on the server-side. A freely available servlet container will be used to host the web application.

4.2 Software Tools

This section provides an overview of the selected development tools and software packages based on the selected technologies.

4.2.1 Java

Java is a high level Object Oriented programming language first introduced by Sun Microsystem. It is platform independent, where the application developers write their code once and run their application anywhere (in theory). The decision was to use Java JDK 1.7 for server-side development. This allows easy integration with J-Express Modularized components as part of the system logic layer, reduces the amount of new code at the server-side and put focus on the client-side development, which is considered the main challenge in DIVA.

4.2.2 File System

In DIVA the state of the system objects that have to be durably stored is serialized and saved as files in the file system. This approach was chosen for its simplicity. While it would probably be desirable to use a data storage system which offers a richer set of features, using the file system was sufficient for the purposes of the prototype.

4.2.3 GWT 2.6.0

This is the latest version of GWT; and has default support for Java 7 in addition to several improvements such as code size for the generated JavaScript. A brief description for GWT was provided in Chapter 4.1.1-d.

4.2.4 SmartGWT 4.1 Add-on

SmartGWT is GWT-based framework that has comprehensive GWT widget library for GUI layouts. In DIVA SmartGWT add-on tables layout will be considered for table-based visualisation components (omics data information table, and rank product tables), in addition to its different GUI layout components.

4.2.5 ProtovisGWT 0.4.1 Add-on

ProtovisGWT is an open source GWT data visualisation module developed as part of the Choosel Visual Data Exploration Framework [16]. In DIVA the Protovis dendrogram module will be used to develop the side and top trees for the hierarchical clustering visualisation.

4.2.6 JFreeChart 1.0.17

JFreeChart is an open source Java chart library which can be used for generating a wide range of chart types. In DIVA, J-Free-Chart will be placed at the server-side and will be responsible for generating line chart and scatter plot images in order to display them on the client-side.

4.2.7 JHeatChart 0.6

JHeatChart is an open source Java API for generating heat-map charts. The generated charts can be saved as an image file or can be used as a Java image object for additional processing. In DIVA the library will be used to generate heat-map images used for hierarchical clustering visualisation.

4.2.8 Software Tools

What follows is a brief description of the (well-known) software tools used in the implementation stage, and reasons why they are useful in developing the project.

a. NetBeans IDE

The integrated development environment (IDE) used to write the Java code is NetBeans IDE. The decision was made to use NetBeans because of the comprehensive support for the required Java technologies such as JDK 7and Java EE 7, as well as the Tomcat server integration, and Apache Maven support. This reduces the development and testing efforts and organise the code development.

b. Apache Maven

Maven is a popular open source building tool used for managing and organising projects developed in Java and in other languages. The tools developed to reduce the development effort and to provide a uniform build system as well as supporting dynamic downloads the project dependencies from different Maven repositories.

c. Apache Tomcat

Apache Tomcat is an open source web server introduced by Apache Software Foundation and implements Java servlet container. In DIVA Apache Tomcat 6 or newer will be used as the web server to host the developed application since this version supports servlet hosting required for GWT deployment.

d. Google Chrome and V8 JavaScript Engine

A JavaScript engine is virtual machine commonly used by web browsers to execute JavaScript code. Normally each browser uses an independent JavaScript engine: Spider monkey JavaScript engine in Firefox, V8 Engine in Chrome, JavaScript Core in Safari, and Chakra in Internet Explorer 9 each with their own specifications and limitations. During DIVA development and evaluation we will consider the built-in V8 JavaScript engine for Google Chrome as the main JavaScript compiler because of its performance and memory management efficiency as well as the available developer tools (Chrome Dev-Tools) supported in Google Chrome. Chrome Dev-Tools provides a good facility to track down layout issues, set JavaScript breakpoints, get insights for code optimization and help in the system performance evaluation.

Chapter 5

DIVA Design

5 DIVA Design

This chapter explains the overall design of DIVA, highlighting the decisions made about the logical organisation of the main components inside the system. In DIVA we adopted a web distributed system as the main architecture design. GWT is considered as the main development tool for both the browser based client and Java-based server, and GWT RPC is responsible for managing the client-server communication.

5.1 Incremental Development Model

DIVA development has adopted incremental development as the main development model where the software process phases are interleaved instead of treating them as separate and distinct phases, and the system is developed in incremental steps or versions. Figure 9 (based on incremental development [17, p. 33]) illustrates the concept of incremental development. It allows for incrementally developing the specifications and provides better understanding of the problems that may be faced in the different phases.

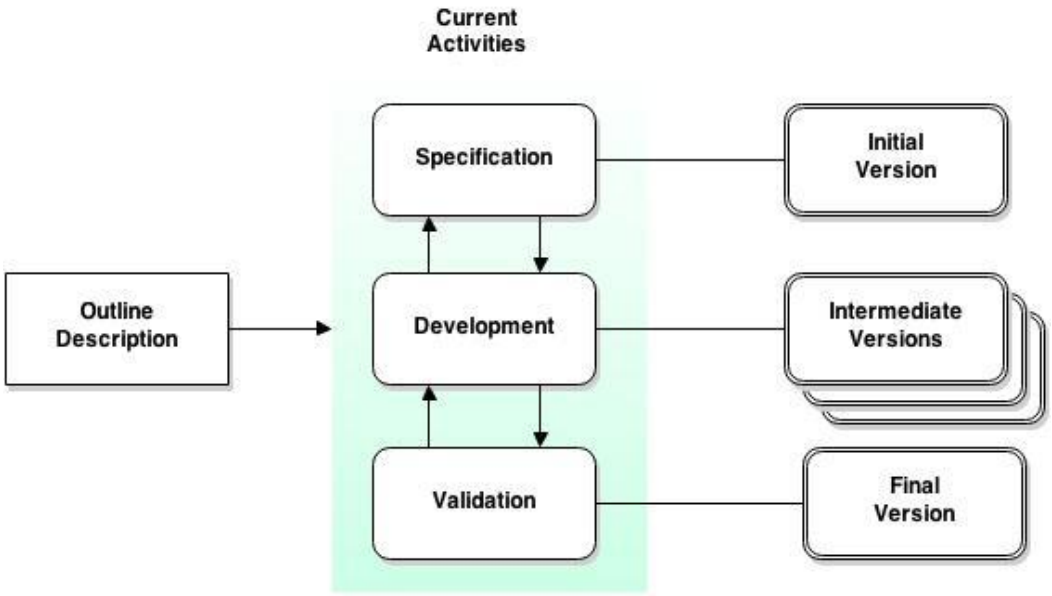

Figure 9: Incremental development model.

5.2 System Design

This phase involves architectural design, and modular decomposition. Architectural design is an early stage of the design process. At this stage the sub-system units and its intercommunication are identified, while the modular decomposition is for decomposing these sub-system units into modules [17] .

5.2.1 System Architecture

A three-tier architecture was adopted in the DIVA design, where the presentation tier, logic tier, and a data storage tier are physically separated. The server encapsulates the business logic, the client encapsulates the presentation, and (at the current stage) a file system encapsulates data storage (Figure 10).

The main advantages of the three-tier architecture are [18]

- Increased Scalability.
- Enhanced Security.
- Separation of application logic from presentation.

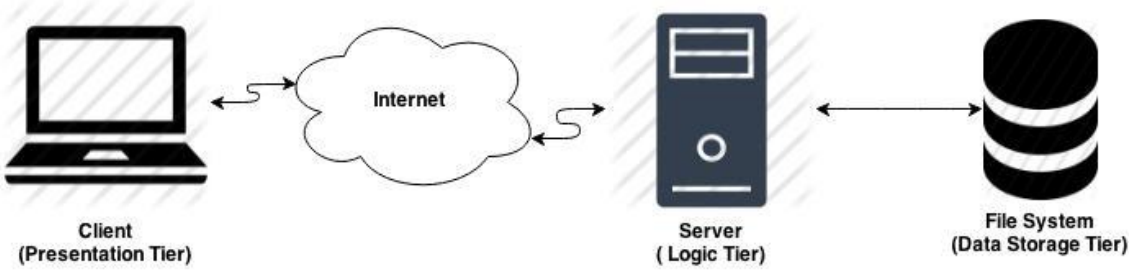

Figure 10: Three-tier architecture.

A detailed design of the client and the server, along with the design of the main components in the system follows.

5.2.2 Model-View-Presenter (MVP) Design Pattern

MVP is a design pattern developed in response to the shortages of applying Model-View-Controller (MVC) pattern to the modern component based GUI. In MVP the components themselves handle user input such as mouse movements and clicks instead of the controller's

layer. Before going through the MVP design pattern, we will give an overview of the Model-View-Controller (MVC).

MVC is a design pattern that aims to decompose the system into three main layers: (i) the model layer which holds the real business logic; (ii) the view layer which is responsible for the presentation such as text, checkbox items; and (iii) the controller layer which manages user actions, such as taking user input and passing it to the model layer for processing.

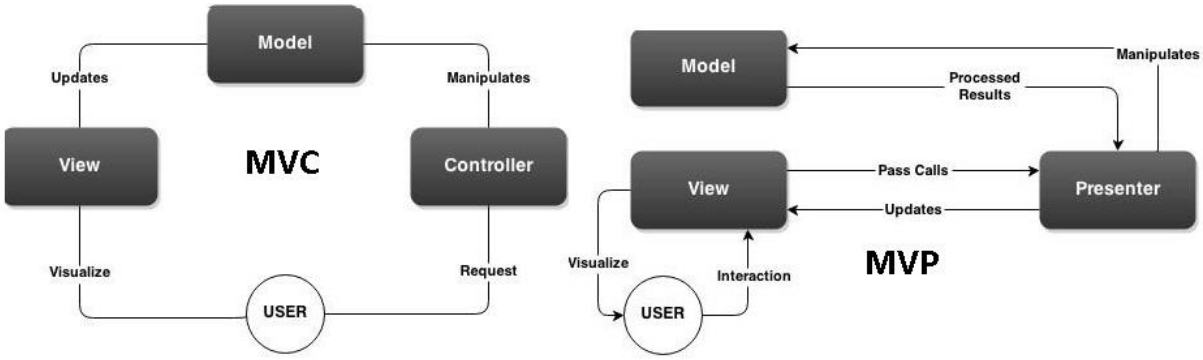

Figure 11: Comparison between MVC and MVP.

MVP is a user interface architectural pattern that considers Model-View separating concepts. In the MVP the system is decomposed into three main layers: model and view layers as in MVC with replacing the controller layer by presenter layer. The main different between MVC and MVP is that the GUI components initially handle the user's input themselves instead of controller layer in MVC. The presenter layer in MVP acts as an intermediary layer and it is responsible for managing the interactions between the model and the view layer. Figure 11 illustrates the difference between MVC and MVP processes.

DIVA adopted the Model-View-Presenter design pattern which is supported by GWT applications. In GWT MVP the design include one more layer which is the app-controller layer. This layer can be considered as a presenter for the entire GWT-Application where the centralized management occurs, and is responsible for managing the client-side application in addition to different functions (see Chapter 6.4.2 for more information).

Adopting MVP in DIVA allows decoupling the development and increase code quality by splitting the application into small independent components. This design also allows the developers to focus on the module development without worrying about other components, in addition to giving a clear logic structure for the application.

As a GWT-based application, DIVA will follow the GWT MVP design specifications where the code is organised into four main sections/components:

- **The Model** containing the application business logic.
- **The View** containing the GUI components required by the application.
- **The Presenter** containing the logic required to initialise the visualisation components as well as the data sync via RPCs back to the server.
- **The App-Controller** containing the logic that is not specific to any presenter.

5.2.3 Client Design

There are many aspects that affect the client design of an application like DIVA. The design must reflect the MVP design pattern, and in addition it must consider the generated result size of omics data that is to be visualised. Finally it has to consider the computing resources limitation of a browser based client. The design should provide the users with an easy-to-navigate graphical user interface (GUI). Thus special practices and patterns will be adapted and applied in the design trying to improve the overall system performance. The two most important practices are:

- **Limit the use of memory on the client:** The limitation of the browser memory and its effect on the client performance has to be considered during the development. In addition to removing any unnecessary features; the design of the application has to manage the memory efficiently.
- **Off-load computations to the server:** The DIVA client is designed to be a presentation tier only; i.e., no intensive processing should be performed on the client-side. The main omics data processing should be performed on the server-side as well as any computationally heavy process required for interface visualization.

a. Client Architecture

As can be deduced from Figure 12, the client adopted a layered architecture, where each layer only interacts with the layers directly below and above it. This architecture allows layers to evolve independently from one another.

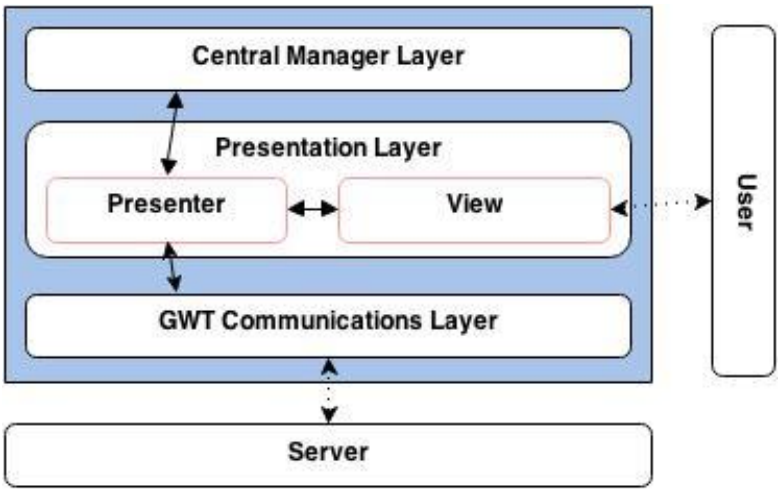

Figure 12: DIVA client architecture.

- **Central Manager Layer** is the layer responsible for coordinating the multiple interactive visualisations. It is placed locally at the client-side. This design insures fast interaction between the different modules and reduces the number of RPCs between the client and the server. In addition there was no actual need for placing it at the server-side, since the client-side supports all its required functionality.
- **Presentation Layer** represents the user view of the system. It handles the display of the results to the user and handles user interactions. This layer is where the initial processing of the user requests is done.
- **GWT Communications Layer** handles network communication with the server.

5.2.4 Server Design

There are some specifications that had to be considered when designing the server-side module for DIVA. The design should support both easy integration for J-Express Modularized components, and the GWT communication layer. In this part we are going to give an overview of the server architecture and describe the server-side design.

a. Server Architecture

As shown in Figure 13 the server adapts a layered architecture for the same reasons that were mentioned earlier.

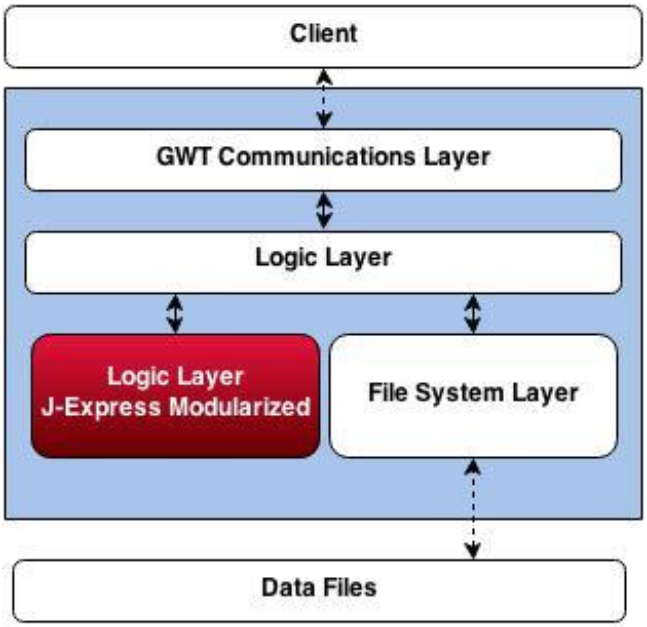

Figure 13: DIVA server architecture.

- **GWT Communications Layer** handles network communication with the client.
- **Logic Layer** contains the application core logic for the application.
- **J-Express Modularized Logic Layer** is responsible for performing the analysis of the omics data and generating results.
- **File-System Layer** is responsible for storing and retrieving the data to/from omics datasets files.

Chapter 6

Implementation

6 Implementation

In order to explain the different implementation choices for DIVA this chapter starts by presenting the main development principles and outlines the key decisions taken based on them during the implementation process. Next an overview of the developed system is given with brief explanations of the different system processes and components followed by outlining the challenges faced during the development process and how they were addressed.

6.1 Main Development Principles

There are five principles that guided the DIVA development process. In this section we describe these principles and their importance in achieving high system performance.

6.1.1 Minimise the Client-Side Rendering, and Repaints

The rendering, and repaint processes are normally heavy for the client-side computations capabilities. DIVA systems support multiple interactive visualisations of large scale omics data. This means that the system should efficiently support a large number of rendering and repainting processes. This can affect the system performance and may become critical processes that can affect the feasibility of the whole system. To overcome this main challenge, reducing the amount of rendering and repaints on client as much as possible is important

6.1.2 Minimise Computations on the Client

Off-loading relatively heavy computations from the resource-limited client to the server is one of DIVA's main design principles. This was followed in the implementation phase by moving the analysis process and as much as possible of the heavy processes required for maintaining both the interactivity and the visualisations to the server-side. However, some computation was still kept on the client-side to reduce network traffic whenever it was feasible to do so.

6.1.3 Central Manager

The third principle considered during the development stage is using a single central manager layer to coordinate the selection updates between the visualisation components. The selection update process is unified for all visualisation components, and done locally on the client-side.

6.1.4 Memory Management

Another important principle is keeping the memory usage on the client-side to minimum and increasing the efficiency of the automatic memory management provided by the JavaScript Garbage Collectors. The problem was to investigate the optimum data size used on the client-side without hitting the client system performance.

6.1.5 Network Communication Management

Network communication management is a corner stone for DIVA. The design we choose requires a large number of network calls between the client and the server to distribute the computational processes and data between the client and the server.

The network communication management depends on both the client and the server implementations, where a communication layer is developed to organise the different network communications between the client and server. Using the GWT RPC plays an essential role to increase the communication efficiency, and provides an easy way to develop communication functions between client and server.

6.2 Implementation Decisions

Based on the principles described in section **Error! Reference source not found.**6.1, the decision was made to: (a) move the rendering and repaints of heat-map, PCA scatterplot, and profile plot to the sever-side; (b) use partial rendering tables on the client-side to visualise the table-based visualisations (omics data information and ranking tables); (c) develop a Selection-Manager component that represents the central manager layer in the system; and (d) distribute interactivity computational processes across the system.

6.2.1 Server-Side Charts Rendering and Repaints

As shown in Figure 14, the rendering process starts from the presenter layer (visualisation component) where the client sends to the server the required information for rendering the graphs (selected data indexes in the case of PCA and Profile plot, and in case of heat-map arranged rows and columns indexes). The server starts generating the chart images, compresses them into Base64 string and sends them back to the client combined with the information required for interactivity. The process can be summarised in the following steps:

1. The client sends request to render chart.
2. The server processes the request.
3. The server generates chart image.
4. The server converts the images into Based64 strings.
5. The server sends the images and information back to the client.
6. The client visualises the image.

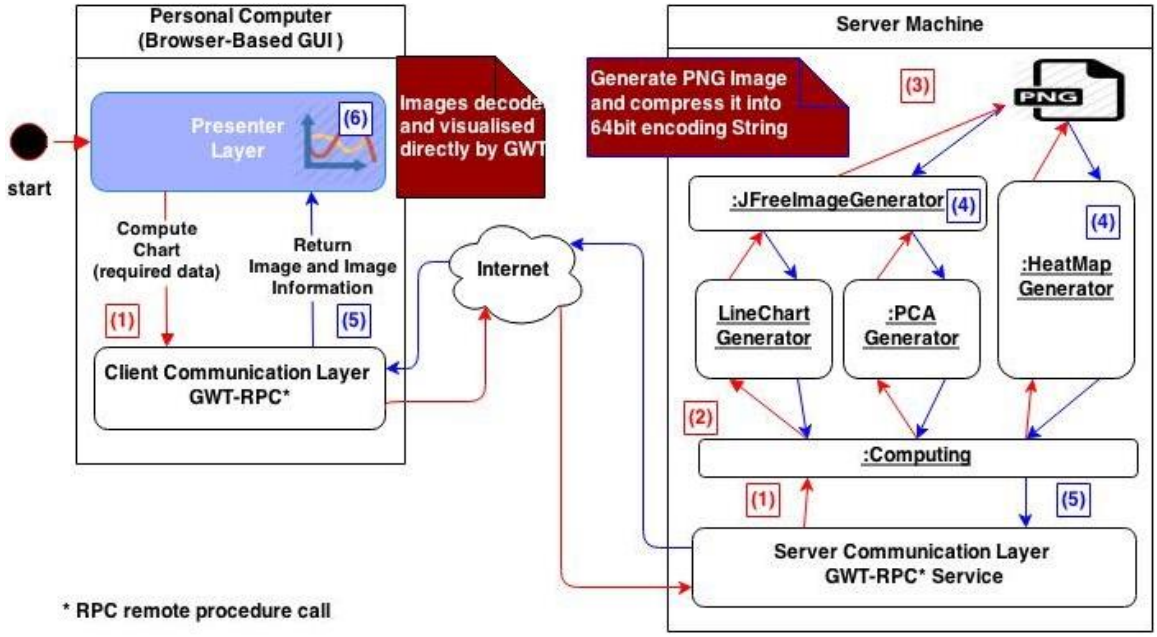

Figure 14: Server-side charts rendering.

Using this technique boosts the performance of the client-side and reduces the effect of the data size to a minimum, where the results are represented on the client-side as a grid of pixels in an image. The high speed of GWT RPC communication plays a key role here by organising the client-server connections and reducing the time required for client-server data transfer time.

Three additional features make the implementation of these components simpler and improve the system performance. First, is the auto image decoding for Base64 string supported in GWT. The images are transferred between client and server as strings and directly visualised on the client-side. Second, is the mouse event handlers supported by GWT for images. This feature provides an easy way to handle user selections on the images by calculating the selection coordinates and converting it into indexes. Finally, the ChartRenderingInfo which is a class provided by JFreeChart and contains all the required information to map the selection coordinates to omics data indexes (see Chapter 4.2.6 for more information about JFreeChart).

6.2.2 Client-Side Partial Rendering Tables

The decision was made to use tables layout provided by SmartGWT add-on (see Chapter 4.2.4). This table layout has one important feature that increases the overall client performance which is set showAllRecords function. This feature allows rendering only the visible parts of the table in the view port. Using this table layout has large impact on the table rendering speed and the overall client performance especially with large datasets.

6.2.3 Selection-Manager

Selection-Manager is the component responsible for coordinating user's selection notifications and sharing the selected data indexes between the different visualisation modules. The component consists of two main Java classes; SelectionManager and ModularizedListener.

Both classes are based on the J-Express Modularized Selection-Manager layer with some modifications to suit the browser based client environment. The selected omics data will be represented in the Selection-Manager components as a Selection class which contains an integer array with the data selection indexes and the selection type (rows or columns).

The ModularizedListener class is an abstraction class which contains the required functions and attributes for registering the visualisation module component into the Selection-Manager. All visualisation components inherit this class and implement the updating selection methods to be eligible for Selection-Manager registration and to allow the Selection-Manager to perform the required updates. The modularised listener is the class responsible for unified the selection update process between different visualisation components.

The activity diagram of Figure 15 explains the central manager principle.

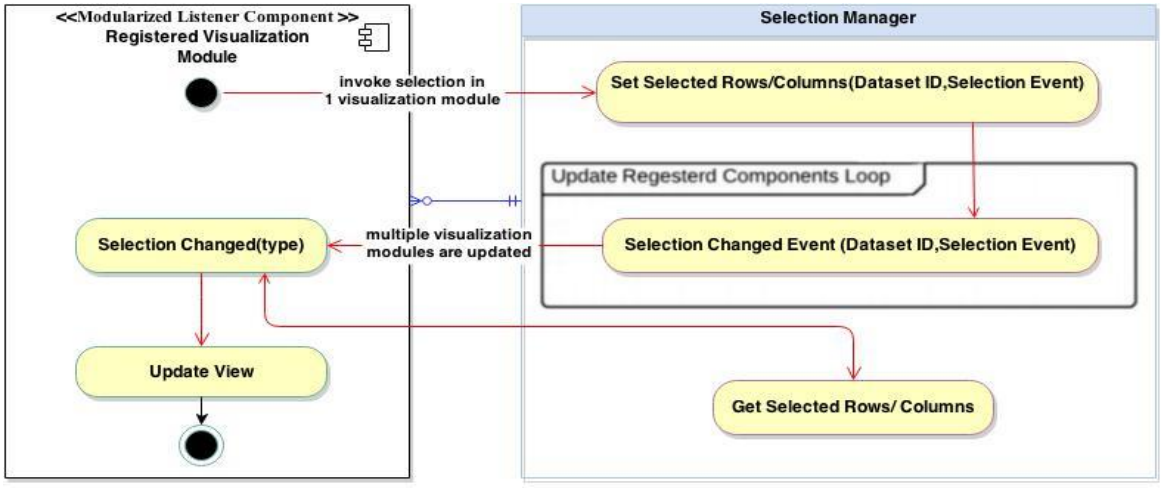

Figure 15: Activity diagram for the central manager.

6.2.4 Interactivity computational processes

Interactivity computational processes consist of three main functions. The selection function is the function responsible for converting the data selection into data indexes. Post-selection function (Selection-Manager) is the function responsible for coordinating selection between different visualisation components; and pre-visualisation update function is the function responsible for optimising the selected indexes to suit the visualisation component. The challenge here is how to efficiently distribute these functions between the client and the server supporting fast interactive selection.

The selection function: the selection function for the components draw on the client-side is easy to handle. Representing the data on node tree allows adding data indexes reference to these nodes. The decision was made for this case to leave these functions on the client-side. On the other hand this function can turn to be heavy for data represented as a grid of pixels in an image as PCA plot especially with large datasets. The function in this case will be responsible for converting the x, y coordinates into omics data indexes. for large scale datasets (25 K rows) this function is turned to be very time consuming and also caused blocking in the main browser thread (the browser is single threaded) which affects the overall performance of the system. To solve this we have decide to offload the process to the server-side. This solution enhances the system performance and increases the interactivity response time for large selections. Additionally this process hides the unresponsive warning caused by

large selections in the scatter plot. The client-side component does not have this problem since it was easy to add index reference for the omics data for each selectable node.

Post-selection function (Selection-Manager): To handle multiple updates for visualisation components and organise this process without causing slowness or blocking for the singly threaded JavaScript engine we used a timer to organise the updates and inserting breaks avoid making the system unresponsive (see Chapter 6.5.2-b for more information about breaking long-executing JavaScript using Timers). The Selection-Manager is placed in the local client to reduce both the effect of network calls on the performance and the coupling between the client and the server.

Pre-visualisation update function: this function is important for some components like rank product tables, where the tables are sorted by ranking values and not by the omics data indexes. For updating such tables' visualisation there are problems with mapping the omics data indexes to rank table indexes where this process turned to be very heavy with big selections from large datasets. To solve this problem the re-indexing process is offloaded to the server-side. This solution shows a good impact on the overall system performance especially with large datasets selection.

6.3 Functions and Visualisations

This section shows the main related functions that have significant impact on this study using graphs, UML object sequence diagrams (OSD) and screenshots for the normal scenarios. To clarify these functions and make sense of the descriptions, we start by giving an overview of the flow of control of DIVA, stating the processes order. After viewing the process orders, a detailed description of these functions is provided. The descriptions of the functions considered less important for this study are available in Appendix A.

6.3.1 System Overview

As shown in Figure 16, the flow of control of DIVA has a number of functions (in blue rounded rectangle) and visualisations (in white rectangle) in addition to one function in purple (update selection) which represents with its interconnected functions the central selection update process.

The flow of control for DIVA system in Figure 16 can be explained using the following ordered steps:

Start Up System → Load Web-Page (Start Session) → Select Dataset (Load Dataset) → Update Dataset Information → Invoke Analysis Functions → Selection Process → Update visualisation view → the rest of processes.

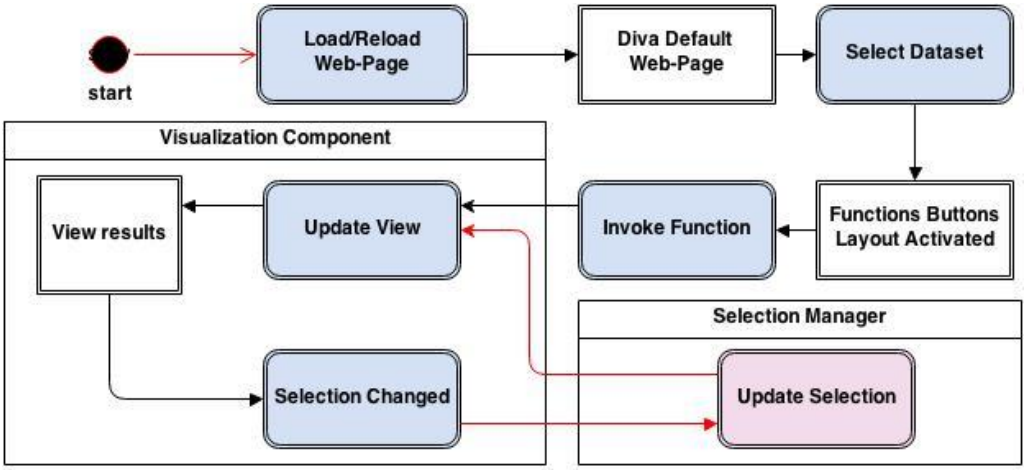

Figure 16: Flow of control for DIVA system.

6.3.2 Invoke Analysis Functions (Profile Plot-PCA-Rank Product)

When a user invokes one of the following analysis methods (profile plot-PCA-rank product), the system performs the required analysis, stores the generated results in the file system, and returns the analysis results to the client to be visualised.

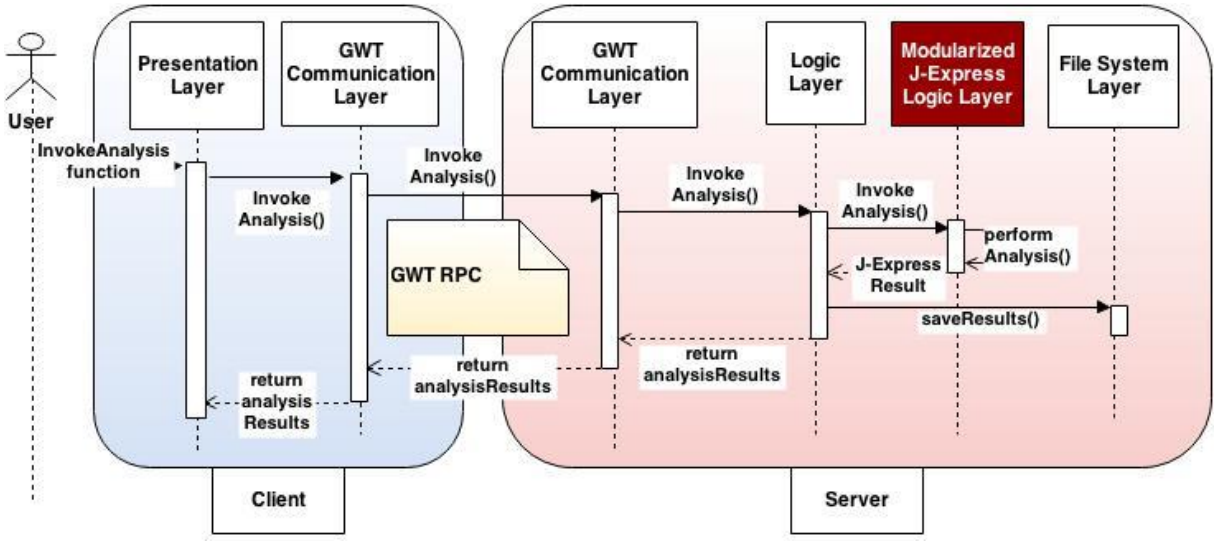

Figure 17: OSD invoke analysis function.

The visualisation of PCA and profile plot is described in Chapter 6.2.1, however for the table-based visualisations (rank product analysis, and omics data information tables), a partial rendering table (list grid) provided by SmartGWT Add-on (see Chapter 6.2.2) is used.

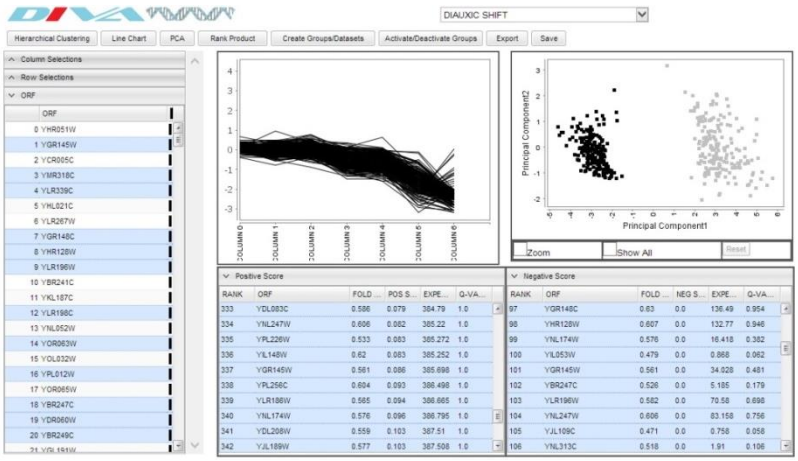

Figure 18: Invoke PCA, profile plot, and ranking analysis visualisation.

6.3.3 Selection Process

As described earlier the system adopts an MVP design pattern, which means that the visualisation components will initially handle the user's selections themselves. When a user selects data from any of the visualisation components (tables or graphs) the component will be responsible for converting the selection into a group of indexes and then notify the Selection-Manager with the selection indexes and the selection type (rows or columns selection). The central manager will then be responsible for notifying all the registered visualisation modules and pass to them the selection. Each visualisation component will then be responsible for updating its visualisation and highlight the selected data.

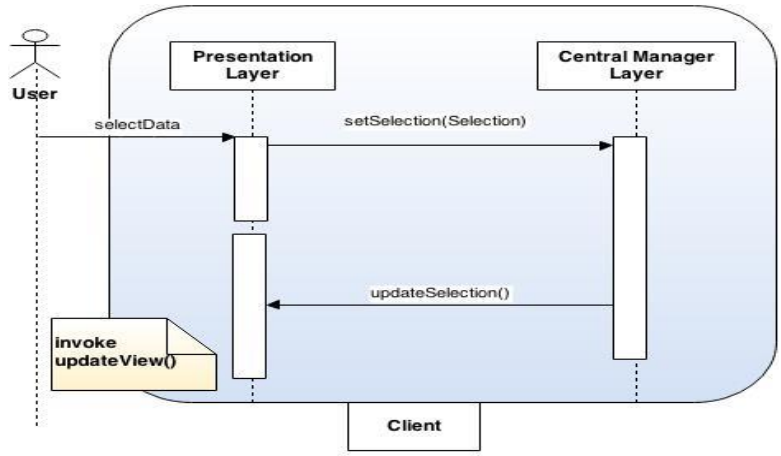

Figure 19: OSD selection process.

Please note that in the hierarchical clustering component the selection data is highlighted in the tree only when the user selects nodes from the side/top dendrograms. However a clean selection for the tree happens upon receiving change selection notifications from other visualisation modules to avoid any misleading visualisation.

6.3.4 Update view for PCA, and Profile plots (RPC required)

The update visualisation process for both PCA and Profile Plot is different from the other visualisation components. In these components the update process happens by generating updated chart images on the server-side and then sending it to the client-side to be visualised.

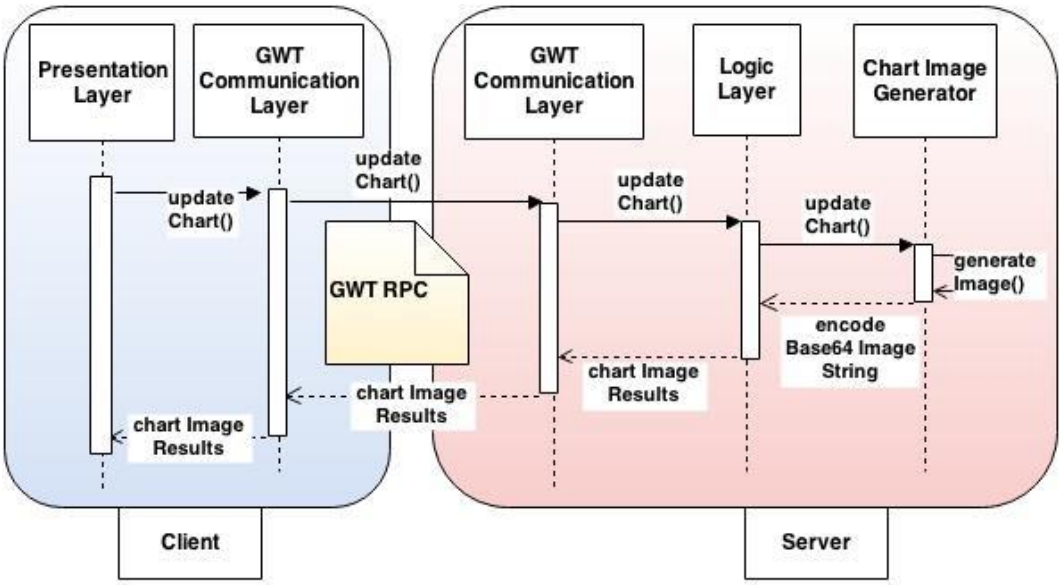

Figure 20: OSD update view for PCA and line chart.

6.3.5 Invoke Hierarchical Clustering Analysis

When the user invokes the hierarchical clustering function, the system performs the required analysis on the server-side, stores the generated results in the file system, and returns the clustering results to the client-side to draw the side and top clustering tree on the client-side (the dendrograms). After drawing the trees on the client-side the component sends the arranged omics data indexes (rows and columns indexes) to the server-side to compute the heat-map and return the generated image results to display on the client-side.

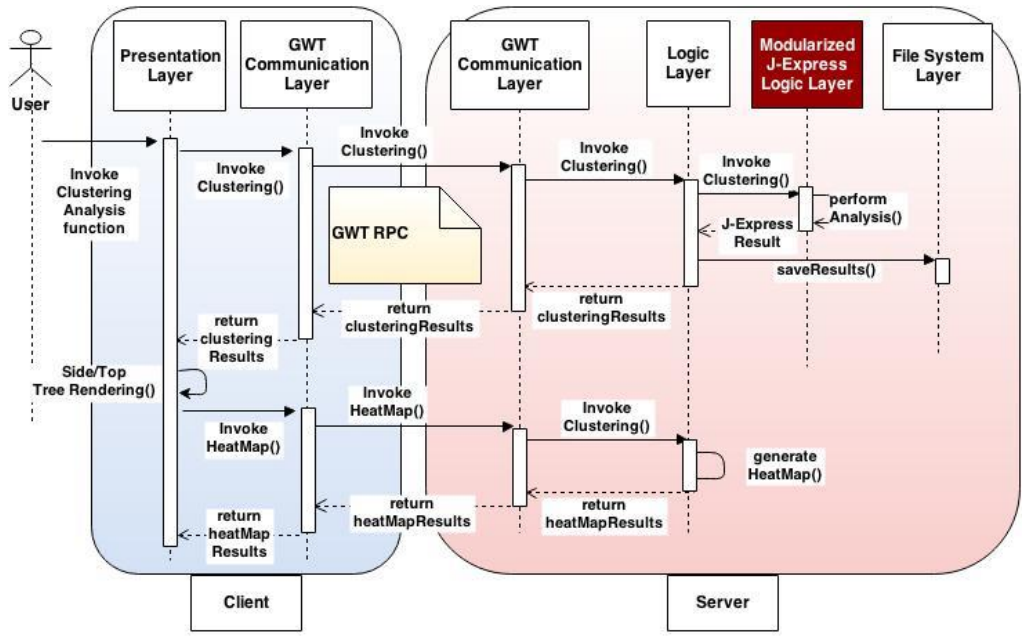

Figure 21: Invoke hierarchical clustering analysis.

This is the second important process in the system since we distribute the rendering processes of the component between the client (side and top tree) and the server (heat-map graph). The heat-map is generated on the server as an image and then sent to the client-side to be visualised.

JHeatChart is used to generate the heat-map images, but Protovis GWT Add-on is used to draw the top and side tree on the client-side. A customised tool-tip is developed to provide the description and information for the hierarchical clustering component (the side/top tree and heat-map) instead of using the default tool-tip generated by the charts to reduce the cost of the tree repaints.

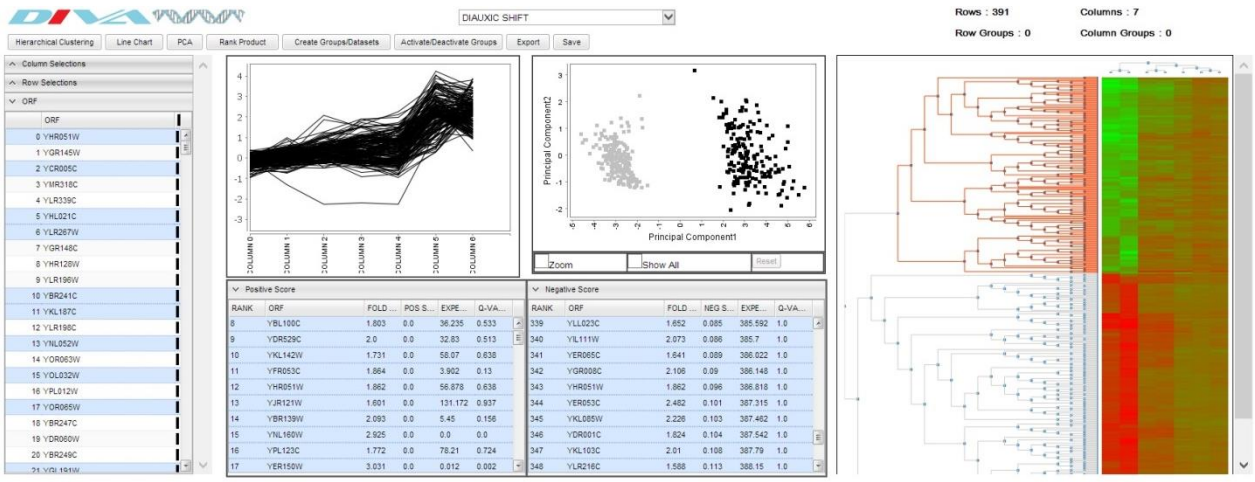

Figure 22: DIVA selection update in all visualisation types.

6.4 Main System Components

This section provides an overview of the main DIVA components clarifying their main roles and responsibilities.

6.4.1 Selection-Manager

As we explained in 6.1 the Selection-Manager is the component responsible for coordinating the multiple interactive visualisations in DIVA. The Selection-Manager main function is not only to update the visualisation components but also to organise the updating process reducing the effect of long-scripting process on the client-side performance.

6.4.2 The App-Controller Component

The app-controller in DIVA is part of the presentation layer and is represented in DIVA as DivaMain class. It is the component responsible for managing the client-side modules and provides the following functions:

- Provides the client entry point
- Initialises different components (the Selection-Manager as single instance variable, GWT-Client-Service, HTML components, and all the GUI panels)
- Manages different functions (updates the dataset information, omics analysis functions invocations, export data, save data-sets, create groups etc.).

6.4.3 GWT-RPC-Communication Component

This component is represented in DIVA as GWTService class and GWTServiceAsync Java interface on the Client-side in addition to GWTServiceImpl class on the server-side. This component is responsible for handling all the requests, responses and the transferred Java objects between client and server in addition to managing the serialization and de-serialization processes for the transferred objects.

6.4.4 DIVA Logic Component

This component has the core of the application logic and it is responsible for optimising the generated results from Modularized J-Express logic layer to suit the visualisation modules on DIVA. In addition it provides the required utilities for other services like exporting data, updating datasets, and facilitating communication to the storage layer. This component consists of one main class Computing in addition to other required utilities classes. This class interacts directly with J-Express Modularized component.

6.4.5 J-Express Modularized

This is the component responsible for performing the selected omics analysis in DIVA. This component is represented in the system as six main classes imported from J-Express Modularized:

- SomClustCompute class: performs clustering analysis and generates ClusterResults class.
- PCACompute class: performs principal components analysis and generates PCAResults class.
- Compute Rank class: performs rank product analysis and generates RPResults class.

In addition to previous classes J-Express Modularized provides some other classes like Dataset, Groups and other classes required for the analysis processes (more information about J-Express Modularized available in Chapter 1.3.3).

6.4.6 File System Component

This is the component responsible for saving and retrieving data to/from files in addition to reading and writing data (datasets or computing results) from/to serialised objects or text

files. This component has main DB class represents the file system in addition to FileSystemUtil, and DatasetFileReader classes.

6.4.7 Omics Data and Generated Results Representation

The representation for omics data and the generated results is different in both client and server processes due to the difference in the specifications.

DivaDataset class represents the omics data inside the system (server-side only). The class inherits the J-Express Modularized Dataset class and fully compatible with J-Express Modularized component.

The following classes are shared between client and server and fulfil the serialization specifications required by GWT-RPC.

DatasetInfo class provides the basic information of the omics dataset in addition to omics table data.

SomClusteringResult class provides the required information to draw the side and top trees (dendrogram) as well as the required information for tool-tips and data interactivity.

HeatMapImageResult, LineChartResult, PCAImageResults classes provide the clustering, line chart, principal components analysis results images (Base64 encode string) respectively in addition to the required information for interactivity.

RankResult class provides the information and data required for rank product table.

6.5 Development Challenges

This chapter discusses the issues and challenges faced during the implementation phase. The complete documented source code for DIVA is available at:

http://diva-omics.googlecode.com/svn/trunk.

The DIVA prototype was developed in Java using GWT and run in the browser environment as a JavaScript-based application. In this section we describe the limitations for such applications, and how different techniques for developing GWT and JavaScript based applications were implemented to address these challenges and to enhance the overall system performance.

6.5.1 JavaScript Engine Memory Limitations

Google Chrome with its built-in JavaScript engine (V8) is used as the client-side environment in DIVA during the development and testing. The memory allocation defaults for the V8 engine are 700 MB and 1400 MB on 32 bit and 64 bit machines respectively. In the latest versions of V8, memory limits on 64 bit systems are organized by the operating system which means that the true limit cannot be generally stated [19]. Memory is normally considered as one of the important performance bottlenecks. Any shortage in memory has a big impact on the application and can lead to slowness, instability, unresponsiveness and even browser crashes. In DIVA the memory is a key for the application performance since the application deals with large scale omics data, and multiple interactive visualisations. These factors make the memory management a major challenge for the client-side development. Thus we are going to outline the main problems that cause memory shortages and show how we address them during the implementation stage.

a. Browser-based Client Memory Leak

The memory life cycle for JavaScript normally goes through three main steps: (i) allocate the needed memory; (ii) use this memory; and (iii) release the allocated memory when no longer needed. These steps explicitly happen in JavaScript. In this section we are going to focus on the last step where the JavaScript Garbage Collector (GC) automatically tracks memory allocation and releases any allocated memory that is no longer needed.

Although JavaScript web based applications have an automatic memory management provided by the JavaScript GC, these applications are subject to memory leaks, i.e., continuous loss of available computer memory when for some reason the GC cannot release the unused memory. In the case of DIVA the risk of memory leaks are higher since the application deals with different visualization components and large scale data sizes, which makes efficient memory management an essential task. Development procedures based on developers' experiences with GWT and JavaScript based application development were used to boost the application performance and reduce the memory leaks as much as possible.

Avoid instance/ global / static variables: For objects to be eligible for a JavaScript GC, they must not be in context or referenced by variables in context (cannot be reached by any in-context variables). The global variables are always in context therefore they are never

destroyed by the garbage collector which may cause memory leaks. Static variables also cause same problem since they have a global state and extent the entire run of the application (long life time), that is why they are not eligible for garbage collection while the class is loaded. In DIVA the number of instances and static variables is reduced to a minimum and local variables are used instead.

De-referencing: One of the best ways of making an object eligible for the JavaScript garbage collector is assigning "null" to its object reference. In DIVA all the object references are assigned to "null" after they finish their role in the application.

b. Code Separation

One of the main concepts we considered during the development is to keep the memory usage on the client-side to minimum. The problem was to investigate the optimum data size we can keep on the client-side without hitting the client system performance. The decision was made to separate the analysis results code and send only the very basic code required to maintain the data interactivity as different data-structures that have the omics data indexes, graphs tool-tips information, basic datasets information, in addition to data required for initialising the visualisation components or drawing it (if the drawing is done on the client-side). The reason for keeping these minimum data on the client is that we can limit the number of calls to the server-side as much as possible.

c. Reduce Rendering on Client-Side

Client-side rendering causes a significant increase in the memory usage (especially for large scale data) where a large number of nodes is added to the DOM tree and kept in the memory (these objects are not eligible for GC). In addition the size of the generated images is relatively small. Offloading the rendering and repaints of the charts to the server reduces the memory usage for the client-side.

6.5.2 JavaScript Execution Limitations

In DIVA performance is one of the essential factors that affects the feasibility of such a system. The system is required to perform the requested omics analysis and interactively visualize the results, all within an acceptable time frame. The application performance is affected by many factors, as the client and the server computing capabilities, the browser

type, and network speed. However different practices and procedures are adopted in DIVA to enhance the overall system performance and optimize the use of the available computing resources. Even though different optimisations are done at the server-side where most of logic is located, we focus on the client because this is where the main limitations of systems like DIVA are located. As it is always possible to upgrade the server, this is not always an option for the clients.

a. Breaking Long-executing JavaScript Using Timers

JavaScript interpreters are single-threaded which means that only one script can be executed at any given time. At the same time browsers have limits on how long JavaScript can run. In DIVA the problem appears when there are multiple updates for the visualization components. The long-executing JavaScript code may in this case lead to slowness, unresponsive script warnings or even crashes. To prevent the browser from displaying a warning about long-running scripts, and to reduce the chance of this occurring, breaks are inserted to delay the different executions in the updating loop. GWT supports a timer class that allows a delay of execution (in milliseconds) using a scheduled repeating function. This gives a chance for the updating executions to be organized into a queue and executed one by one.

b. Minimizing Client-Side Rendering, Repaints, and Reflows

Rendering is the image generating process from a model, like drawing charts or tables. In the client-side rendering the browser normally analyses the HTML source code to construct a Document Object Model (DOM) tree where every HTML tag is represented as a node in the tree. Next, styling information is applied to the DOM tree by analysing the Cascading Style Sheets (CSS) code and combining the extracted style information together with the DOM tree constructing a render tree. After constructing a render tree the browser is ready to draw and view it on the screen (Figure 23) [20].

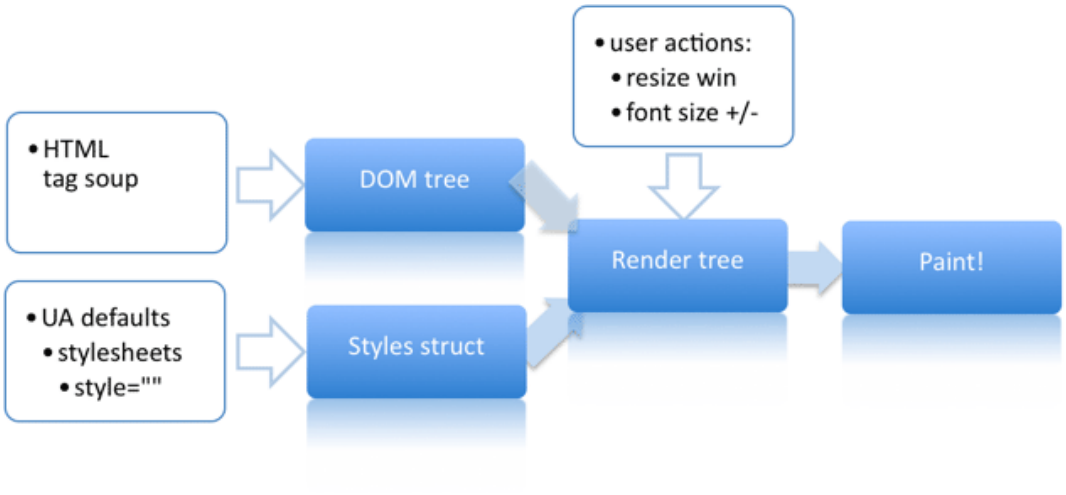

Figure 23: The rendering process [20].

The rendering process in system like DIVA can turn out to be very heavy in terms of overhead, especially with the large scale of the data, i.e., a large number of nodes in the DOM tree.

A repaint is the process that happens when changes are made to the DOM tree. There are different processes that trigger the repaint process, e.g., changes in geometric properties of a node or because of change in style like changing background colour. In the repaint process the main layout is not changed, e.g., the selection highlight on the hierarchical clustering dendrogram. This process is heavy since the browser must verify the visibility of all other nodes in the DOM tree. Client-side rendering and repaints are heavy processes especially in DIVA case where size of data has major effect on the client-side performance.

The reflow process involves changes that affect the layout like resizing the window, or scrolling. Reflow is critical to the performance when the DOM tree is large. An element reflow causes the subsequent reflow of all child and parents elements as well as any elements following it in the DOM tree. Therefore reducing client-side rendering can decrease the effect of the reflow process on the client system performance by reducing the size of the DOM tree (e.g. instead of rendering thousands of nodes for each graph on the client-side, using images generated on server-side and rendering it as one node only on the client- side). The decisions made to reduce the amount of rendering on the client-side can be summarised as follows:

For Rank Products and omics data information tables, partial rendered customised SmartGWT tables is used where the rendering is happening to the visible records only instead

of pre rendering the full records (rendering only records visible in the viewport). This table layout improves the system overall performance especially with the relatively large scale datasets (24K records).

For Profile Plot, PCA Scatter Plot, and Heat-map, the rendering and repainting processes are done on the server-side as images and then visualised on the client-side. It is worth mentioning that there is a number of optimisation processes done on the server-side for generating the updated charts in order to reduce the processing time for generating the images, and enhance the interactive selection process.

For The Hierarchical Clustering Side and Top Trees, the rendering and repainting processes are done on the client-side. However, we tried to reduce the repaint processes as much as possible, for example by creating external tool-tips to show the nodes information instead of using the self-generated tool-tips (requiring chart repaints). Additionally, code optimisation was carried out several times, removing most for-each loops, and also to use different data-structures to reduce the computing time as much as possible.

Chapter 7

Results

7 Results

In this chapter we evaluate the system and provide the main performance results for the DIVA prototype. These results are extremely important for this study, where the final evaluation and the feasibility of the system depend on these results.

Performance testing is a type of testing concerned about the responsiveness, throughput, reliability, and/or scalability evaluation of a system under a specified workload. In this thesis performance testing is used as a technique to give clear insight about the system behaviour, and to evaluate the current system limitations.

Measuring the performance characteristics, such as memory usage in a distributed system like DIVA is not straightforward since there are distributed components and layers working together to achieve the system's functionality. In the following sections of this chapter we carry out a performance evaluation using different techniques, scenarios and development tools trying to measure response time and memory usage.

7.1 Testing Environment

This section defines the - testing environment such as hardware specifications, software, and network configuration, as well as the different tools used in the testing process. These can be summarised as follows.

Client and Local Web Server Machine: The machine specifications are a Lenovo ThinkPad S1 Yoga PC [Processor: Intel(R) Core (TM) i7-4500U CPU@ 1.89 GHz, Memory: 8.00 GB RAM, and System type: MS Windows 8.1 64-bit operating system].

Local Web Server: Apache Tomcat 8.0.3.0 and Java Platform JDK 1.7 with Java virtual machine (VM) that has available maximum memory 5.50 GB.

Web Server: Virtual server running on Xen, 2proc, memory: 4GB.

Network Connection Speed: 35 Mbps.

NetBeans Profiler: a feature included in NetBeans to profile CPU and memory usage as well as basic JVM monitoring.

Client-Side Browser: Google Chrome version 34.0.1847.131m 32bit with V8 JavaScript engine and the Chrome Development Tools (Dev-Tools) was used to give clear insight into the memory usage and JavaScript processing time.

Testing Datasets: For testing the performance five datasets of different sizes were used. Three of the datasets represent sample datasets from genomics data and two represent proteomics data. The datasets are as follows:

- DS1 - Diauxic shift (391 rows and 7 columns).
- DS2 - 5K GENES BY 24 SAMPLES (5002 rows and 24 columns).
- DS3 - 24K GENES BY 24 SAMPLES (24 526 rows and 24 columns).
- DS4 - DIVA Proteomics Example Dataset Regulated (1399 rows and 12 columns).
- DS5 - DIVA Proteomics Example Dataset All (5373 rows and 12 columns).

7.2 Performance Evaluation

In distributed systems like DIVA, the entire system performance is dependent on the three main tiers; (i) the client, (ii) the server, and (iii) the network connection between them. This makes the system performance evaluation a hard task. To simplify this mission, the decision was made to evaluate each of these tiers independently. This way gives a clear idea about each tier, and provides a clear overview for the overall system performance.

There are two main system settings for the testing environment:

1. **Settings-1**
   Client-side runs on local machine.
   No Network.
   Server-side runs on same local machine.
2. **Settings-2**
   Client-side runs on local machine
   Network speed 35 Mbps
   Server-side runs on online server machine

Setting-1 was used to evaluate the client-side performance using the Chrome Development Tools to measure the memory usage on the client-side, the response time and other performance characteristics for the interactive data visualization.

Also Setting-1 was also used for server-side memory usage and garbage collectors efficiency evaluation. This test was done using the NetBeans profiler on a local machine where it is easy to monitor the memory usage and to get an overview over the garbage collection activity on the server-side.

Setting-2 was used to evaluate the overall processing time for the server-side plus network transmission time using the Chrome Development Tools (network activity monitor).

The tests were performed with a single user invoking the data analysis and performing the selections to test the interactive visualization. Larger multiple selections were also performed to stress the interactive visualisation on the client-side to evaluate the stability, scalability and system responsiveness.

7.3 Evaluation Scenarios

The evaluation scenario for the server-side testing (local and online) followed the same order and criteria, to make the results consistent and to reflect the real world system performance. However when testing the client-side some additional tests were carried out to give better insight into the client-side limitations and shortages.

The standard client-side testing scenario:

1. The user selects the dataset (start by smaller size datasets to larger ones).
2. Invoke the line chart visualisation process.
3. Invoke PCA analysis process.
4. Invoke Rank Product analysis process.
5. Select all data using PCA component (Selection Update_i).
6. Invoke hierarchical clustering analysis process.
7. Select all data using root node on side tree for the hierarchical clustering dendrogram (Selection Update_ii).

The testing scenario will follow the following order:

Loading the main page (start session) → do steps 1 to 7 for DS1 → repeat steps 1 to 7 for DS2 → repeat steps 1 to 7 for DS3 → repeat steps 1 to 7 for DS4 → repeat steps 1 to 7 for DS5→ repeat steps 1 to 7 for DS1 (DS1_HM - DS1 with High memory usage) → end session.

Reloading the main page (start new session) → do steps 1 to 7 for DS5 (DS5_LM-DS5 with low memory usage) → end session.

The standard server-side testing scenario (local and online server machines):

1. Start-up the system, convert the datasets text files (.txt files) into stored serialised dataset objects (.ser files) and load dataset lists.
2. The user selects the dataset (start by smaller size datasets to larger one).
3. Invoke the line chart visualisation process.
4. Invoke PCA analysis process.
5. Invoke Rank Product analysis process.
6. Invoke hierarchical clustering analysis process.
7. Select all data using root node on side tree for the hierarchical clustering dendrogram.

7.4 Evaluation

In this section we give an overview of the performance evaluation results using a number of charts for the different processes. The evaluation includes memory usage, scripting, paints and rendering time for the client-side in addition to server-side memory usage (using a local server), server-side processing time and network transmission time (using an online webserver). The full evaluation and testing tables are available in Appendix B. The evaluation results are discussed in Chapter 8.

7.4.1 Client-side Evaluation

As mentioned before, the client-side testing is performed under test Settings-1. The test started by loading DS1 on clean memory browser and performing the different testing steps until the last testing step for DS1_HM without cleaning the client memory usage (same session was kept alive). Then we forced the page reload to clean the memory usage and perform the testing steps from 1 to 7 for DS5 only (DS5_LM).

a. Client-side Memory Usage Evaluation Results.

Figure 24 shows the client-side memory usage in megabytes. The test is done in one session for DS1 to DS1_HM. Then cleaning up the client-side memory usage (reload the page) and retesting is happened for DS5 alone (DS5_LM). For each dataset, the test starts by loading the dataset and invoke line chart analysis (deep red bars), invoke PCA analysis (green bars),

invoke ranking analysis (purple bars), then perform the first data selection (blue bars). After that invoking the hierarchical clustering is performed (orange bars) followed by second data selection (light blue bars).

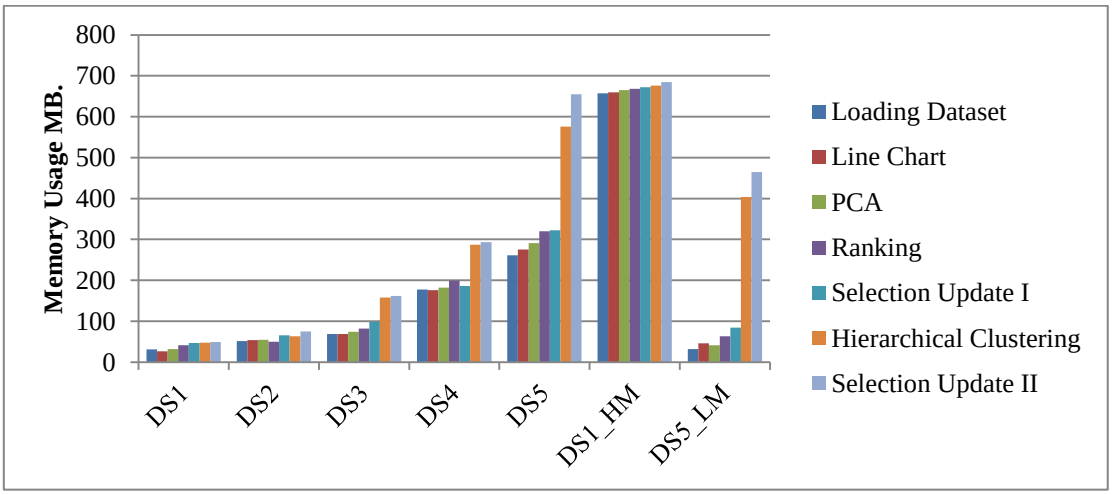

Figure 24: Memory usage on client-side.

b. Client-Side Scripting, Paints and Rendering time

The following figures represent the client-side scripting (blue), paints (green) and rendering (red) time in second for different visualisation components in DIVA. Every dataset label is represented as index number in the x-axis.

Index 1 represents DS1 Index 2 represents DS2 Index 3 represents DS3
Index 4 represents DS4 Index 5 represents DS5 Index 6 represents DS1_HM
Index 7 represents DS5_LM

Please note that the index 6 represents DS1_HM (391 rows) with high memory usage on the client-side and index 7 represents DS5_LM (24K rows) with low memory usage on the client-side so the significant variation in the evaluation measurements values for index 6, and index 7 in the following figures back to the dataset size and the client-side memory usage.

Figure 25 shows the scripting, rendering, and painting time in seconds for loading dataset process on the client-side.

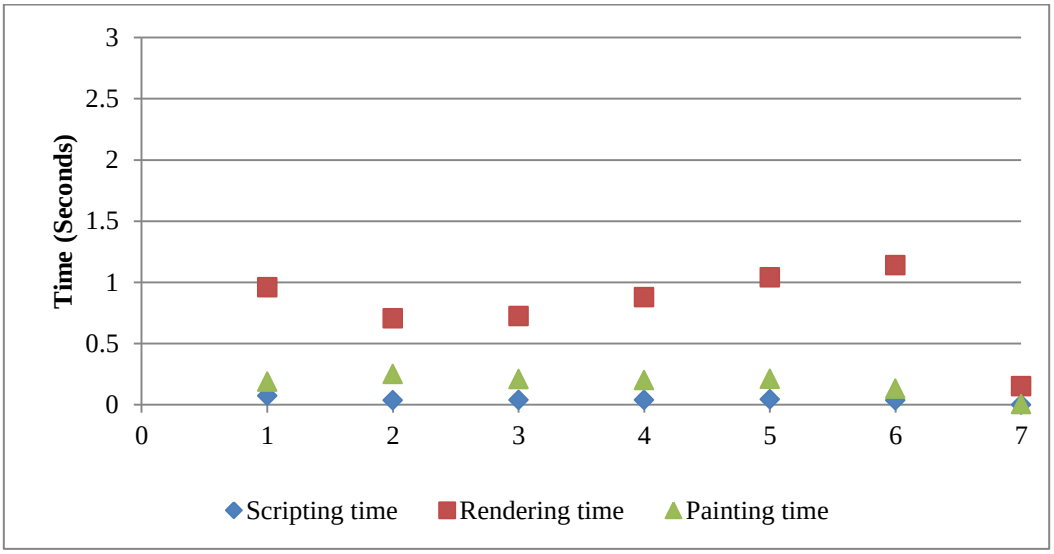

Figure 25: Dataset loading process times (client).

Figure 26 shows the scripting, rendering, and painting times in seconds for invoking line chart process on the client-side. The painting time for index 7 (DS5_LM) is goes up in comparison with index 6 (DS1_HM). This is mainly because of the line chart processing time delay on the server-side for large datasets. The client keeps the painting for the progress indicator component while waiting for line chart rendering on the server-side.

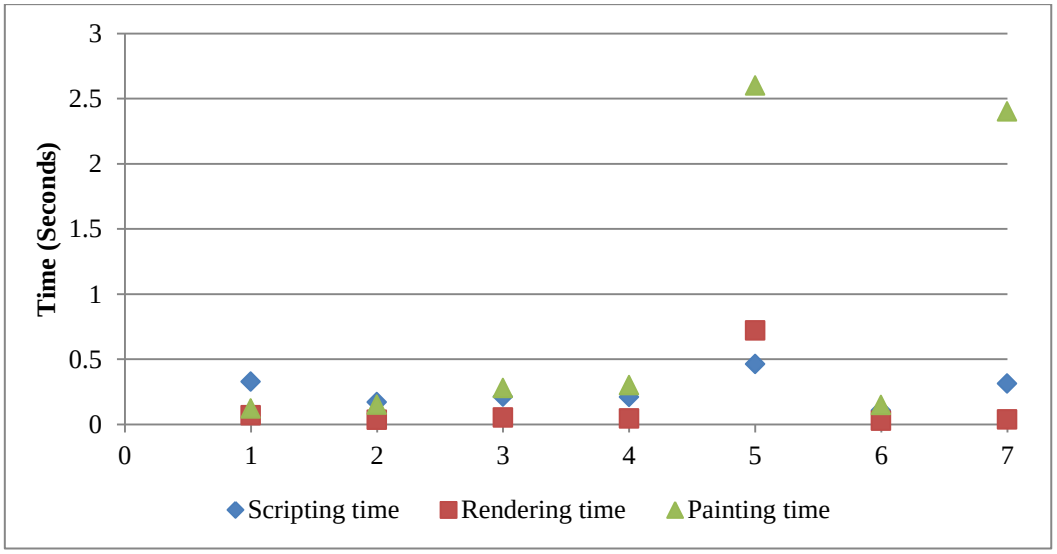

Figure 26: Line chart analysis process times (client).

Figure 27 shows the scripting, rendering, and painting time in seconds for invoking PCA analysis process on the client-side.

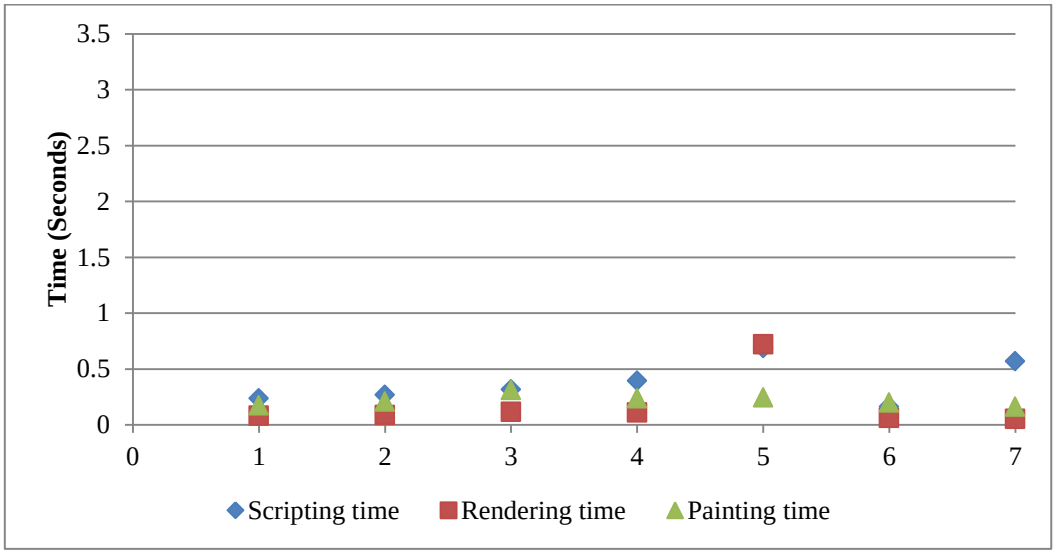

Figure 27: PCA analysis process times (client).

Figure 28: Ranking analysis process times (client). shows the scripting, rendering, and painting time in seconds for invoking ranking analysis process on the client-side.

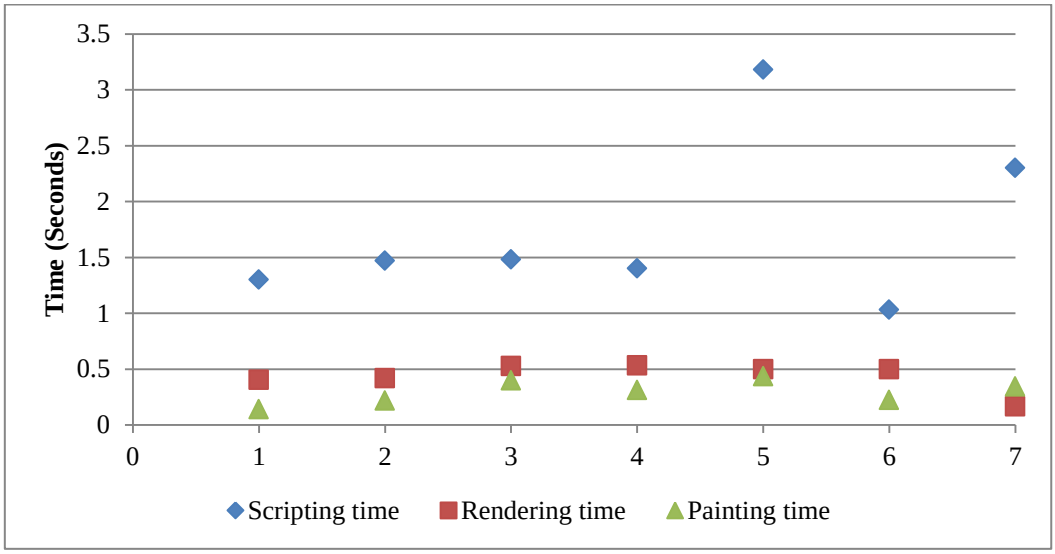

Figure 28: Ranking analysis process times (client).

Figure 29 shows the scripting, rendering, and painting time in seconds for selection update_i process (before invoking hierarchical clustering).process on the client-side.

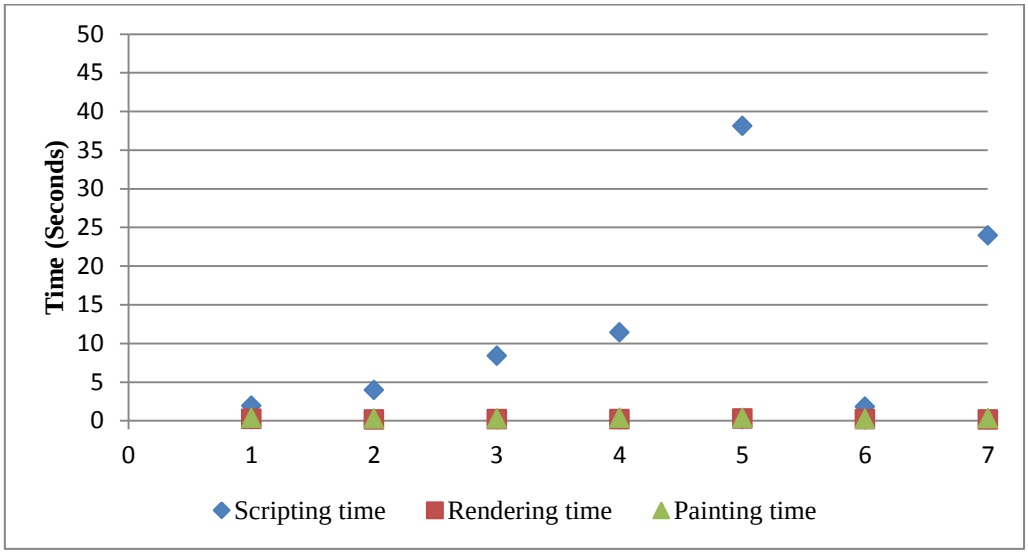

Figure 29: Selection update_i process times (client).

Figure 30 shows the scripting, rendering, and painting time in seconds for invoking hierarchical clustering analysis process on the client-side.

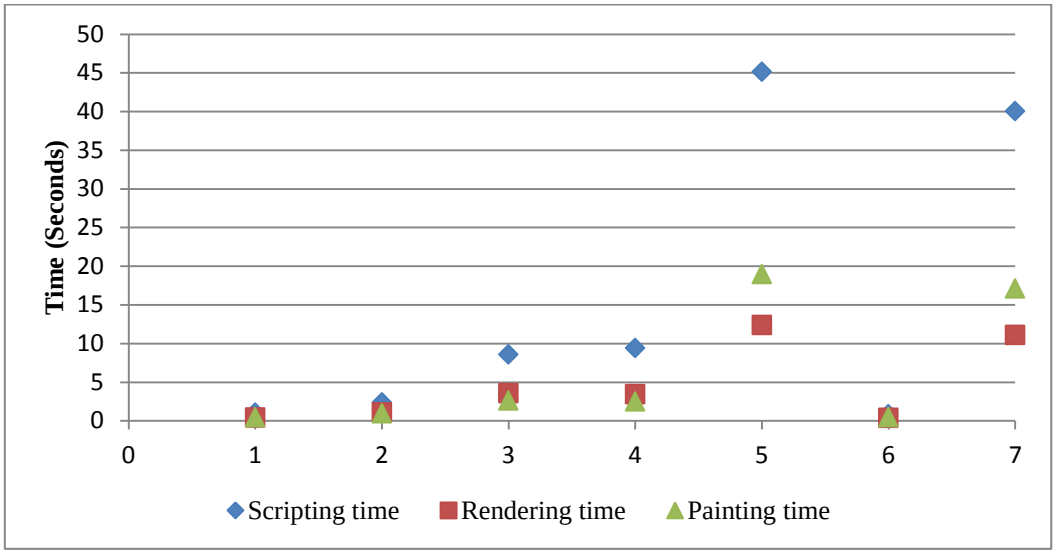

Figure 30: Hierarchical clustering process times (client).

Figure 31 shows the scripting, rendering, and painting time in seconds for selection update_ii process (after invoking hierarchical clustering) on the client-side.

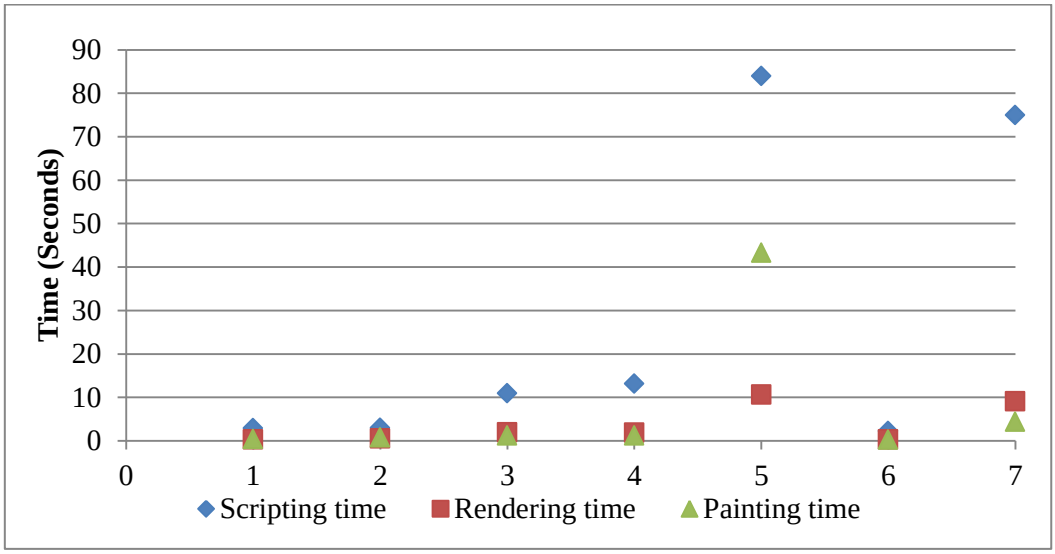

Figure 31: Selection update_ii process times (client).

7.4.2 Server-Side Evaluation

The server-side evaluation testing is divided into two main tests. The first one evaluates the memory usage and garbage collectors activity on the server-side and is performed under test Settings-1. The second test evaluates the server-side processing time as well as network transmission time. The second test is performed under test Settings-2.

a. Server-side Memory Usage Evaluation

Figure 32 shows the heap size (in pink) and the used heap (in purple) during performing the ordered processes (line chart analysis, PCA analysis, Rank Product analysis, Hierarchical Clustering analysis, and selection update).

(1) Loading DS1. (2) Loading DS2. (3) Loading DS3.
(4) Loading DS4. (5) Loading DS5. (6) Performing ranking analysis for DS5.
(7) Performing hierarchical clustering analysis and generating heat-map for DS5.

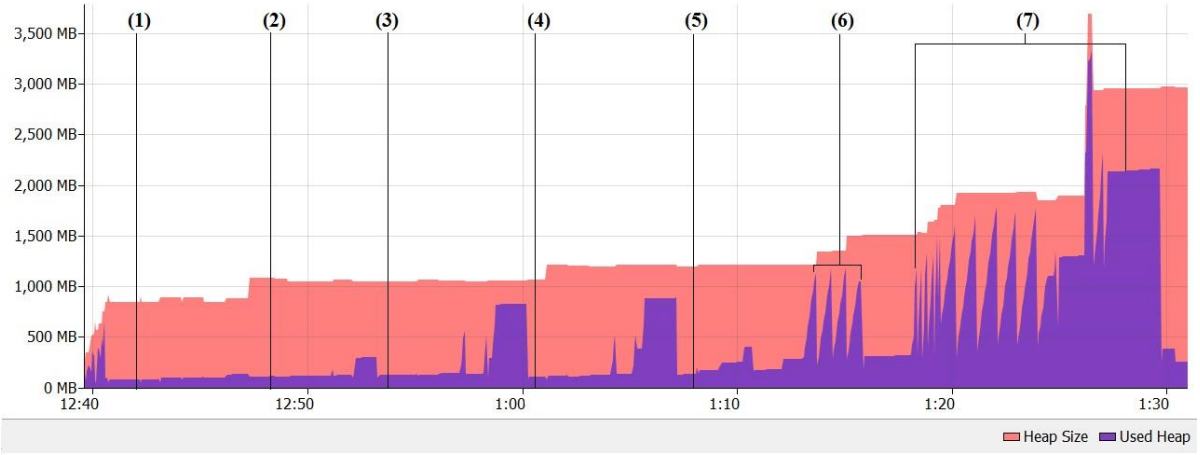

Figure 32: Heap size on server-side (local server) during testing process.

b. Server-side Processing and Network Transmission Time

The next figures represent the server-side processing time in seconds (the blue bars) and the total time in seconds for data processing on the server-side plus the network transmission time between the server and the client (the red bars).

Figure 33 shows the line chart analysis processing time on the server-side (online web server) and network transmission time for the available datasets.

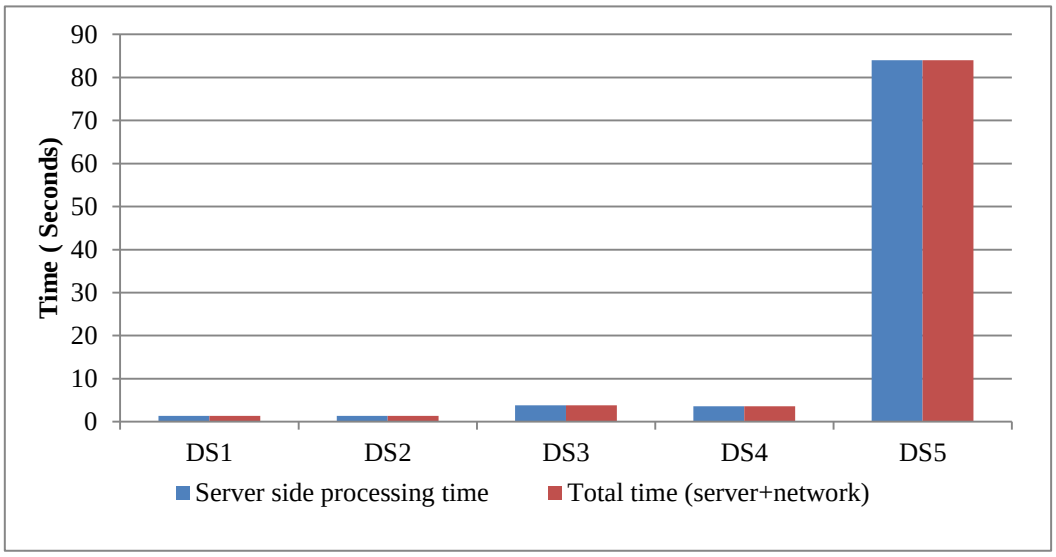

Figure 33: Line chart process (network and server-side latency).

Figure 34 shows the PCA analysis processing time on the server-side (online web server) and network transmission time for the available datasets.

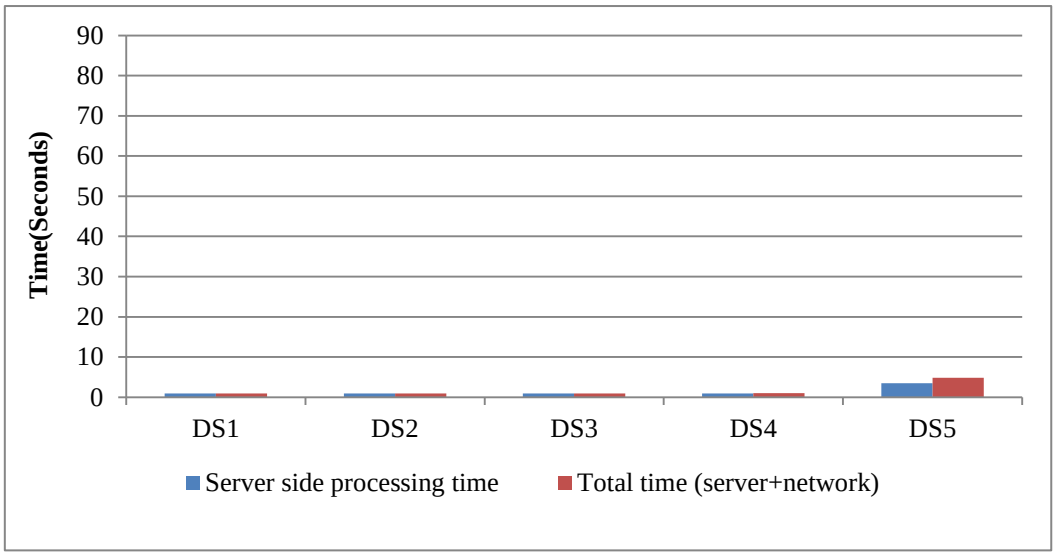

Figure 34: PCA analysis process (network and server-side latency).

Figure 35 shows the Ranking analysis processing time on the server-side (online web server) and network transmission time for the available datasets.

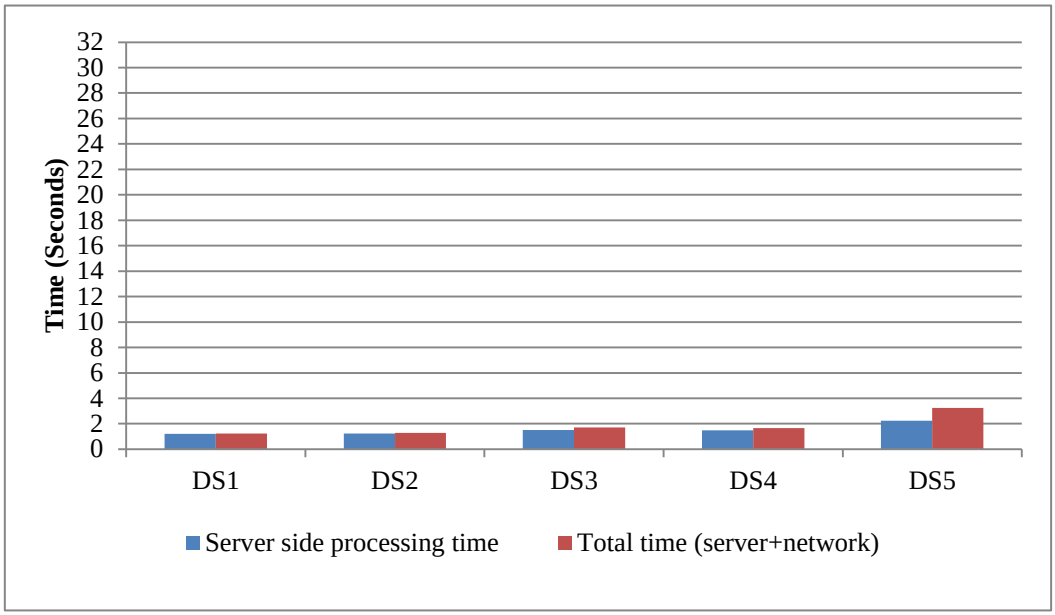

Figure 35: Ranking analysis process (network and server-side latency).

Figure 36 shows the Hierarchical Clustering analysis processing time on the server-side (online web server) and network transmission time for the available datasets.

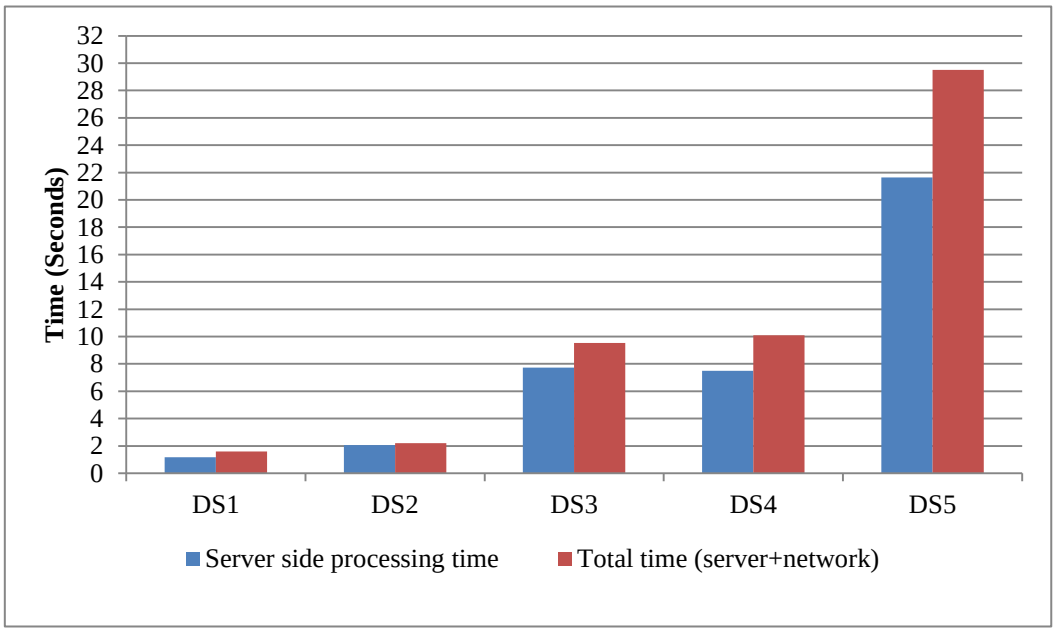

Figure 36: Hierarchical clustering analysis process (network and server-side latency).

Figure 37 shows the Selection update processing time on the server-side (online web server) and network transmission time for the available datasets.

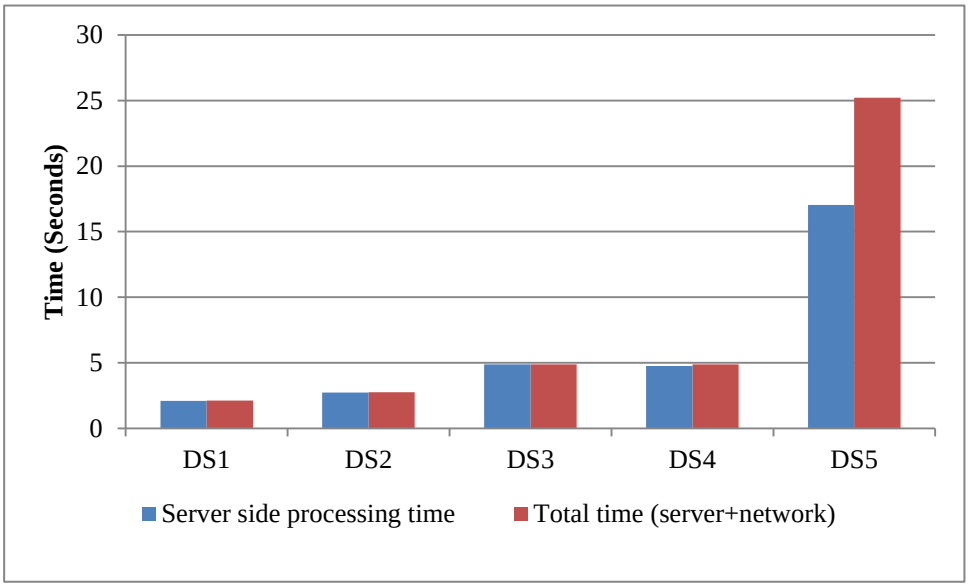

Figure 37: Selection update process (network and server-side latency).

Chapter 8

Discussion

8 Discussion

In this chapter we evaluate the system's performance, analyse results and discuss the effects of the proposed principles for developing distributed interactive visual analysis system of omics data based on the results presented in Chapter 7. In addition we also extract the current system limitations and show how these limitations would affect the performance.

8.1 Memory Management Evaluation

8.1.1 Client-side Memory Management

For browser-based distributed systems that deal with large scale data like in omics, the memory is one of the important performance bottlenecks. How to manage the client-side memory to keep it at a minimum level while dealing with large scale visualisation results is one of the main issues that arise during the development. In DIVA as a prototype for such systems, we applied different principles and methods to decrease the memory usage (see Chapter 6.5.1).

One of the major principles that help to control the memory in the client-side is to reduce the rendering process on the client-side as much as possible using partial rendered components or by moving rendering process to the server-side.

Figure 38 (reproduced from Figure 24) shows the client-side memory usage in megabytes.

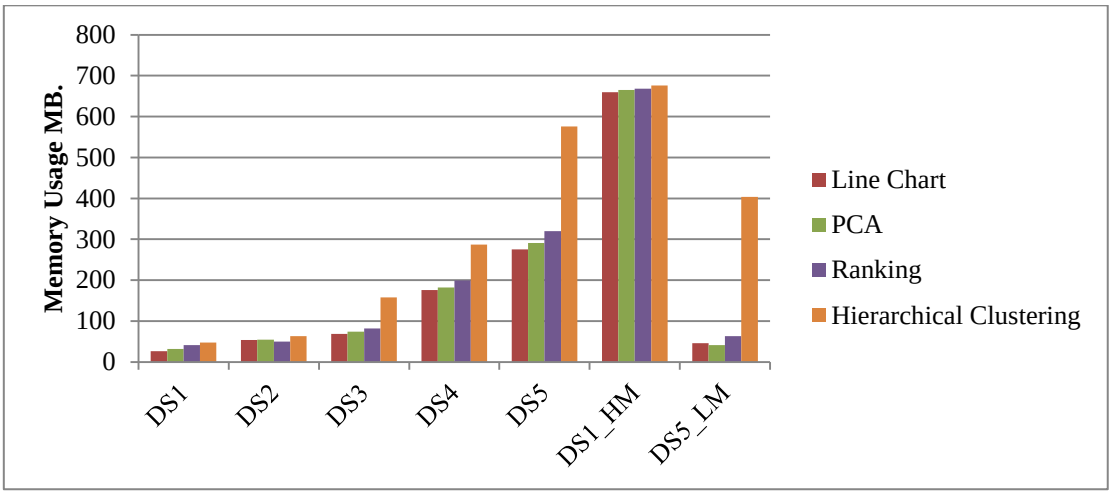

Figure 38: Memory usage on client-side.

From Figure 38 we can see the impact of distributing the rendering processes for line chart, PCA plot and ranking on the client-side memory usage. As we can see the three components did not result in any dramatic increasing of the memory usage even with the large dataset DS5.

The significant increase in the client-side memory usage happened mainly because of the hierarchical clustering component (the side tree) where the memory usage almost reached 700 MB (the V8 JavaScript engine memory limit). Although the component shows better performance when the client memory usage was low (DS5_LM) in comparison to its performance when the memory usage was high (DS5), it is still causing an unacceptable increase in the memory usage for large datasets.

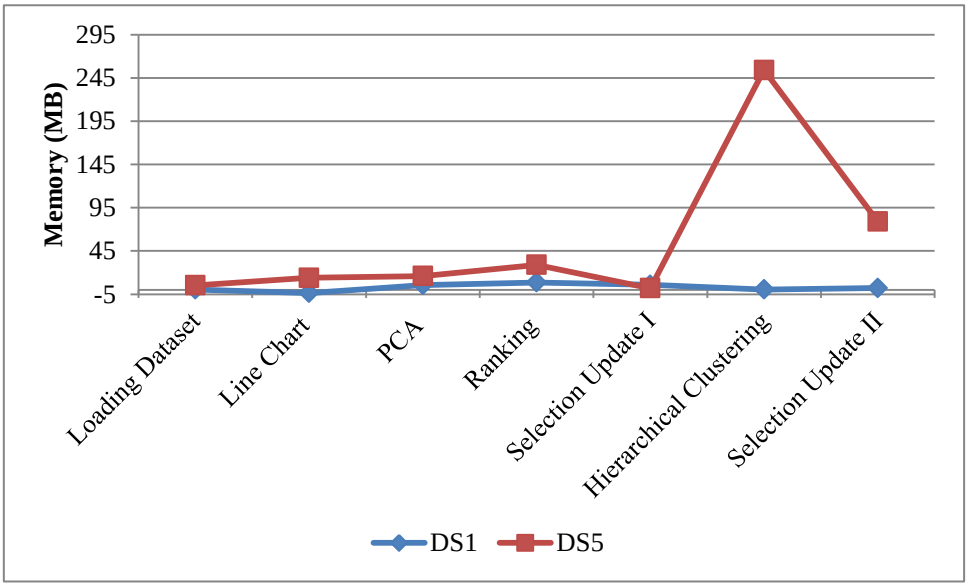

Figure 39: Memory behaviour for DS1and DS5.

Figure 39 shows the client-side memory behaviour when invoking different analysis processes for DS1 (blue line) and DS5 (red line). The memory behaviour (ups or down) indicates a very good memory management for loading datasets (omics information table), line chart, PCA, and ranking tables. The significant increase in memory happened when invoking the hierarchical clustering component on the large dataset DS5. The reason for this is the cline-side rendering process, where large number of nodes (24K+ clustering nodes) is added to the DOM tree and kept in the memory.

8.1.2 Server-Side Memory Management

Even though the memory is not generally as limited on the server-side as on the client-side, the memory management on the server-side is still important in order to enhance the entire system performance. Any drop in the server performance will directly affect the client-side. Also, as the server should allow a large number of users to perform different data analysis for different datasets efficiently, a good memory management on the server-side is required for such systems. In DIVA different techniques are used during the development to enhance the memory management on the server-side

Figure 40 (a repeat of Figure 32) shows the heap size (in pink) and the used heap (in purple) during performing the ordered processes (line chart analysis, PCA analysis, Rank Product analysis, Hierarchical Clustering analysis, and selection update).

(1) Loading DS1. (2) Loading DS2. (3) Loading DS3.
(4) Loading DS4. (5) Loading DS5. (6) Performing ranking analysis for DS5.
(7) Performing hierarchical clustering analysis and generating heat-map for DS5.

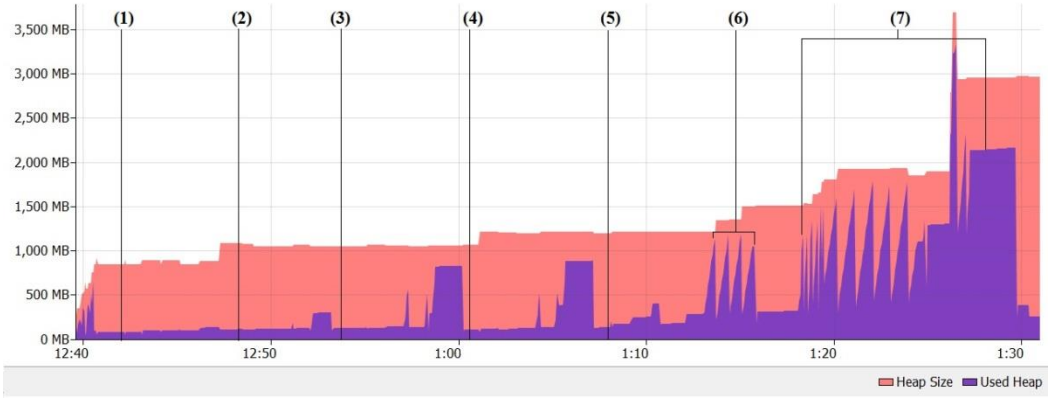

Figure 40: Heap size on server-side (local server) during testing process.

We can see that the used heap memory increases with the size of the data sets. With the relatively large datasets like DS5 we can see that there is a dramatic increase in the used heap size especially when performing hierarchical clustering analysis. This increase in the used heap size triggers the Garbage Collectors (GC) to manage the memory and reduces the used heap size.

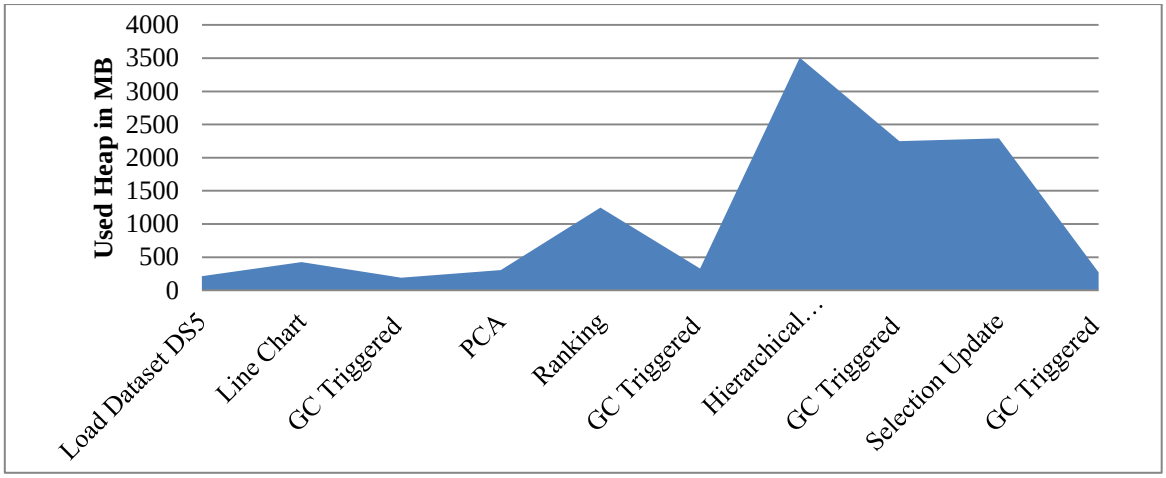

Figure 41: Used heap on server-side (local server) during DS5 analysis processes.

As shown in Figure 41 the activity of the GC on the server-side is very efficient, especially with large datasets like DS5. In DS5 we can see that the used heap for ranking process reaches 1250 MB which triggers the GC to reduce the used heap to 329 MB, also the hierarchical clustering process increased the used heap to reach 3507 MB. This significant increase in the used heap size triggered the GC to reduce the used heap size to 2249 MB. After this we can see a dramatic drop in the memory usage when the selection update process is invoked, where the process triggered the GC again to reduce the memory to 271 MB.

8.2 Network Communication Management Evaluation

Network communication management is critical for the success of distributed interactive visual analysis systems that deal with large scale omics data. The distribution of the different computational processes between the client and the server requires large number of network calls which need to be well managed otherwise it will cause a big performance hit. One of the examples that show how the good network management increases the feasibility of interactive visualisation for such systems is Figure 42 (a repeat of Figure 37) which shows the selection update processing time on the server-side (online web server) and network transmission time for the available datasets. We can see the efficiency of the network communications on transferring the PCA results over network.

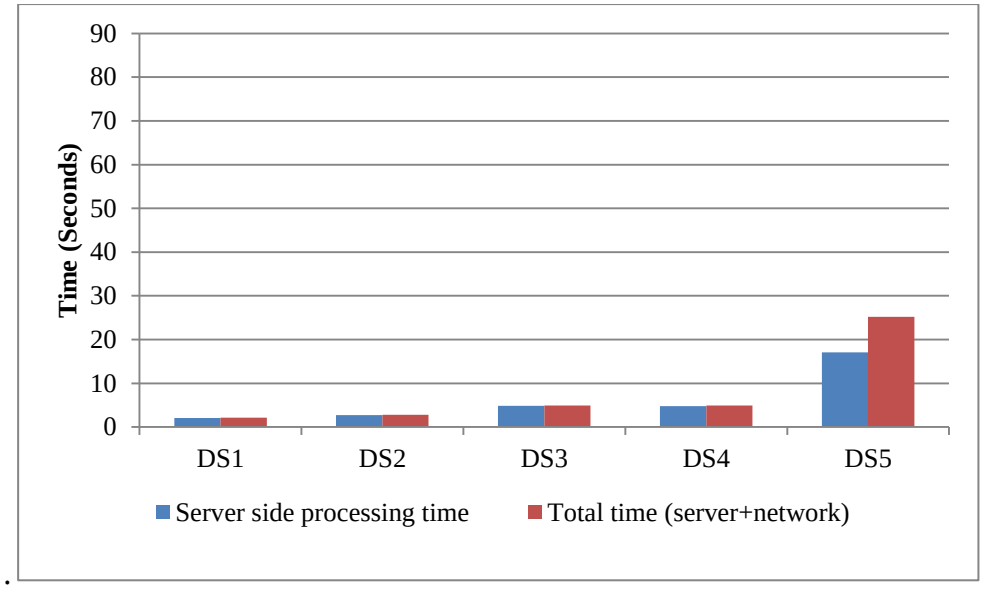

Figure 42: Selection update process (network and server-side latency).

There is an increase in the network transmission time when updating the selection for DS5 (all data selected in 24K dataset). This increase happens because of the size of data sent with the PCA plot.

Figure 43 (a repeat of Figure 33) shows the line chart analysis processing time on the server-side (online web server) and network transmission time for the available datasets. While there are long delay in server-side processing time for line chart (can be optimised in the future), the network transfer time is still very efficient.

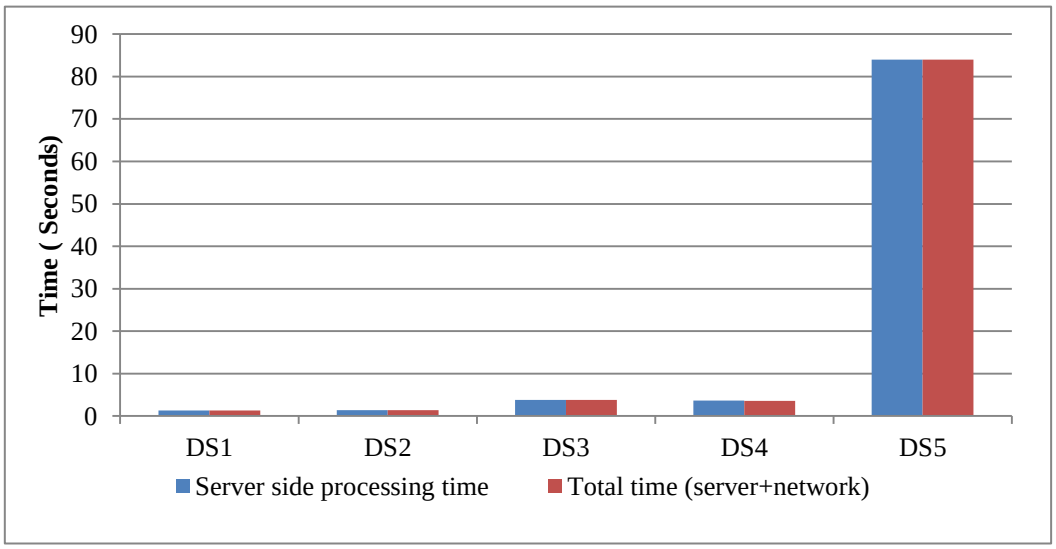

Figure 43: Line chart process (network and server-side latency).

Figure 44 (a repeat of Figure 36) shows the Hierarchical Clustering analysis processing time on the server-side (online web server) and network transmission time for the available datasets.

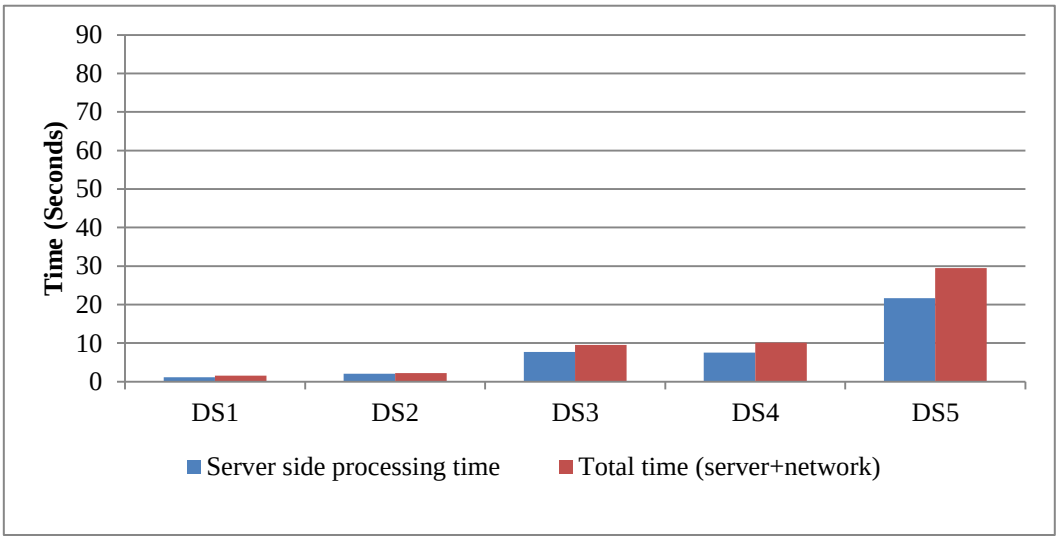

Figure 44: Hierarchical clustering analysis process (network and server-side latency).

From Figure 44 we can see long network transmission time especially with the largest dataset DS5. The reason for this increase is the size of the data sent over the network. The time is still acceptable and does not influence the overall system performance since the transmission is done once then the client-side will be the responsible side for updating (repaints) the component.

It is worth mentioning that GWT RPC played an essential role to manage the network calls between client and server in DIVA.

8.3 Interactive Visualisation Computational Processes

The distribution of the computational processes for visual interactive distributed systems is considered the second main challenge that affects the efficiency and visualisation interactivity feasibility. The balance between the workload on the limited client-side and number of network calls has to be optimised to keep the system performance. This section shows the impact of distributing computational processes required for handling the interactive visualisation on the overall performance for such systems.

8.3.1 Evaluation of Computational Process Required for Visualisation (Rendering)

Rendering large visualisation components is one of the heaviest processes on a browser-based client-side. For browser-based distributed omics analysis systems that support multiple interactive visualisations, client-side rendering could turn to be a very heavy process that affects the visualisation feasibility. To come over this challenge the rendering computational efforts have to be distributed between the client and server to achieve a balance between the workload on client-side and the network calls. To highlight the effect of this distribution, we evaluate the impact for distributing each of the visualisation components in DIVA using the results from the previous chapter.

The challenge was how to optimise the rendering processes and distribute them between the client and server. The main target is reducing the workload on the client-side to a reasonable limit while keeping the network traffic within the safe limit that does not affect the interactivity speed. The decision is made to draw table-based components and side/top trees for hierarchical clustering on the client-side and drawing the rest on the server-side.

Table-based components evaluation: the decision was made to draw the table-based components (omics data information table and ranking tables) on the client-side. This reduces the amount network traffic,

Figure 45 (reproduced from Figure 25) shows the scripting, rendering, and painting time in seconds for loading dataset process on the client-side and Figure 46 (reproduced from Figure 28) shows the scripting, rendering, and painting time in seconds for invoking ranking analysis process on the client-side.

Index 1 represents DS1 Index 2 represents DS2 Index 3 represents DS3
Index 4 represents DS4 Index 5 represents DS5

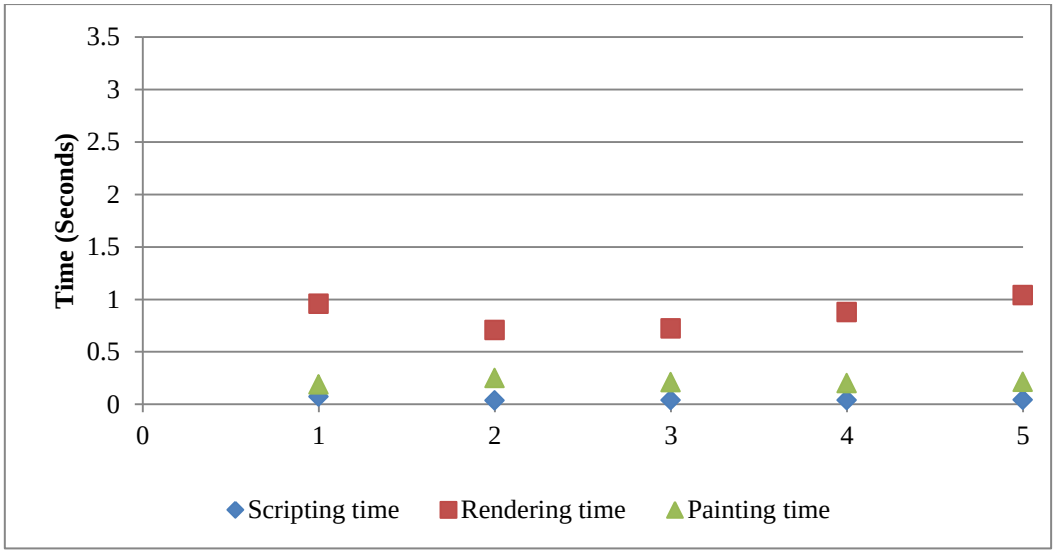

Figure 45: Dataset loading process times (client).

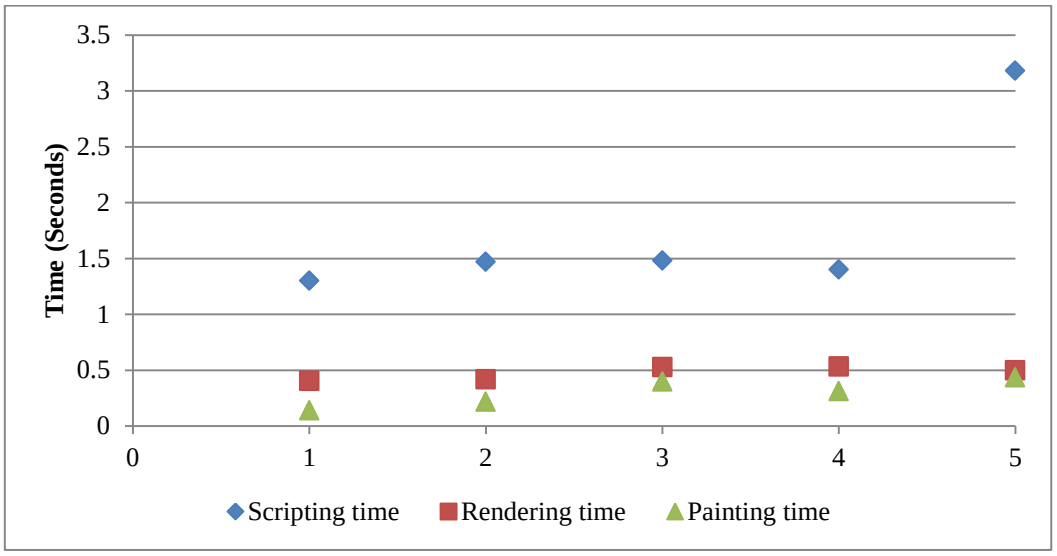

Figure 46: Ranking analysis process times (client).

As shown in Figure 45 and Figure 46 the system shows a very efficient rendering process for omics data information table, and ranking tables even with large dataset when using partial rendered visualisation components. It is clear that there is as increase in the scripting time for ranking tables with large datasets (DS5). This increase is expected and it happens because of the initialisation of the two ranking tables. On the other hand this increase is still acceptable.

Line Chart, Scatter plot, and Heat-Map rendering evaluation: the decision was made for the three components to be rendered on the server-side. Heat-map component was hard to be evaluated independently from the hierarchical clustering component since the heat-map is part of the component visualisation beside the side and top trees.

Figure 47 (reproduced from Figure 26) shows the scripting, rendering, and painting time in seconds for invoking line chart process on the client-side and Figure 48 (reproduced from Figure 27) shows the scripting, rendering, and painting time in seconds for invoking PCA analysis process on the client-side

Index 1 represents DS1 Index 2 represents DS2 Index 3 represents DS3
Index 4 represents DS4 Index 5 represents DS5

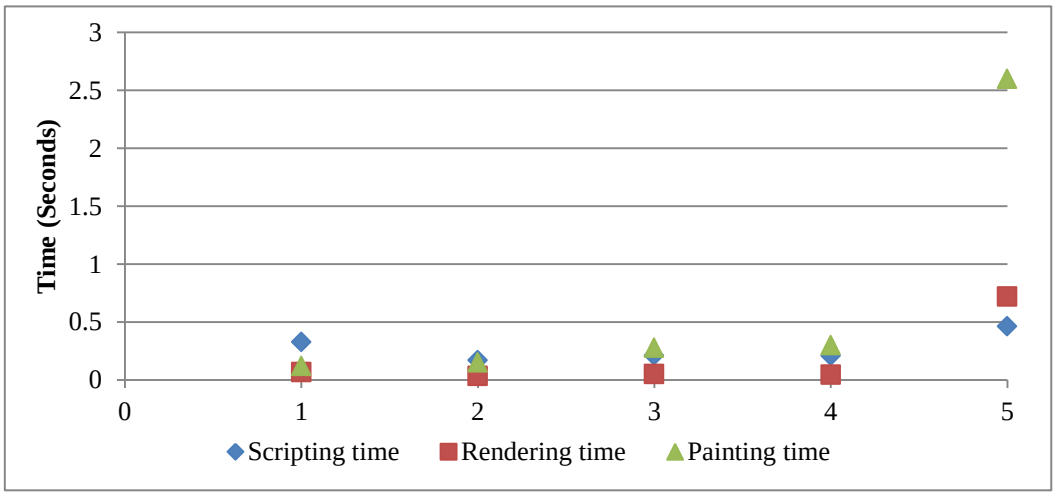

Figure 47: Line chart analysis process times (client).

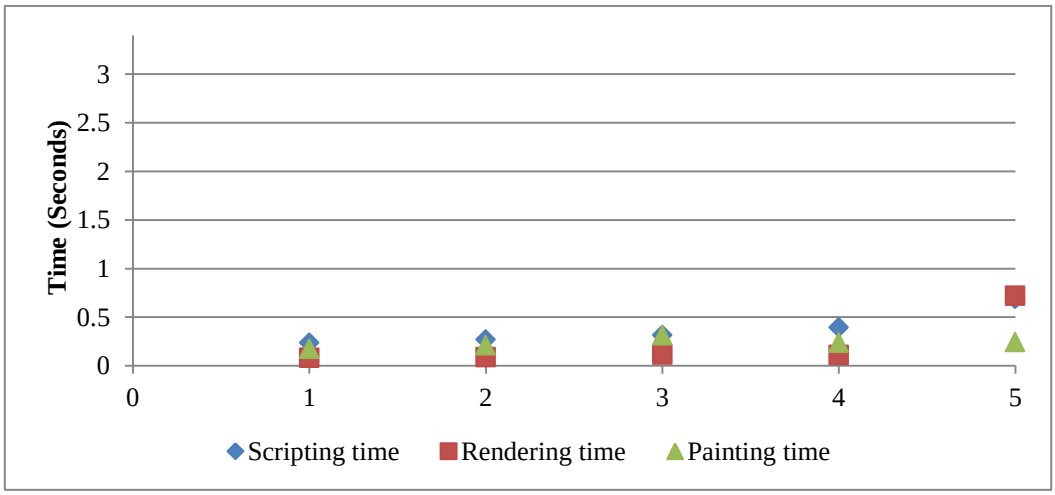

Figure 48: PCA analysis process times (client).

From Figure 47 Figure 48 we can see the reflection of server-side rendering on the client-side performance especially for the PCA plot. The rendering on server-side reduces the negative effect of the dataset size to a minimum. For line chart we can see there is still some delay in

painting on DS5. This delay happened because of the waiting time for server-side line chart painting (the progress indicator).

Hierarchical Clustering: Side/top trees with heat-map are used to represent the hierarchical clustering. The decision was made to draw the heat-map one the server-side and the side/top tree on the client-side.

Figure 49 (reproduced from Figure 30) shows the scripting, rendering, and painting time in seconds for invoking hierarchical clustering analysis process on the client-side.

Index 1 represents DS1 Index 2 represents DS2 Index 3 represents DS3
Index 4 represents DS4 Index 5 represents DS5

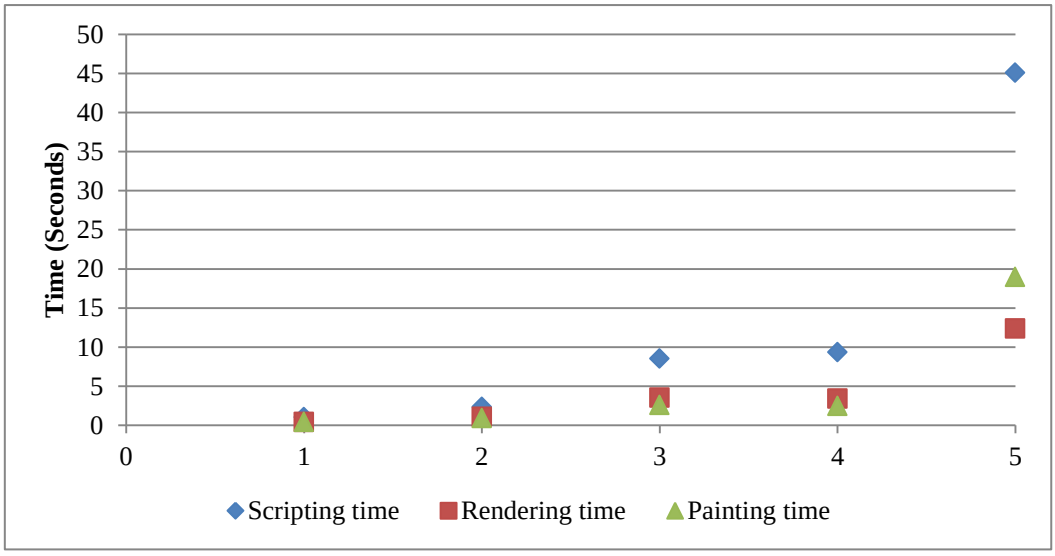

Figure 49: Hierarchical clustering process times (client).

As shown in Figure 49 the component shows good performance on small datasets DS1, and DS2 and acceptable performance slightly affected for average datasets (5K rows). The problem appears with relatively large datasets (DS5) where we can see big delay in scripting and painting time. Drawing this large amount of nodes and adding them to the DOM tree have great negative impact on the client performance. Therefore we strongly recommend moving the drawing of the side tree to the server-side. However, because of limitation of time, this is proposed as a future work.

From all of these processes and results, we can conclude that reducing the client-side rendering by distributing the process between client (using partial rendered components) and the server has a big positive impact on the visualisation feasibility of such systems.

8.3.2 Evaluation of Interactivity Computational Processes and Visualisation Update (Repaints) Distribution.

Computations required for maintaining the system interactive visualisation have a big impact on interactivity feasibility. Keeping all of these computations on the client-side can turn to be very heavy for the client computing capabilities especially with large scale data which affect the functionality feasibility. Offloading all of the computations to the server-side will significantly increase the networking calls and threatens the interactivity feasibility. These computations are very critical and need to be well distributed and well managed to keep the interactivity consistent (large number of functions executed in short time and distributed between the client and the server).

To show the impact of distributing such processes on the system, we are going to evaluate both the interactivity computational process and visualisation updates processes together for DIVA. Both processes are running together at the same time, and closely related. It is very hard to get them separated.

Figure 50 (reproduced from Figure 29) shows the scripting, rendering, and painting time in seconds for selection update_i process (before invoking hierarchical clustering) on the client-side.

Index 1 represents DS1 Index 2 represents DS2 Index 3 represents DS3
Index 4 represents DS4 Index 5 represents DS5

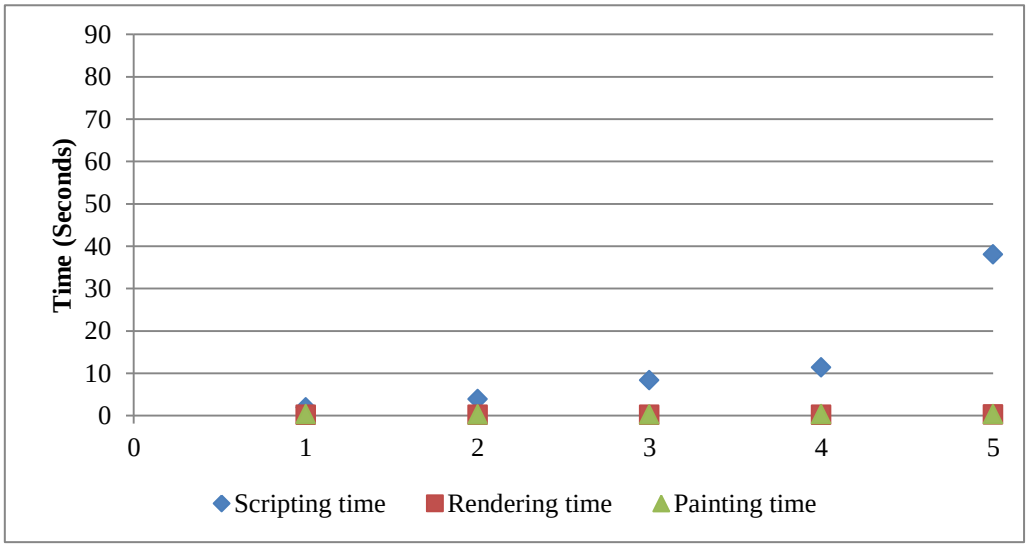

Figure 50: Selection update_i process times (client).

From Figure 50 we can see the efficiency of updating process before invoking the hierarchical clustering component especially the painting and rendering process. It is clear that the major delay occurs in scripting process. The larger is the dataset, the larger is the delay in scripting. As we can see for DS5 the scripting process has a large delay (38.1 seconds). The main reason for this delay is the interactivity computational processes (the selection, post-selection function (Selection-Manager), and pre-visualisation update functions). The selection is done in this component using PCA plot which means there will be pre-processing for omics data indexes (the x, y coordinates into omics data indexes) and this is done on the server-side. In the DS5 case, the selection was for all data (24K rows) which required time to be processed on the server-side. After that the Selection-Manager spent some seconds to coordinate the updates for each component (expected to not be long delay). In the pre-visualisation update functions, the client has to re-index the omics data selection for the positive and negative ranking tables on the server-side (24K x2) and send it back to the client. Then the scripting is required to repaints the line chart and PCA in the server-side. All of these factors lead to increasing the scripting time which is the maximum heavy selection performed before invoking hierarchical clustering. However, this is a rare situation as normally users will not select all data at once.

To show the effect of the hierarchical clustering component on the interactivity performance Figure 51 (reproduced from Figure 31) shows the scripting, rendering, and painting time in seconds for selection update_ii process (after invoking hierarchical clustering) on the client-side.

Index 1 represents DS1 Index 2 represents DS2 Index 3 represents DS3
Index 4 represents DS4 Index 5 represents DS5

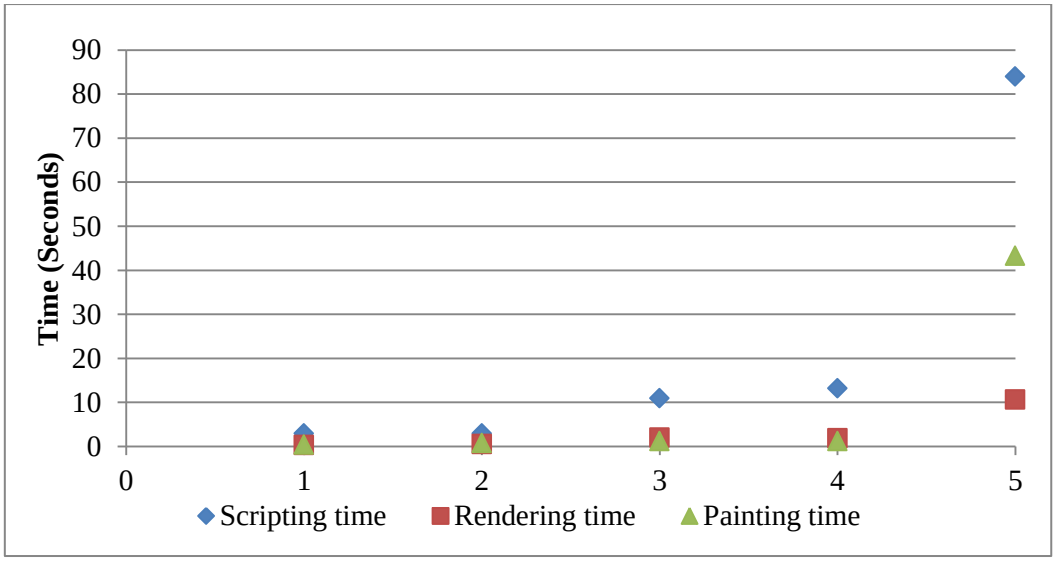

Figure 51: Selection update_ii process times (client).

As shown in Figure 51 there is a delay in scripting process in the different datasets. This delay significantly increases for DS5 (the largest dataset). Also the delay in painting and rendering processes significantly increases.

Figure 52 shows the impact of invoking hierarchical clustering component on the selection interactivity for DS5. The difference in delay time for scripting, rendering and painting process between 1 (DS5 selection update_i) and 2 (DS5 selection update_ii) shows the negative impact of drawing and repainting the side tree on the client-side.

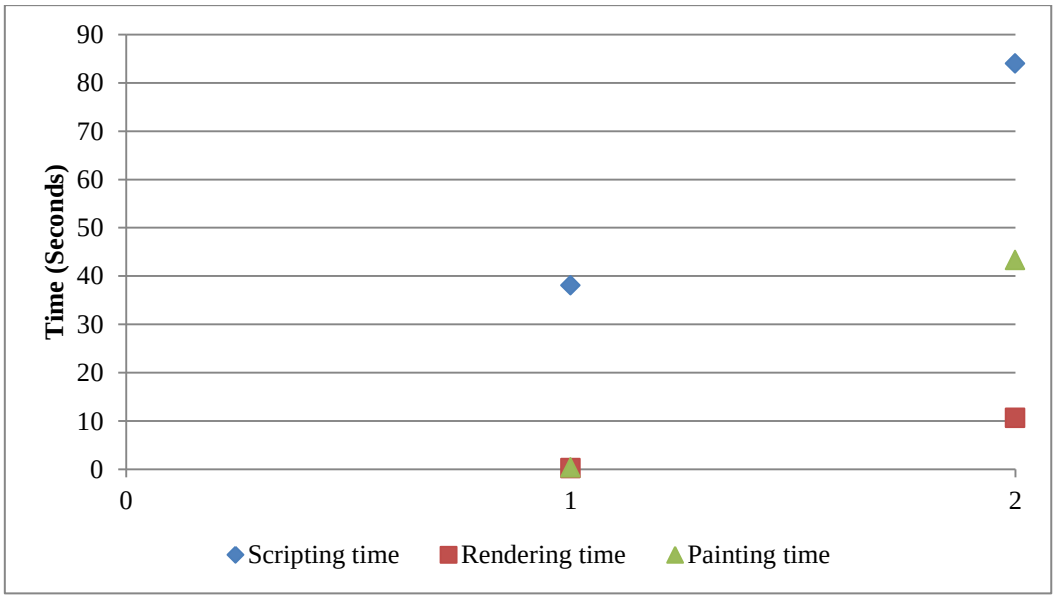

Figure 52:DS5 selection update_i and selection update_ii processes times (client).

8.4 Limitations

As a part of the main goal for this study, this section outlines the limitations of a distributed interactive visual analysis system of omics data. The limitations here refer to client-side, overall performance, and used technologies limitations. The client-side limitations are well outlined in the Chapter 6.5. Here we will focus on the overall performance and technology limitations.

8.4.1 Performance Limitations

There are many factors that can affect systems like DIVA and limit its performance. This section introduces some of these factors and shows how they impact the overall system performance.

a. Dataset and Selection size

One of the major factors that affect the system's performance is the dataset size. In such systems although many techniques can be used to reduce the effect of the dataset size on the performance, there are still some factors that cause delay in the interactive visualisation. For example there will always be a minimum amount of data has to stay on the client-side which we cannot move to server-side like the omics data indexes and other data required for client-side rendered components. Also some computations that have to be performed on the client-side are heavy for the client especially with very large data, and very large selections.

In addition to these general limitations, in the following, we outline other limitations of the developed DIVA prototype:

For line chart in DIVA, the drawing on the server-side can cause long delay in the response time for large selection on large datasets. The time spent at the server-side is still long in comparison to the PCA plot image generation.

For PCA plots we can see that the size of data transferred with PCA plot images increased with large scale datasets, which causes delay in the network transmission time. Also mapping the selection (x, y) coordinates to indexes in the PCA plot selection is still heavy for large selections on large datasets.

For ranking tables updating process, the delay caused by the re-indexing process required to map the omics data indexes to ranking values. This process turns out to be heavy when the user selects a large amount of data for large datasets (for example 24K indexes). The re-indexing process happened in the two tables at the same time (positive and negative ranking tables) which increased the load on the server and on the network.

For hierarchical clustering, the size has major effect on the component performance. The component (side tree) is drawn on the client-side. Increasing the size of the dataset leads to increasing the rendering and repaints processes on the limited client-side. Moreover, increasing the size also leads to increasing the memory usage on the client-side which has a negative effect on the client-side performance. Due to time constraints we were not able to optimise this component and decided to keep the existing method of drawing the side tree on the client-side. However we recommend moving this process to the server in future development, and generate it as an image. This should have a significant impact on the processing time and memory usage of the client-side.

b. Memory usage on the client-side

The second major factor that can impact the interactive visualisation performance is the memory usage size when invoking the analysis process. Long using session leads to increase the memory usage on the client-side. High client-side memory usage has negative impact on the interactive visualisation performance especially with large datasets.

Figure 53 shows the scripting, rendering, and painting time in seconds for selection update_i process (selection from PCA chart and updating omics data information table, line chart, PCA, and ranking tables only) for both DS5 (browser memory usage is 322 MB) and DS5_LM after cleaning up browser memory usage (browser memory usage is 48.3 MB).

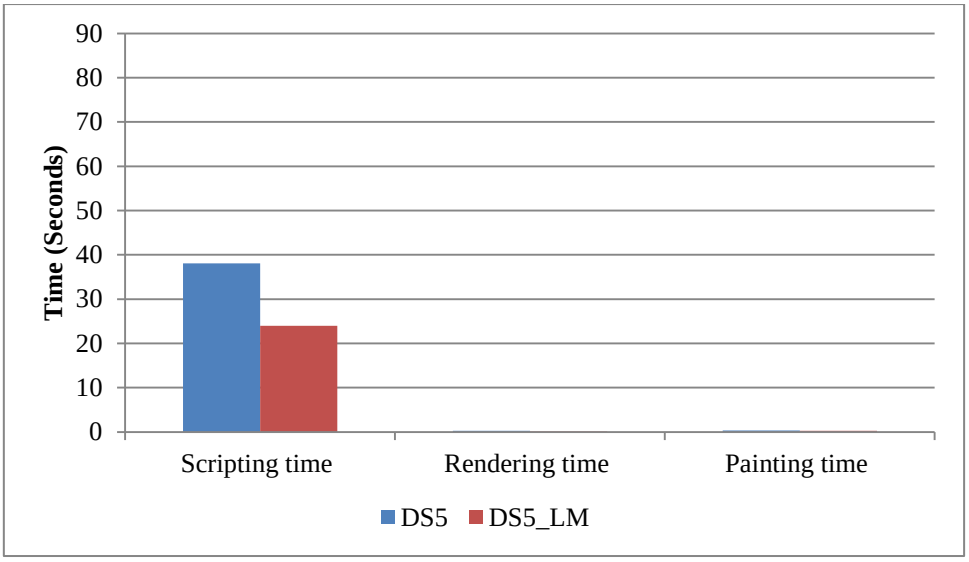

Figure 53: Selection update_i process times (client) for DS5 and DS5_LM

As we can see the reloading process reduces the scripting delay time from 38.1 second to 23.9 seconds, while the rendering and painting time remained almost the same since they already have a very good performance and most of rendering and repaints happening on the server-side. On the other hand cleaning up the memory has great impact on the performance after invoking the hierarchical clustering component where the load increases on the client-side. Figure 54 shows the scripting, rendering, and painting time in seconds for selection update_ii process (selection hierarchical clustering side tree and updating all the analysis visualisations) for both DS5 (browser memory usage is 655 MB) and DS5_LM after cleaning up browser memory usage (browser memory usage is 465 MB).

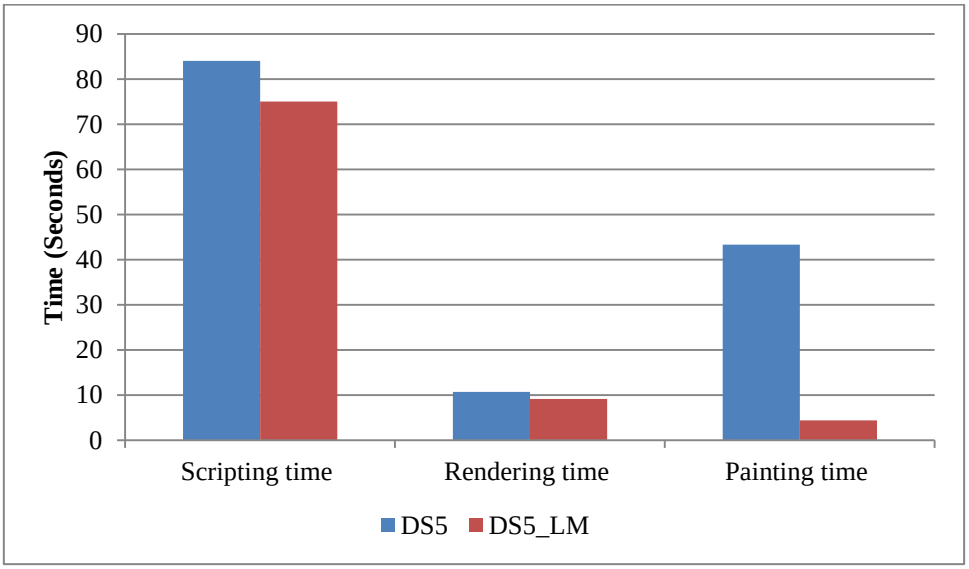

Figure 54: Selection update_ii process times (client) for DS5 and DS5_LM

From Figure 54 we can see there is improvement in all processes especially with painting process. We can see also that the improvement in scripting delay after cleaning the browser memory is less in selection update_ii than the case of selection update_i. The reason for that is when invoking the hierarchical clustering analysis the scripting delay is significantly increases. The selection function from the side tree is done on the client-side. Also the significant increase in the memory (from 48.3 MB before invoking the hierarchical clustering analysis to 465 MB after the invocation) has negative impact on the scripting process.

We recommend refreshing the session frequently while using the system. Reloading the web page from time to time is preferable to avoid the high memory usage caused by the long using session.

8.4.2 Technologies limitations

The use of modern technology plays a major role in this study. This section highlights the current limitations for the main technologies used during the development

ProtovisGWT Add-on: this technology allowed client-side rendering for side/top trees required for hierarchical clustering representation. Although the technology shows good performance on small scale datasets, the performance decreases significantly with large datasets. The technology itself does not have a problem with rendering the large scale datasets tree, the problem is the client-side rendering itself. We recommend developing a component using different server-side technologies to draw the side tree on the server-side as image and visualise it on the client-side which will have great impact on the overall performance.

JHeatChart: This technology is used to generate the heat-map graph on the server-side. The main limitation for this technology that it uses two colours range (minimum and maximum). We recommend replacing this generator with three colour range heat-map generator.

JFreeChart (Line chart generator): Although this technology shows a very good performance for PCA chart generator, there is still delay happening when generating images for line chart on the server-side especially with large scale datasets.

Figure 55 (a repeat of Figure 33) shows the line chart analysis processing time on the server-side (online web server) and network transmission time for the available datasets.

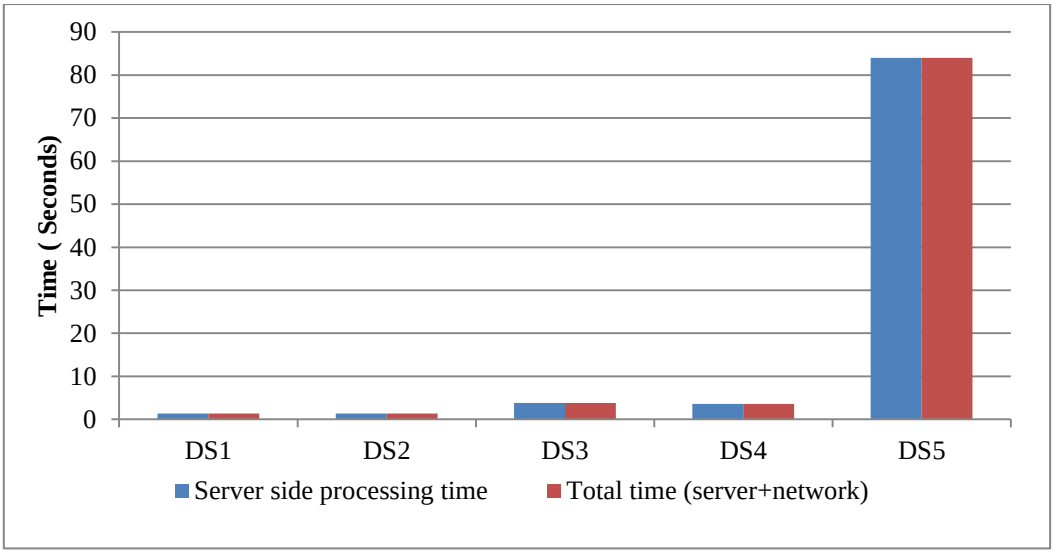

Figure 55: Line chart process (network and server-side latency).

As shown in Figure 55 the server needs almost 1.5 minutes to initialise, and process the line chart for DS5. The reason for the delay is that the current version of JFreeChart draw the chart line by line where each line is consider as an independent series. We recommended changing the line chart generator or performing more optimisations in future versions of DIVA.

Chapter 9

Conclusion

9 Conclusion

The purpose of this study is to investigate the feasibility of developing a distributed system for visualisation of omics data analysis that is interactive, scalable (i.e., able to handle large scale omics data) and biologist-friendly. It is motivated by the limitations of existing systems which lack one or more of these properties. To demonstrate the feasibility of such a system and outline its main limitations we developed a prototype system, Distributed Interactive Visual Analysis system of omics data, which attempts to fulfil all the above requirements.

In order to come over the limitation of being biologist-friendly, a pre-analytical decision is made to use a web application with a browser-based GUI that requires no installation and is easy to use with no programming skills.

The main challenge in a browser-based distributed system that supports multiple interactive visualisations of large-scale omics data becomes the distribution of data and computations across the client and server.

Offloading the whole data and the entire workload to the server increases the number of required network calls which limits the system's interactivity. On the hand keeping the data and entire workload on the client limits its scalability. Instead, a balanced distribution of both the data and the main computational processes across the system is necessary to avoid overloading the client and to keep the network calls number within the safe limit. To apply this in DIVA we moved the analysis process (which is performed using J-Express Modularised component) and as much as possible of the heavy processes required for maintaining both the interactivity and the visualisations to the server-side. However, we kept some computations on the client-side to reduce network traffic whenever it was feasible to do so.

The line chart, PCA, and heat-map graphs are drawn on the server-side and then sent as images to be displayed on the client-side. The heavy functions required to maintain the data selection interactivity is also offloaded to the server-side while the main Selection-Manager functions are kept on the client. Ranking tables, omics data information table and side/top trees for hierarchical clustering are drawn on the client-side. The evaluation of the prototype has shown that it successfully achieves good interactivity and good performance even with relatively large datasets (24K rows, 24 columns).

The results of the prototype are very promising. However, there are two main components in the current prototype that suffer from poor performance especially when processing large datasets: (i) the side tree on the hierarchical clustering component, and (ii) the line chart image generator. Due to time constraints we were unable to handle these issues. Our recommendations for future development are to optimise the line chart image generator, and to replace the side tree component by a server-side rendering component (in the same way as PCA and Line charts).

Based upon these findings, we can extract that there are two main limitations for distributed interactive visual analysis systems of omics data:

(i) The size of dataset and user-selection size. There will always be a minimum amount of data and computations to keep on the client-side otherwise the number of network calls will cross the safe limit and hit the performance. With a very large scale data, this amount of data and computation can turn to be very heavy for the browser capabilities.

(ii) The browser memory usage. High memory usage affects the system performance. Long session increases the memory usage on the client-side which affects the functionality feasibility. We recommend a page reloading from time to time to reduce the length of sessions.

From the study and the prototype evaluation, we conclude that even with the current limitations for the browser memory and computing capabilities, it is still feasible to develop a browser based distributed system for interactive visual analysis of omics data. This system will achieve good performance, scalability, besides being biologist-friendly.

.

References

[1] Wikipedia, “Central dogma of molecular biology,” Wikipedia, the free encyclopedia, [Online]. Available: www.en.wikipedia.org/wiki/Central_dogma_of_molecular_biology. [Accessed 26 04 2014].

[2] T. A.Hiwarkar, “SOFT COMPUTING METHODOLOGIES IN BIOINFORMATICS AND ITS ADVANCE TOWARDS BIOLOGICAL DNA,” *nternational Journal Of Engineering And Computer Science*, vol. 2, pp. 1506-1511, 5 May 2013.

[3] THANURA ELVITIGALA, ASHOKA POLPITIYA, WENXUE WANG,, “HIGH-THROUGHPUT BIOLOGICAL DATA ANALYSIS,” *IEEE CONTROL SYSTEMS MAGAZINE*, vol. 30, no. 6, pp. 81- 100, 2010.

[4] N.M. Luscombe, D. Greenbaum,M. Gerstein, “What is bioinformatics? An introduction and overview,” *International Medical Informatics Association Yearbook*, pp. 83-100, 2001.

[5] Karol Kozak, Aagya Agrawal, Nikolaus Machuy, Gabor Csucs, “Data Mining T echniques in High Content Screening: A Survey,” *Journal of Computer Science & Systems Biology*, vol. 2, 2009.

[6] Ahson, Tangirala Venkateswara Prasad, Syed Ismail, “Visualization of microarray gene expression data,” *Bioinformation*, vol. 1(4), p. 141–145, 2006.

[7] A. Basilevsky, in *Statistical Factor Analysis and Related Methods, Theory and and Applications*, New York, NY, John Wiley & Sons, 1994.

[8] Nils Gehlenborg, Seán I O’Donoghue, Nitin S Baliga,Alexander Goesmann,Matthew A Hibbs,Matthew A Hibbs,Oliver Kohlbacher,Heiko Neuweger,Reinhard Schneider, Dan Tenenbaum, Anne-Claude Gavin, “Visualization of omics data for systems biology,” *Nature Methods*, vol. 7, p. 56–68, 2010.

[9] Breitling R, Armengaud P, Amtmann A, Herzyk P, “Rank products:a simple, yet powerful, new method to detect differentially regulated genes inreplicated microarray experiments,” *FEBS Letter*, vol. 573, pp. 83-92, 2004.

[10] Jianguo Xia, Ngan H. Lyle, Matthew L. Mayer, Olga M. Pena, Robert E. W. Hancock, “INVEX—a web-based tool for integrative visualization of expression data,” *bioinformatics*, vol. 29, p. 3232–3234, 2013.

[11] Jonassen B., Dysvik I., “J-Express: exploring gene expression data using Java,” *Bioinformatics*, vol. 14, no. 4, pp. 369-370, 2001.

[12] Amit U. Sinha, Scott A. Armstrong, "iCanPlot: Visual Exploration of High-Throughput Omics Data Using Interactive Canvas Plotting," *PLoS ONE*, vol. 7, no. 2, 2012.

[13] Tutorialspoint, "GWT - RPC Communication," Tutorialspoint, [Online]. Available: www.tutorialspoint.com/gwt/gwt_rpc_communication.htm. [Accessed 05 05 2014].

[14] Java, Oracle Corporation, [Online]. Available: http://www.java.com/en/download/faq/java_mobile.xml. [Accessed 18 04 2014].

[15] "Java Servlet Technology Overview," Oracle Corporation, [Online]. Available: www.oracle.com/technetwork/java/overview-137084.html. [Accessed 16 04 2014].

[16] l. Grammel, "choosel modular webbased visualizations," Chisel Group, University of Victoria, 22 4 2011. [Online]. Available: http://www.slideshare.net/lgrammel/choosel-modular-webbased-visualizations?from=ss_embed. [Accessed 20 4 2014].

[17] I. Sommerville, "Software Engineering," 9th ed., Boston, Pearson Education, 2011, p. 33.

[18] I. Sommerville, Software Engineering, 8th ed., Addison-Weasley, 2007.

[19] S. Pasquali, "Mastering Node.js," in *Expert techniques for building fast servers and scalable, real-time network applications with minimal effort*, Birmingham, Packt Publishing, 2013, p. 16.

[20] S. Stefanov, "Rendering: repaint, reflow/relayout, restyle," 17 12 2009. [Online]. Available: http://www.phpied.com/rendering-repaint-reflowrelayout-restyle/. [Accessed 09 05 2014].

Appendix A

a. Start Up System

On system start up the system will start by reading all the dataset text files and convert them into Diva Dataset serialised objects stored in the file system for faster future access.

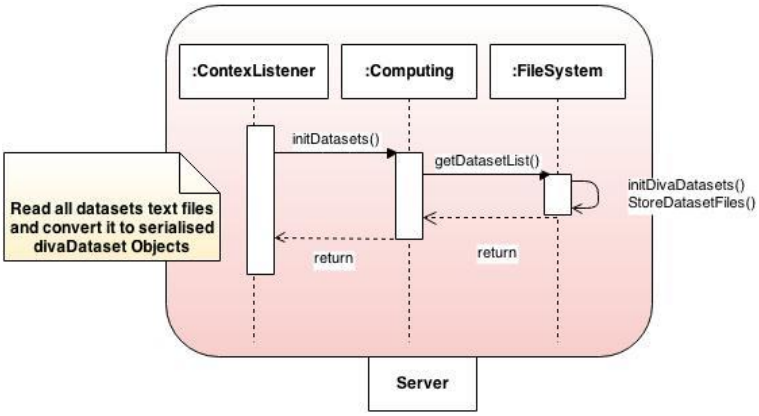

Figure 56: OSD start up system.

b. Load Web-Page (Start Session)

When the users open the DIVA web page a session is started and the client module is loaded and the list of available dataset is provided.

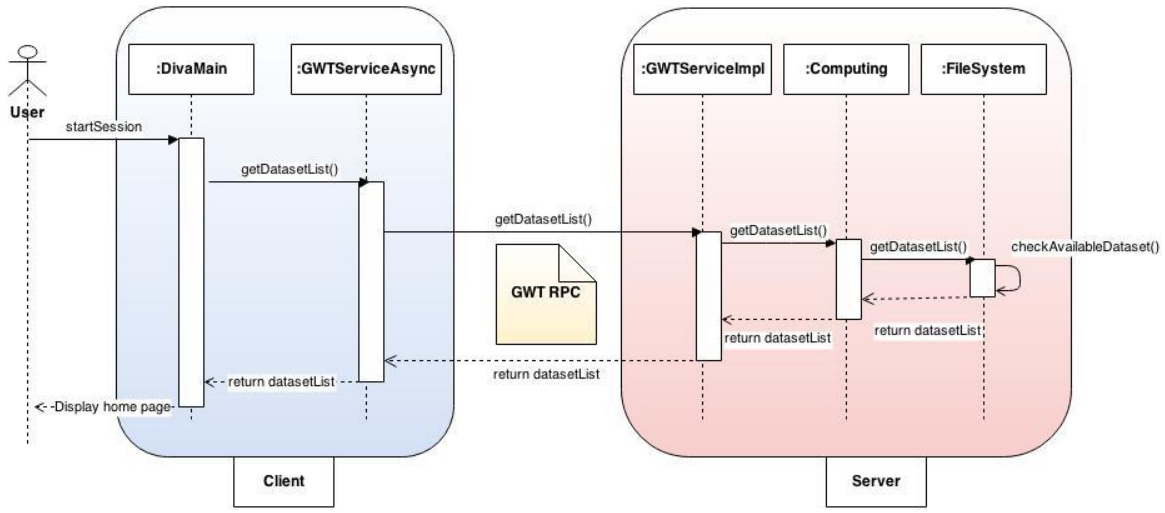

Figure 57: OSD start session.

:

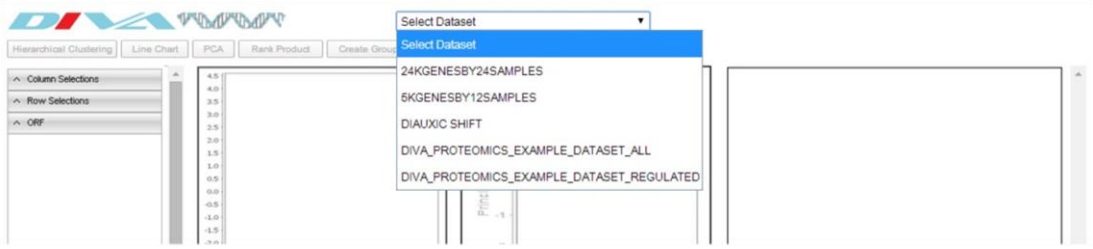

Figure 58: DIVA default web- page.

c. Select Dataset (Load Dataset)

When user selects a dataset from the drop down menu, the system loads the selected Diva Dataset on the server-side and sends a Dataset Info object to the client-side to update the dataset information and omics table visualisation.

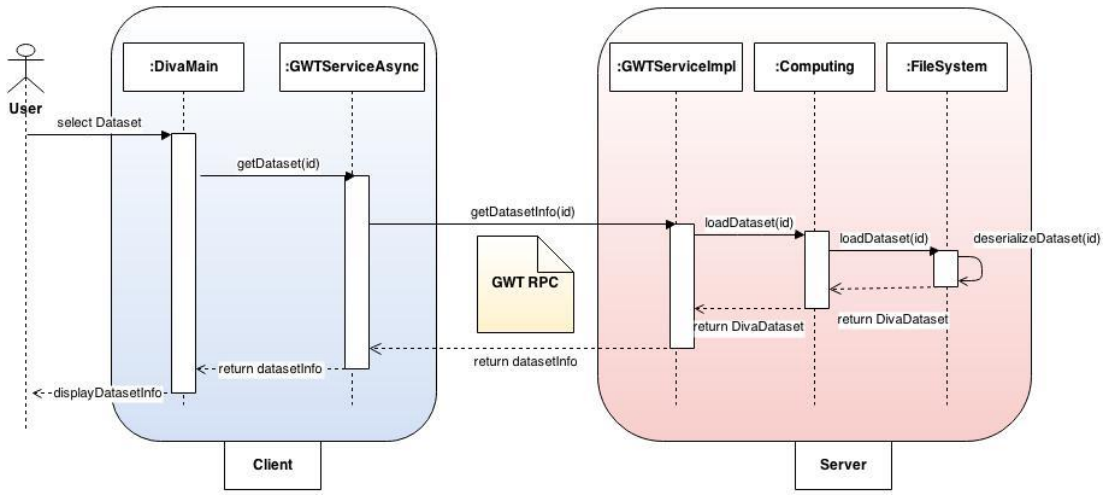

Figure 59: OSD load dataset.

d. Update Dataset Information

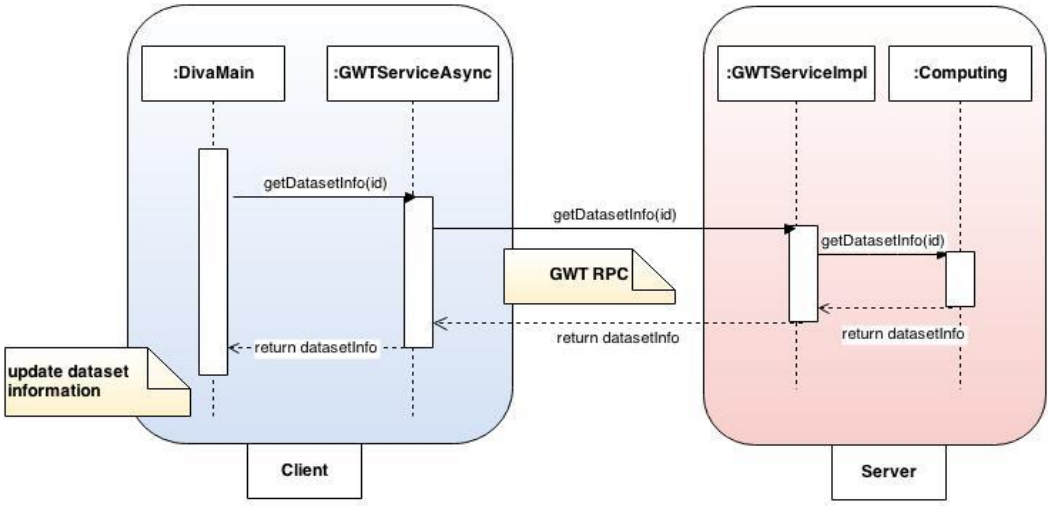

Figure 60: OSD update dataset information.

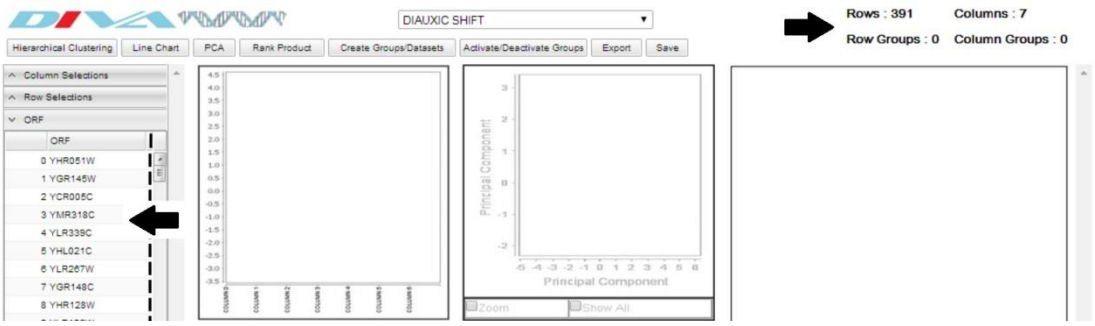

Figure 61: Update dataset information and omics information table.

e. Create Group

When the user wants to create a new group the system starts with checking the current rows and columns selection and provides the user with a panel to select the group type (row or column) and group colour, in addition to prompt the user to enter the group name. Then the system automatically updates the dataset information and the current omics table in addition to highlighting the group members with the new group colour in the current profile plot, or PCA plot

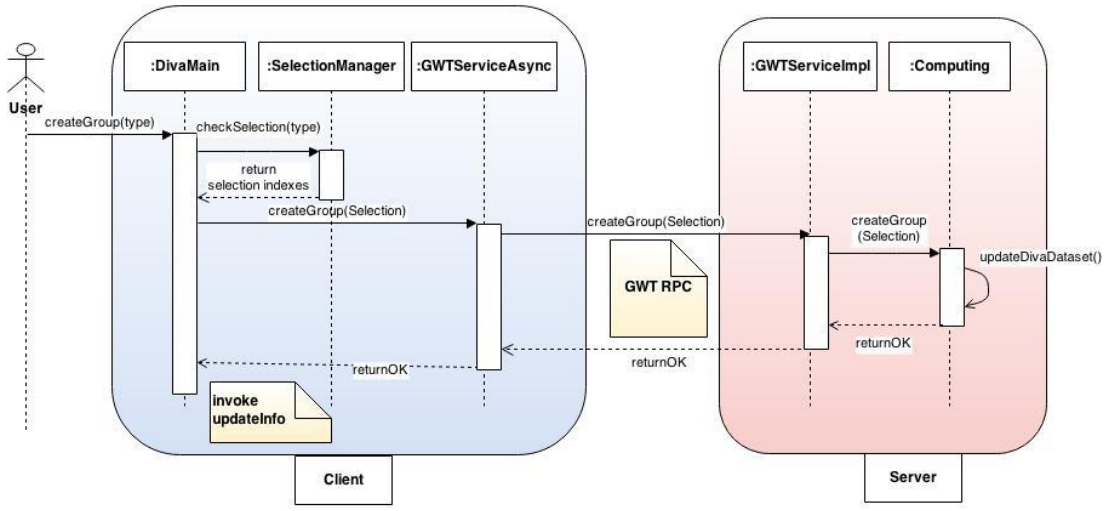

Figure 62: OSD create group.

f. Create Sub-Dataset

When user wants to create a sub-dataset the system starts with checking the current rows selection. If the selected rows number greater than zero the system activates the sub-dataset panel and prompts the user to enter the new sub-dataset name. Then the system stores the sub-dataset into the file system as a new dataset available to share with other users. On the successful sub-set storing DIVA automatically reloads the web-page providing the user with the new updated dataset lists.

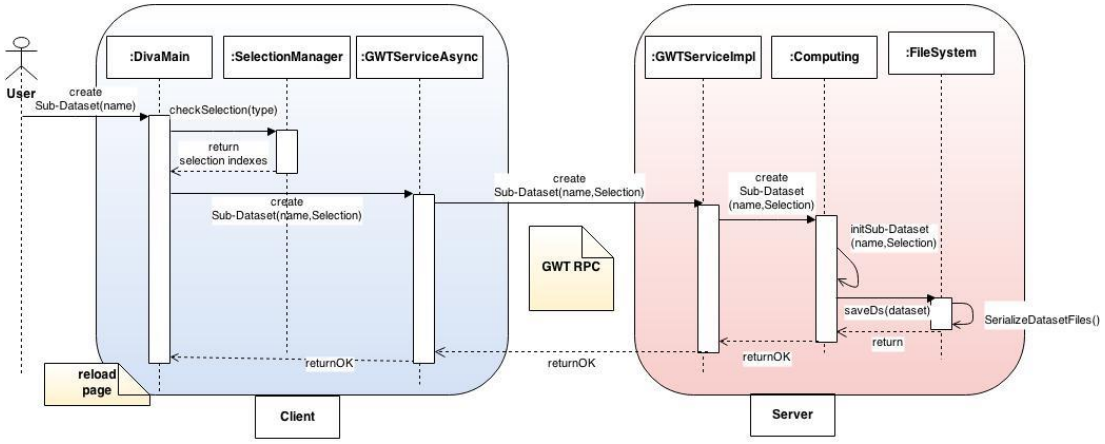

Figure 63: OSD create sub-dataset.

g. Save Dataset

When user wants to save the current dataset, the system prompts the user to enter the new dataset name. Then the system saves it with the current created groups in the file system as a new dataset available to share with the other users. On successful saving the system automatically reloads DIVA web-page providing the user with the new updated dataset lists.

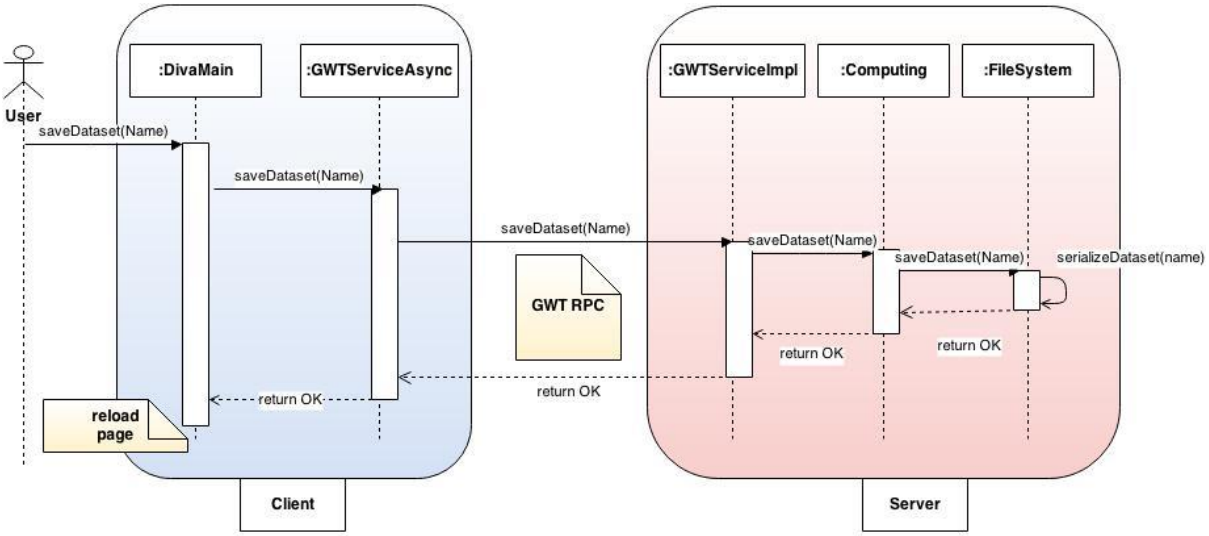

Figure 64: OSD save dataset.

h. Activate Group

When user wants to activate a selected row group, the system provides the user with an activate group panel that has all the available row groups to select, the system currently allow at most two row groups to be active at the same time. On successful activation for the group the system updates the current omics table in addition to highlighting the group members with the group colour in any available visualisation component (profile plot, and PCA plot).

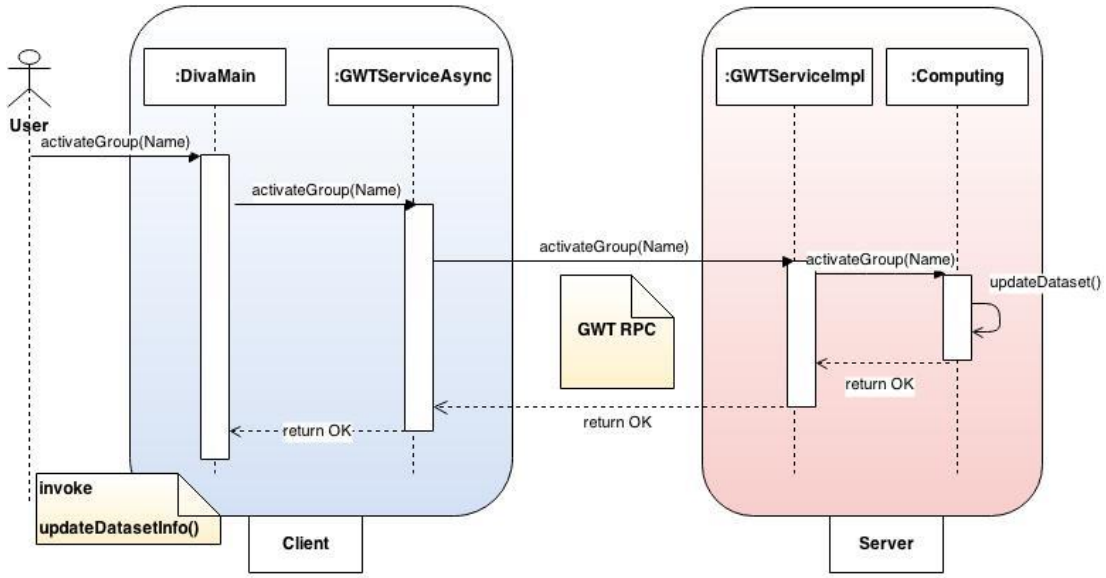

Figure 65: OSD activate group.

i. Export Dataset

When user wants to export data the system provides the user with the available row groups to export, This technique allow the user to export full dataset or just a part of it. The system creates the text dataset file on the server-side as available to download resource, and then forwards the file download link (URL) to the client-side and automatically download the file.

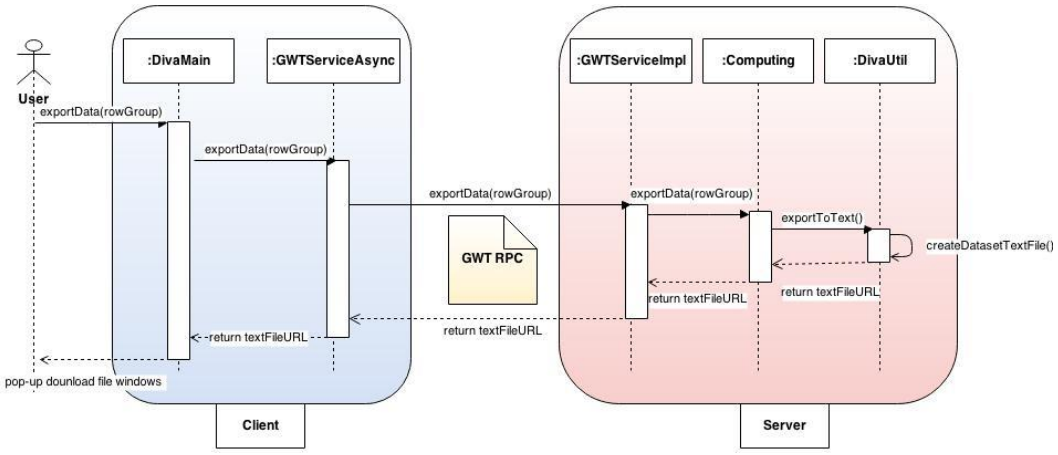

Figure 66: OSD export dataset.

.

Appendix B

Server-side Memory Usage Evaluation Results (Local Server)

Process	Heap size	Used Heap
Start-up system and Start session	795 MB	505 MB
Load Dataset DS1	**891 MB**	**96 MB**
Invoke Line Chart	945 MB	114 MB
Invoke PCA	940 MB	113 MB
Invoke Ranking	940 MB	119 MB
Invoke Hierarchical Clustering	933 MB	147 MB
Selection Update	1140 MB	149 MB
GC Triggered	1140 MB	125 MB
Load Dataset DS2	**1136 MB**	**124 MB**
Invoke Line Chart	1106 MB	128 MB
Invoke PCA	1104 MB	134 MB
Invoke Ranking	1104 MB	254 MB
GC Triggered	1127 MB	131 MB
Invoke Hierarchical Clustering	1108 MB	316 MB
Selection Update	1108 MB	320 MB
GC Triggered	1104 MB	138 MB
Load Dataset DS3	**1104 MB**	**140 MB**
Invoke Line Chart	1104 MB	156 MB
Invoke PCA	1119 MB	159 MB
Invoke Ranking	1119 MB	586 MB
GC Triggered	1108 MB	146 MB
Invoke Hierarchical Clustering	1114 MB	870 MB
Selection Update	1114 MB	876 MB
GC Triggered	1125 MB	117 MB
Load Dataset DS4	**1279 MB**	**125 MB**
Invoke Line Chart	1279 MB	141 MB
Invoke PCA	1268 MB	137 MB
Invoke Ranking	1263 MB	546 MB
GC Triggered	1277 MB	145 MB
Invoke Hierarchical Clustering	1277 MB	929 MB
Selection Update	1277 MB	938 MB
GC Triggered	1264 MB	143 MB
Load Dataset DS5	**1264 MB**	**214 MB**
Invoke Line Chart	1282 MB	426 MB
GC Triggered	1282 MB	191 MB
Invoke PCA	1282 MB	306 MB
Invoke Ranking	1422MB	1247 MB
GC Triggered	1592 MB	329 MB
Invoke Hierarchical Clustering	3875 MB	3507 MB
GC Triggered	3150 MB	2249 MB
Selection Update	3109 MB	2291 MB
GC Triggered	3125 MB	271 MB

Table 2: Server-side memory usage evaluation results.

*MB is megabytes

Server-side Performance Evaluation Results

Process	Server-side time Second	Total time (server + network)
Load Dataset DS1	0.616	0.618
Invoke Line Chart	1.32	1.33
Invoke PCA	0.914	0.916
Invoke Ranking	1.21	1.23
Invoke Hierarchical Clustering	1.159	1.59
Selection Update	2.1	2.14
Load Dataset DS2	0.018	0.009
Invoke Line Chart	1.37	1.38
Invoke PCA	0.912	0.914
Invoke Ranking	1.22	1.28
Invoke Hierarchical Clustering	2.065	2.201
Selection Update	2.73	2.77
Load Dataset DS3	4.32	4.43
Invoke Line Chart	3.82	3.82
Invoke PCA	0.924	0.933
Invoke Ranking	1.5	1.71
Invoke Hierarchical Clustering	7.73	9.52
Selection Update	4.875	4.895
Load Dataset DS4	1.43	1.55
Invoke Line Chart	3.62	3.61
Invoke PCA	0.945	0.984
Invoke Ranking	1.49	1.65
Invoke Hierarchical Clustering	7.5	10.1
Selection Update	4.76	4.89
Load Dataset DS5	7.69	7.92
Invoke Line Chart	84	84
Invoke PCA	3.51	4.8
Invoke Ranking	2.23	3.25
Invoke Hierarchical Clustering	21.63	29.5
Selection Update	17.04	25.22

Table 3: Server-side performance evaluation results.

Client-side Performance Evaluation Results

		DS1	DS2	DS3	DS4	DS5	DS1_HM	DS5_LM
Loading Dataset	Memory (MB)	30.9	51.1	68.7	177	261	657	31.9
	Loading time (Second)	0.072	0.036	0.039	0.039	0.043	0.036	0.000
	Scripting time (Second)	0.958	0.706	0.723	0.878	1.04	1.14	0.150
	Rendering time Second)	0.189	0.251	0.210	0.2	0.212	0.131	0.005
	Painting time (Second)	0.268	0.419	0.634	1.19	1.17	0.324	0.2
	Others time (Second)	0.183	0.370	0.165	0.238	0.363	0.628	0.162
Line Chart Analysis Process		**DS1**	**DS2**	**DS3**	**DS4**	**DS5**	**DS1_HM**	**DS5_LM**
	Memory (MB)	26.5	53.3	68.3	176	275	660	45.5
	Loading time (Second)	0.009	0	0.005	0.000	0.11	0	0.003
	Scripting time (Second)	0.329	0.17	0.213	0.210	0.463	0.107	0.313
	Rendering time Second)	0.069	0.036	0.052	0.045	0.72	0.028	0.039
	Painting time (Second)	0.121	0.151	0.279	0.3	2.6	0.149	2.4
	Others time (Second)	0.106	0.17	0.169	0.181	1.25	0.145	0.591
PCA Analysis Process		**DS1**	**DS2**	**DS3**	**DS4**	**DS5**	**DS1_HM**	**DS5_LM**
	Memory (MB)	32.1	54.7	74.3	182	291	665	40.9
	Loading time (Second)	0.002	0.002	0.602	0.002	0.002	0.002	0.001
	Scripting time (Second)	0.237	0.269	0.317	0.395	0.686	0.162	0.568
	Rendering time Second)	0.081	0.088	0.115	0.110	0.72	0.063	0.052
	Painting time (Second)	0.175	0.209	0.312	0.236	0.244	0.198	0.163
	Others time (Second)	0.121	0.114	0.170	0.126	0.175	0.117	0.141
Ranking Analysis Process		**DS1**	**DS2**	**DS3**	**DS4**	**DS5**	**DS1_HM**	**DS5_LM**
	Memory (MB)	40.8	49.9	81.9	199	320	668	62.9
	Loading time (Second)	0.136	0.144	0.174	0.165	0.152	0.154	0.044
	Scripting time (Second)	1.3	1.47	1.48	1.4	3.18	1.03	2.3
	Rendering time Second)	0.404	0.417	0.526	0.531	0.498	0.497	0.166
	Painting time (Second)	0.139	0.217	0.397	0.312	0.433	0.222	0.342

	Others time (Second)	0.128	0.199	0.217	0.186	0.416	0.124	0.257
Selection Update I		**DS1**	**DS2**	**DS3**	**DS4**	**DS5**	**DS1_HM**	**DS5_LM**
	Memory (MB)	46.7	65.4	98	186	322	672	84.3
	Loading time (Second)	0.039	0.033	0.05	0.042	0.045	0.031	0.049
	Scripting time (Second)	1.92	3.95	8.4	11.4	38.1	1.8	23.96
	Rendering time Second)	0.246	0.177	0.2	0.213	0.27	0.206	0.168
	Painting time (Second)	0.359	0.290	0.294	0.344	0.369	0.229	0.305
	Others time (Second)	0.473	0.408	0.347	0.445	0.482	0.362	0.759
Hierarchical Clustering Analysis process		**DS1**	**DS2**	**DS3**	**DS4**	**DS5**	**DS1_HM**	**DS5_LM**
	Memory (MB)	47.2	62.9	158	287	576	676	404
	Loading time (Second)	0.005	0.016	0.014	0.014	0.008	0.005	0.005
	Scripting time (Second)	1.03	2.35	8.55	9.37	45.1	0.828	40.02
	Rendering time Second)	0.415	1.07	3.55	3.43	12.38	0.350	11.1
	Painting time (Second)	0.449	0.928	2.59	2.47	18.97	0.439	17.1
	Others time (Second)	0.247	0.216	0.615	0.589	3.43	0.139	3.35
Selection Update II		**DS1**	**DS2**	**DS3**	**DS4**	**DS5**	**DS1_HM**	**DS5_LM**
	Memory (MB)	49.3	74.9	162	293	655	685	465
	Loading time (Second)	0.022	0.031	0.025	0.020	0.018	0.028	0.115
	Scripting time (Second)	2.91	2.95	10.96	13.16	84	2.22	75
	Rendering time Second)	0.294	0.560	1.97	1.86	10.67	0.298	9.1
	Painting time (Second)	0.380	0.775	1.2	1.19	43.29	0.286	4.35
	Others time (Second)	0.148	0.506	0.580	0.419	`4.73	0.156	5.21

Table 4: Client-side performance evaluation results.

*MB is megabytes