THE UNIVERSITY
of EDINBURGH

Experimental Reproducibility in High-Throughput Multi-Omic Analysis Systems

Thesis submitted for the degree of Doctor of Philosophy

The University of Edinburgh

2015

Abstract

The reproducibility of scientific studies is an important issue facing modern biology. A large number of studies published today cannot be reproduced, and the situation has been described as a reproducibility crisis. It has been shown that the inclusion of computational analysis within a study, adds a further level of complexity in reproducing the findings in that study. Even the reproduction of only the computational component of a study is fraught with difficulty. When provided with the source data, a list of the tools used and a protocol, it can still be difficult to produce the same results. One reason for this is that variation between different tools, versions, configurations, dependencies, operating systems and hardware, all contribute towards variation in the results. The work presented here addresses the problem of reproducibility through the design and implementation of a novel reproducible analysis system, Cumulus. The Cumulus system combines technologies such as virtualisation and high-throughput workflow systems, to automate the process of fully recording an analysis environment. Recording of an analysis environment allows it to be shared and reliably reproduced by other researchers. Automating this process enables reproduction of bioinformatic analysis by high-throughput analysis systems.

The thesis then goes on to show how the Cumulus system was applied to reproduce and amend a published RNA-seq analysis and to create a novel proteomic analysis pipeline. This proteomic pipeline was then used in the analysis of a pilot study, to identify binding partners of the Nanog protein, dependant on a part of the protein previously shown to be required for the maintenance of pluripotency. This analysis resulted in the identification of a novel Nanog interactome. In addition to this, a further set of tools are presented, including the Stembio Visualisation Framework, a framework which enables the construction of interactive visualisations using the Cumulus system. The initial application of this framework has been accepted as part of a publication in the Journal of Experimental Medicine.

Lay Summary

It is important to be able to reproduce the results of a scientific study. Without the ability to do this, it is not possible to understand or build on the work the study presents. A large number of published studies however, cannot be reproduced. This is often because there are a large number of complex components involved in a scientific study and the reporting of these components is inadequate. The work presented here, Cumulus, is a software tool which automates the process of reporting of these components for large-scale systems. Cumulus is then used to reproduce and investigate an existing scientific study. In addition, a new scientific toolset is created within Cumulus to investigate high-throughput proteomic data. This scientific toolset is then used to analyse a dataset in a novel proteomic pilot study.

Declaration

I have read and understood The University of Edinburgh guidelines on plagiarism and declare that the work presented is my own, except where otherwise indicated, and has not been submitted for any other degree or professional qualification.

Duncan Godwin

Acknowledgements

Throughout the course of building the body of work which this thesis embodies, I have had the pleasure of having the help and support of a large number of people. Firstly, I would like to express my thanks to my supervisor Dr Simon R. Tomlinson. He has supported me throughout my work, providing the tools, knowledge and working environment to achieve everything I wanted. I am also deeply indebted to other members of the Tomlinson group both past and present, Dr Laura Skylaki, Dr Florian Halbritter, Will Bowring, Aidan McGlinchey, Hina Dalal, Dr Jon Manning, Dr Alison McGarvey, Dr Anastasia Kousa and James Ashmore. I would also like to thank my committee; Professor Val Wilson, Professor Ian Chambers, Dr Paul Travers and Dr Martin Jones. In addition, I would also like to thank people with whom I have worked or have allowed me to use their data, Dr Jingchao Zhang and Dr Nick Mullin.

I would also like to thank my friends who have helped me through the past few years, be it tea breaks, assisting with work, Clockwords or reading the future: Amy Pegg, Dr Julia Watson, Eilidh Livingstone, Dr Sam Hess, Dr Rikesh Rajani, Dr Filip Wymeersch, Colin Plumb, Karolina Punovuori and Dr Lindsay Bennett. Thank you to everyone at the SCRM for making it such an enjoyable period of my life and an exciting place to work. In particular thank you to Kelly Douglas, Fiona Oswald, Vaila O'Connor and Carolyne Ross.

Finally, I would like to say thank you to my partner, Dr Catriona Ford who has been patient and supportive, and also to my parents and brothers who have been great throughout. The work carried out here would not have been possible were it not for the funding provided by my funding body the Medical Research Council. Thank you all.

Contents

Figures

Tables

1 Introduction

1.1 Scientific Method

The scientific method is the foundation of scientific progress. This process and its underlying principles have been developed over several centuries as a pattern through which to determine empirical truths about the world in which we live (Achinstein, 2004). The historical development of the scientific method stretches back to figures such as Aristotle who used inductive reasoning from observations to infer general principles, and deductions from these principles, to test with further observations (Gauch, 2003). More recently, this was furthered by scientists such as Francis Bacon who relied on careful systematic observations to eliminate possible theories based on these observations (Bacon, 1878). These fundamental scientific principles have been developed over time to create a framework to describe the recognised process by which knowledge is obtained.

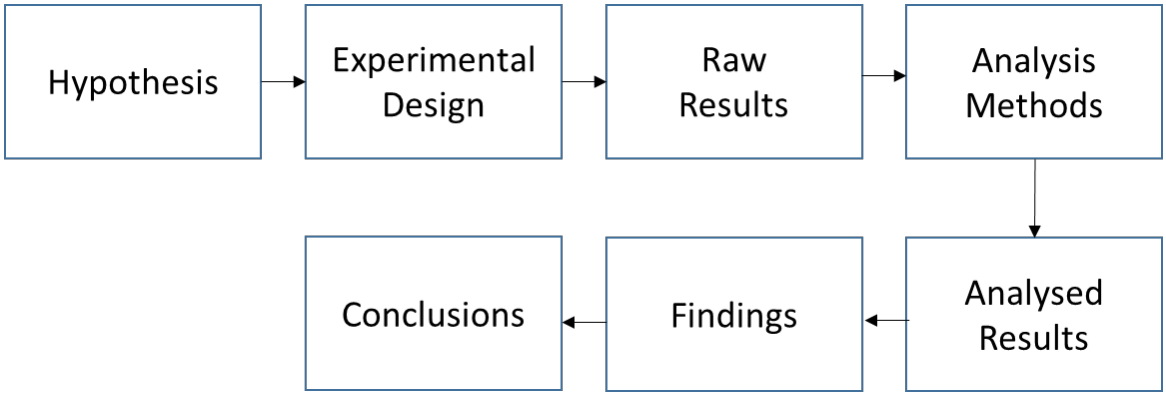

Figure 1 – The Modern Scientific Process: A block flow diagram showing the stages of the modern scientific process. The boxes show the result at each stage and arrows indicate the process flow.

The modern scientific method can be thought of as a process of multiple stages. This multi-stage process can be iterative with each iteration developing the ideas from the previous one. The stages of the modern scientific process are shown in **Figure 1**. Once conclusions have been reached, they are presented to the scientific community for peer review. Others can then inspect, interpret and try to reproduce the work to investigate its veracity.

1.2 Reproducibility & Standards

Scientific reproducibility is the ability of an analysis or study to be reproduced, either by the same researcher or by someone else working independently. This contrasts with the ability

of the researcher to achieve a consistent result which is regarded as replication (Leek and Peng, 2015). It is reported that one of the first proponents of the importance of reproducibility was Robert Boyle (Shapin et al., 1985). He proposed that scientific knowledge is made believable to the community through independent experimental reproducibility. Since then, reproducibility has been consistently accepted as a core tenet of the scientific method. Experiments which cannot be reliably reproduced should be disregarded and are of no significance to science (Fisher, 1974; Popper, 1959).

Independent reproduction is key to determining if the result is generalizable to all instances of a study and therefore not due either to fortuitous conditions present in the original study, unexpected artefacts, errors in experimental setup or other confounding factors. In addition to identifying erroneous results, reproducibility also enables the communication of the specific methods and discoveries to parties other than the original researcher. Without the ability to reproduce the result, other scientists are unable to evaluate, dissect, understand or build upon a finding.

In order to publish research which can be reproduced by other parties, all of the relevant information about that experiment must be made available. This includes all of the source materials, methods, conditions and tools used for each of the stages of the scientific method shown in **Figure 1**. Often the detail of the methodology is so important that even with all of this information, a small miscommunication can make it very difficult to reproduce results (Lithgow et al., 2017). Without methodological detail, it is all but impossible to fully reproduce or understand the reasoning behind a study, in order for others to interrogate and review it.

Bioinformatics has been defined as “the research, development, or application of computational tools and approaches for expanding the use of biological, medical, behavioural or health data, including those to acquire, store, organize, archive, analyse, or visualize such data” (Huerta et al., 2000). The process of recording and publishing an experimental method has an extra layer of complexity in experiments which include a bioinformatic analysis (Ince et al., 2012). The number of papers which include bioinformatic analysis has dramatically increased in recent years (Brusic, 2007). The complexity and scale of these analyses is also rapidly increasing, with many more large-scale studies combining data sets from multiple technologies into a single analysis (Kanehisa and Bork, 2003; Schadt et al., 2010). With larger scale and complexity, the bioinformatic community have identified reproducibility as an important issue and have started to work towards ensuring bioinformatic experiments are adequately reported. An example of this is shown in the Nature editorial “Code share” (Code share, 2014). Suggestions have included requiring the author to publish application code, analysis pipelines and scripts, or providing a description which would allow

an independent researcher to write their own program. These suggestions have often been adopted into the standards required by the publishers of scientific journals.

Scientific journals generally require an author to present their hypothesis, experiment design, methods and findings to the readers in written form. Depending on the journal, there may be specific style and formatting standards in place. These are somewhat varied as the journals can often cover a wide range of experimental methods and disciplines (Guo, 2016). Other academic organisations, such as research councils, which fund a great deal of public research, commonly have guidelines and requirements for how their funded research should be published (e.g National Research Council (US) Committee on Responsibilities of Authorship in the Biological Sciences, 2003). This often includes guidelines on the availability of additional material such as data and analysis tools (https://www.ukri.org/funding/information-for-award-holders/data-policy/common-principles-on-data-policy, 2018).

In addition to journal and organisation-specific guidelines, there are several common standards in place for scientific publications that require the publication of raw data and methods used in a study. These standards generally describe what must be included to adequately describe an experiment. Minimum Information for Biological and Biomedical Investigations (MIBBI), for example, is a project for promoting coherent reporting of biological experiments (Taylor et al., 2008). Other examples of these have been developed for specific data types e.g. Minimum Information About a Microarray Experiment (MIAME: Brazma et al., 2001), Standards for Reporting of Diagnostic Accuracy (STARD: Bossuyt et al., 2003) and the Minimum Information About a Proteomics Experiment (MIAPE: Taylor et al., 2007). These standards may require the publication of machine settings, material preparation methods and raw data which are identified as key to their interpretation.

There are data sources for which there are not yet publishing standards and journals do not commonly publish raw data. Biological image data, for example, has been highlighted in the Nature editorial “Sharing images” as an example of this lack of reporting (Sharing images, 2017). Analysis using image data such as fluorescence microscopy is increasingly common in biological investigations and can cover a large set of different methods (Combs, 2010). Studies which use image-based data may also include meta-data and derived data recorded alongside these images. Examples of meta-data are information about the instrument or conditions in which the data was captured. Derived can be data such as counting areas of colour, or co-location of colours on an image. Without the raw data of the image produced by the investigation, it is not possible to independently check for errors in any derived data such as this. One reason for lack of sharing given in the Nature editorial “Sharing Images”, is data

size. Image data can often be very large and therefore costly for the publisher (Sharing images, 2017).

As reporting guidelines have been developed by journals, funders and the scientific community, it has been found that some datasets do not fit into what are often highly-specific categories. Consequently, there has been increasing effort towards more generic publishing of less-structured data to capture a wider range of sources. This has taken the form of repositories such as Dataverse (King, 2007), FigShare (http://figshare.com), Dryad (Greenberg et al., 2009), Mendeley Data (https://data.mendeley.com), Zenodo (http://zenodo.org), DataHub (http://datahub.io), DANS (http://www.dans.knaw.nl), and EUDat (Lecarpentier et al., 2013). These repositories of less strictly-defined data require looser specifications than the specific reporting guidelines of single technology repositories. Findable, Accessible, Interoperable and Reusable (FAIR) guidelines are an example of an attempt to formalise this looser structure (Wilkinson et al., 2016). FAIR provides generic principles such as using clear identification for data objects and using standard protocols and language where possible.

It has been shown that even in journals that enforce the use of reporting standards, the results of scientific papers can often not be reproduced by other researchers. Nature has reported that the results of 'landmark' cancer papers could be reproduced in only 11% (Begley and Ellis, 2012) of cases. In a separate study, reported in Nature Reviews Drug Discovery, only ~20–25% of preclinical studies could be completely reproduced (Prinz et al., 2011). There are various reasons for this inability to reproduce results and, indeed, some level of disparity is to be expected in normal scientific discourse. Variation in the biology, reagent quality, timings and a huge variety of factors can cause the inability to reproduce results. One of the main reasons reported however is that of methods, code or raw data being unavailable; a particular problem with bioinformatic analysis (Baker, 2016).

In many cases, the bioinformatic analysis element of a paper is not reproducible (Baggerly and Coombes, 2009). A study in Nature Genetics showed that even with the publication of raw microarray data, 10 of 18 studies could not be reproduced (Ioannidis et al., 2009). The reproduction of the analysis using data from the original paper does not suffer from the problem of biological variation. Indeed, the predictability of an outcome given the same inputs is the key feature of a deterministic algorithm. The reasons for the inability to reproduce the results identified in this study was lack of specification of the analysis software and techniques.

1.3 Burden of Reproducibility

For many researchers, the reason studies are not reported as fully as they could be, is due to the time burden reporting carries and lack of incentive to do so (Sansone et al., 2012). For the researcher publishing their study, the cost of fully recording and reporting all of the elements required to reproduce experiments is very high (Monroe, 2015). To document a computational analysis, including all of the methods, versions, components and data can be very time consuming. Moreover, bioinformatic data analysis is often carried out by an expert third party who has little biological knowledge in collaboration with the reporting scientist who has little knowledge of the bioinformatics data analysis process. To a biologist writing or reviewing a paper for a journal, it may not be clear what the important aspects of the system are which should be documented.

In addition to the time required to capture all of the required detail of a study there is the effort required for those trying to reproduce a study. This can also be significant; increasingly so for poorly reported studies or where information has not been captured. Lack of information requires the researcher to guess what was done in the original study or try multiple variations.

1.4 Data Re-use

Data re-use is the use of data collected by researchers for a specific purpose, used by researchers for a different purpose (Pasquetto et al., 2017). The advantages of being able to re-use data are well documented. These include lowering costs, improving the value-for-money of research and reducing the impact on animals (Gould, 2015). Data sharing is releasing data so that it can be used by other researchers for this purpose (Pasquetto et al., 2017). Data is often shared through a website-based data repository containing many datasets of a similar type. Examples of these include Gene Expression Omnibus (GEO: Edgar et al., 2002), PRoteomics IDEntifications (PRIDE: Jones et al., 2006) and the Sequence Read Archive (SRA: Leinonen et al., 2011) which each include many thousands of studies. In re-use, data from many unrelated studies are often combined to produce a meta-analysis (Haidich, 2010)

The ability of researchers to re-use published data is heavily impacted by the quality of both the shared data and the reporting of the research methods used to produce this data (Rung and Brazma, 2013). In order to combine data which is potentially from multiple different instruments, labs and source material it needs to be processed to take account of these differences (Rung and Brazma, 2013). Without information on both these differences and any analysis steps taken, combining data from multiple studies is difficult or not possible.

This makes proper reporting of these studies important in enabling their re-use. Improving this situation could vastly improve the efficiency of research, enabling meta-analysis in areas in which it is currently not possible (Molloy, 2011).

1.5 Sources of Potential Variability in a Bioinformatic Analysis

Assuming the same input data, there are several elements of a bioinformatic analysis which are open to discrepancy. If these elements are not repeated in exactly the same way, they can have a varying impact upon the resultant output of the repeated analysis.

1.5.1 Analysis Method

At certain points in an analysis there are several potential paths an analyst can choose, e.g. including different sets of tools or analysis techniques. These potential paths each have varying levels of impact on the results. The selection of a novel or rarely used technique may be very impactful, whilst changing a threshold may just display the best representation of a dataset. In order to publish reproducible methods, it is important to report the specific method used at each stage of an analysis. For example, when there are multiple different data sets from which a comparison needs to be made, these data sets may often need to be normalised (Dodge, 2006) to adjust them to a comparable scale. There are several different methods which can be used to do this type of normalisation. Each method can have a significant effect on the comparison and result (Quackenbush, 2002). This is only one decision of many which may happen during an analysis each of which can lead to variation in the result of the analysis.

1.5.2 Software and Versions

It has been shown that there is significant difficulty in identifying the correct software tools and databases based soley on the name of the tool included in a paper (Duck et al., 2015). This reported difficulty is due to ambiguity in the names and the extensive number of resources available. As software is developed over time, different versions are released and used. If the same software and version of software is not used to repeat an analysis, there can be substantial differences between the analysis results. As shown in **Figure 2**, version 1 of Bowtie (Langmead et al., 2009), a tool which aligns sequenced reads to a reference genome, has very different levels of accuracy to Bowtie 2 (Langmead and Salzberg, 2012) during alignment of simulated Illumina data. Reproducing an analysis using Bowtie 2 in the place of Bowtie 1 would give different results.

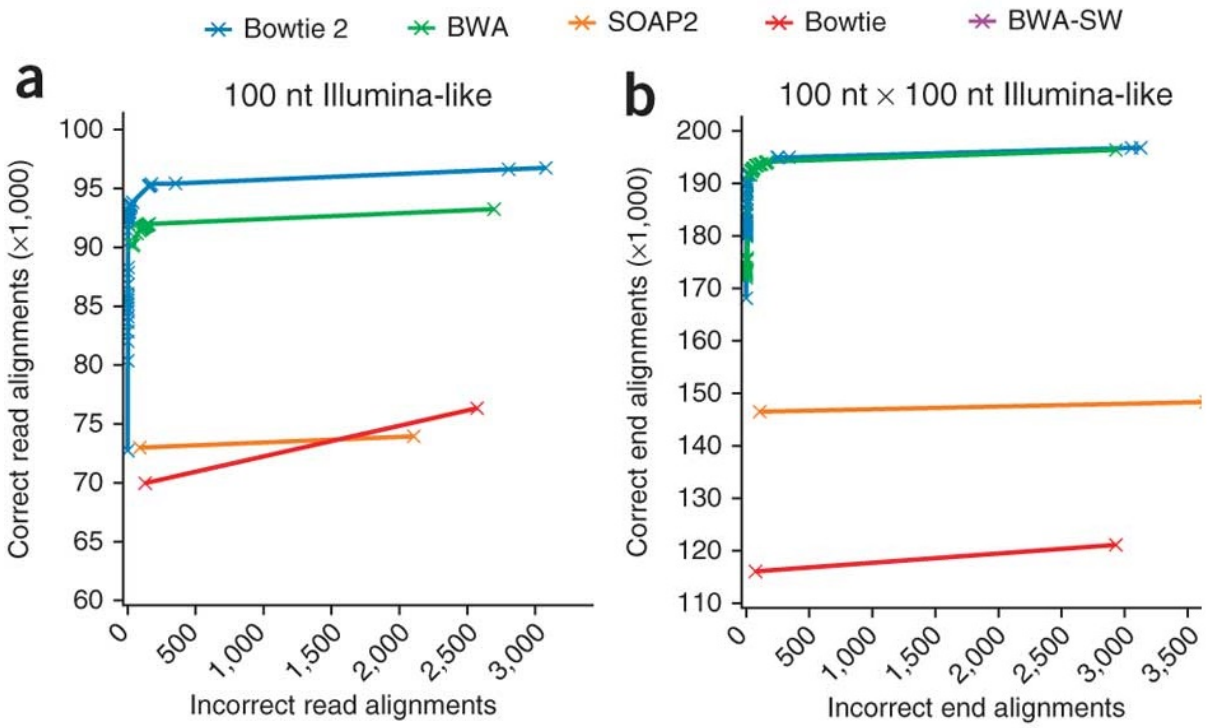

Figure 2 - Sensitivity and Accuracy of Alignment Using Simulated Reads: Cumulative number of correct and incorrect alignments from high to low mapping quality for simulated Illumina-like unpaired 100 nt dataset (**a**) and for simulated Illumina-like paired-end 100 nt × 100 nt dataset (**b**) using indicated aligners. Adapted from Langmead and Salzberg, 2012.

1.5.3 Software Libraries and Dependencies

Software libraries are programs which used by other software to carry out common functions. These common functions may be a regularly used algorithm or instructions to enable interaction with a hardware component. The library exists so that the code to do this common function only needs to exist once and can be used by multiple different applications. Multiple libraries and resources which provide a distinct set of functions can also be brought together into one unit; this is often referred to as a software package. The need for the presence of a particular software library in order that a separate software tool or package may operate correctly is referred to as a "dependency".

Software packages can themselves depend on other packages and the complexity of these requirements can increase dramatically as the number of applications increase. Each software tool has a set of dependencies which themselves have a further set of dependencies *etcetera*. This structure is referred to as a dependency tree. With multiple tools, the difficulty in managing overlapping dependencies also increases dramatically. Over time, software tools are changed and developed in order to improve them and add new features. Intermittently the software is tested and is released as a fixed version.

Dependencies usually refer to one specific version of a piece of software. Each version of a software library or dependency is likely to have slightly different code and will function slightly differently. The supported versions specified as a dependency to an application can often be in the form of a range, version 2.0-2.5 for example. This leads to an application which apparently functions acceptably but may be using slightly different code in its dependencies and which may produce subtly different behaviour or results.

Software dependency conflicts are an additional problem when trying to reproduce a study. One example of this is when one piece of software is dependent on a specific version of a dependency and another piece of software is reliant upon a different specific version of the same dependency. If both pieces of software are installed on the same computer, when the second is installed, it can change the dependencies available to the first, breaking its dependent software. The need to resolve conflicts and the inherent difficulties in incompatibilities has led to the coining of the term “dependency hell” (Jang, 2006).

Software deployment is the installation and configuration of software tools in an operating system. In order to deal with dependency problems, operating system and software library maintainers have developed dependency management applications which have become commonplace in software deployment. Package management tools are one example of this, they maintain a list of software packages and installation scripts available to an operating system which are known to work together. In addition to this they detail all of the other packages on which those packages depend. When a software package is installed, the package manager checks to see if all of its dependencies are also installed. If not, the required packages and all of their dependencies are also installed down the dependency tree in a recursive fashion. As new versions of applications are released, dependencies may change, and a package management tool may keep records for multiple versions of tools. The Linux operating system for example, has several package management database tools such as Apt/DPKG (https://wiki.debian.org/Teams/Dpkg, Blackman 2000) and Yum/RPM (http://yum.baseurl.org, Foster-Johnson, 2003). The Apple operating system macOS / OSX has brew (https://brew.sh) and Microsoft Windows has chocolatey (https://chocolatey.org). These all aim to solve many of the common generic dependency and deployment issues for their specific operating system.

The maintainers of operating systems often end the support of tools very rapidly once new ones are released. Commonly, only the most recent versions are available and installing and configuring anything other than the latest versions can be difficult (Helmke et al., 2017). Reproducing an old analysis environment previously published in a paper using a dependency management system would require the user to know which version of each

piece of software the reporting researcher had installed. Even with this knowledge, going back to a specific point in the release cycle of the thousands of packages which make up an analysis environment would be very challenging. Each tool would have to be downgraded to the correct point or an earlier release would have to be upgraded. As each of these downgrades or upgrades has the potential to change the other installed packages, achieving a specific environment could be very complex. In such circumstances, when a specific set of tools is required other than the current working set, package managers are of limited use.

A programming language is a formally defined language which can be used to instruct a computer to carry out set functions. Programming languages are used to create programs which specify particular algorithms. Much like operating systems, several programming languages have their own package and dependency managers, notably Ruby (https://www.ruby-lang.org, Flanagan and Matsumoto, 2008), the Comprehensive Perl Archive Network (CPAN: http://www.cpan.org) for Perl, the Python Package Index (PyPi: https://pypi.python.org) for Python and Apache Maven (https://maven.apache.org) for the Java programming language.

The configuration of multiple scientific tools in an analysis environment, may require the use of multiple different versions of the same programming language. A common manifestation of this version compatibility issue is the requirement for both versions 2 and 3 of Python, a widely used programming language in bioinformatics (Perez et al., 2011). Problems related to the use of multiple versions of Python have been widespread for a long time (Reitz and Schlusser, 2016). Enabling multiple versions of languages has proven to be complex, with different languages solving the problem in a variety of ways. For Python, pyenv (https://github.com/pyenv/pyenv) enables the user to install multiple versions of Python and swap between them. Ruby enables multiple versions of itself based on the project directory. Java uses Apache Maven to define an XML configuration file for each project.

A core tool of bioinformatics for many years has been R (Ihaka & Gentleman, 1996), a statistical language and set of libraries widely used for data analysis. R itself has built-in package management, versioning and deployment functionality similar to that of other programming languages supported by the Comprehensive R Archive Network (CRAN: https://cran.r-project.org). In addition to this, R has further bioinformatics-specific packages, available through projects such as Bioconductor (Gentleman et al, 2004). The various R package management utilities configure and update tools using install scripts. These install scripts can install new R libraries, but cannot install new operating system dependencies (Gillespie and Lovelace, 2016). Each operating system dependency must be manually installed, and the R install is blocked until this is carried out. This provides the basic structure

of package management and deployment for R. Versioning of R packages is done at a whole-system level. A set of versions are defined for each release of R so when R is updated, a new set of versions are installed. This makes it difficult to work with different versions of R components which are not within the same release. Additionally, as older R releases are removed from the package management database it becomes more difficult to work with these versions. There are no easy means of stepping back to previous versions of R software. This can often lead to users installing multiple versions of R at once for different purposes on a single computer. It can be difficult to know which version of a package is in use as R updates can be automatically triggered (Gillespie and Lovelace, 2016). This can hide software version changes from the user.

Even with the proliferation of tools and standards related to managing the installed versions of software and dependencies, creating and then keeping an analysis environment running over time remains a complex task. In addition, describing this complex environment to someone else in order for them to reproduce it-whilst ensuring that the tools used and all of the dependencies remain the same-is also very challenging (Garijo et al., 2013). This complexity leads to variation in the reproduced environment and, therefore, variation in the results of any reproduced analysis. There are currently no comprehensive means to successfully address this problem.

1.5.4 Hardware Variability

Differences in the underlying computational hardware used in an analysis can produce variation in the results of that analysis. Applications are commonly compiled from human-interpretable source code to machine code using tools specific to the hardware on which it will run. Computer hardware often has varying functionality, and in order to carry out this compilation, the compiler has to map the requirements of the software, to what is available in the hardware. This can lead to variation in the way the compiled software will run on different types of hardware (Corden and Kreitzer, 2012). Where a system has multiple processors working together, variations in the interactions and timing between these processors can also cause differences (Robey et al., 2011). One such example is shown in **Figure 3**, a figure generated from a post from the bioinformatics support forum Biostars (https://www.biostars.org/p/90390). The Burrows-Wheeler Aligner (BWA: Li and Durbin, 2009) is a tool used to align sequenced RNA or DNA reads to a reference genome. The Biostars extract shown in **Figure 3** shows four different alignments carried out with the same input data but different numbers of processors. The output shows that version 0.7.5a-r405 reports a different number of alignments when different numbers of processors are used by

the software. The difference in this case may be relatively small but it highlights that minor changes in runtime parameters can have an impact upon the analysis results.

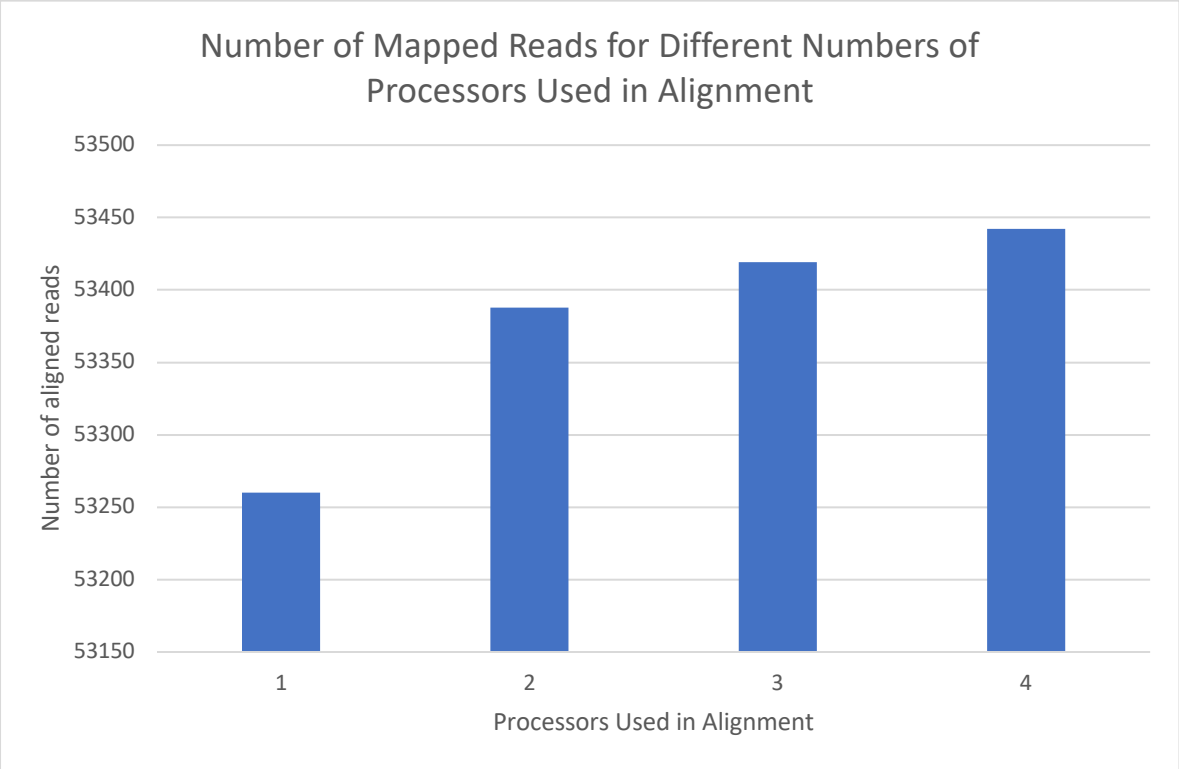

Figure 3 – Variation In BWA Output With Different Numbers of Processors: A bar chart showing the variation in the output of the reported alignments when BWA version 0.7.5a-r405 is executed with the same data set and a different number of processors. Reproduced with data from https://www.biostars.org/p/90390.

The variation in the case of **Figure 3** is likely to be an unwanted side effect of parallelisation or bug. Unrecorded variations of this type affect the reproducibility of an analysis.

1.5.5 Randomization in Software & Seeds

Statistical software often requires the use of random numbers which may cause variation in the output. For example, k-means clustering is a method used for analysing clusters of results within a dataset (Lloyd, 1982). The k-means clustering method uses an initial random selection which can produce a marginally different result each time (Celebi et al., 2013). Commonly available implementations of this however, can accept a random seed. A random seed is a number used to initialise the pseudo random number generator from which all the randomness in the code is generated. Setting this seed to a known number means that each time the script is run, the output will not change (Park and Miller, 1988). This means that a random number can be generated when the analysis is carried out and retained. If this retained number is used again when the analysis is re-run, the random initial selection will also be the same resulting in the same output. Published analysis should therefore contain

these values either in the methods or recorded in the analysis script so that the researcher attempting to reproduce the study can use them.

1.6 Reducing Potential Variability in a Bioinformatic Analysis

In order to enable the reproducibility of a bioinformatic analysis by other researchers, each potential source of variability in that analysis has to be captured and communicated in the publication. This enables the researcher reproducing the study to follow the same process and configure their environment in the same way as the initial study. **Section 1.5** detailed the sources of potential discrepancy which occur when carrying out and communicating an analysis. The following Section details various technologies which comprise the state of the art in simplifying and enabling aspects of reproducing an analysis.

1.6.1 Sharing Analysis Methods: Automated Analysis and Workflow Systems

A workflow management system is a piece of software that provides an infrastructure to setup, execute, and monitor scientific workflows. They provide an environment where *in silico* experiments can be defined and executed (Oinn et al., 2004). These workflows contain the methods, scripts and configuration used to analyse a specific experiment or study. Capturing this information within a workflow enables scientists to run the same analysis in an automated fashion on different data sets or send the analysis to others to allow them to use the same methods. Workflow systems represent some of the most advanced attempts to solve the problem of the reproducibility of a bioinformatic analysis (Cohen-Boulakia et al., 2017).

The primary function of a workflow system is to reduce the complexity of running an analysis for the user. Analysis tools are often complex and require the knowledge of an expert in the field in order to use them. Workflow systems can simplify the process of running a tool by capturing this expert knowledge in code, enabling the process of data analysis to become automated. This has two immediate advantages; firstly, they can enable the researcher to bypass repetitive time-consuming and complex tasks. Secondly, a user, highly proficient in the interpretation of data generated by a technology, can produce a workflow which can then be used by other, less proficient users. This capturing and communicating of methods also makes workflow systems ideal for enabling reproducibility, through the publication of these methods.

Workflow systems standardise software tools by wrapping them in a generic format which allows them to be used interchangeably within the workflow system. The wrapper is generally a script which interprets the inputs and outputs of a tool. These outputs are then linked with a graphical interface which allows them to be connected together to form a

workflow. Examples of workflow systems include Taverna Workbench (Oinn et al., 2004) and Konstanz Information Miner (KNIME: Tiwari and Sekhar, 2007). These kinds of concepts have been further developed in bioinformatics in recent years with the development of tools such as the Galaxy project (Blankenberg et al., 2010) and GeneProf (Halbritter et al., 2012).

Since the work described in this thesis was carried out, the Galaxy project has released its own package management tool, the Galaxy Toolshed (Blankenberg et al., 2014). The Galaxy Toolshed enables Galaxy users to manage the addition of software components to their installed Galaxy system. The approach of the Galaxy Toolshed has been to create a package management tool similar to Apt or Yum in Linux. The Galaxy Toolshed package management is focused primarily on analysis tools and then links out to an operating system level package manager for external dependencies. The Toolshed system is then tied into the user's workflow, thereby connecting the data and results to the experimental methods used.

The Toolshed system has similar strengths and weaknesses to other package managers. It is focused on enabling the user to run the latest versions of tools. Whilst tools can be packaged in such a way as to enable reproduction of a study, dependencies must each either be packaged manually or linked externally in a dependency management system. Because older packages are regularly removed from these dependency management systems the latter can often be unstable. Old packages which are no longer widely used or maintained are referred to as legacy tools. Legacy tools are often poorly supported or not available (Van Heusden, 2015). In some circumstances, two pieces of software in a workflow have dependencies on different versions of the same software; these are referred to as conflicting dependencies. If this conflict occurs in something the operating system can only have one of, such as a core hardware driver, these two pieces of software cannot easily be installed on the same computer. The Galaxy Toolshed does not easily support this scenario or the ability to simultaneously run conflicting software. If different versions of software within the Toolshed have dependencies which have conflicting dependencies, there can be as much complexity in configuration as if the process were being managed manually within Galaxy.

1.6.1.1 Expert Knowledge

There are now many biological technologies and techniques available to the modern lab (Berger et al., 2013). Each of these requires different analysis software to interpret its results. It is very difficult for an individual researcher to be expert in any more than a small number of these. High-throughput workflow-based analysis systems try to address this problem of the explosion in these technologies by providing pre-boxed workflows which require minimal modification by the end user. For example, the GeneProf system provides

workflows for nucleic acid sequencing technologies e.g. RNA-seq and ChIP-seq. There have been many published workflows for the Galaxy system (Davidson et al., 2016; Sheynkman et al., 2014). In the majority of cases a user can then run a standard analysis without having to have expert knowledge in a particular area.

In addition to capturing the expert information on a bioinformatic analysis, they also allow the end user to have minimal knowledge of high performance or large-scale computation. Large-scale or high-performance computing resources aggregate many computers together into formations which give more computational power than would be available from a standard personal computer. The increased power of these systems allows tasks to be split up and executed in a fraction of the time it would on a single computer. The downside of these systems is they are often very difficult to operate, even with many years of experience (Pancake, 1995). Workflows can allow an analysis to be run interchangeably on small or large-scale computing resources. This gives access to these complex, high-throughput resources for researchers who lack formal programming skills or high-performance computing training (Spjuth et al., 2015).

1.6.1.2 Modular Design

A bioinformatic analysis can often require many steps. For example, RNA-seq—a commonly-used transcriptomic assay technique—uses high throughput sequencing to identify the sequences of the fragmented RNA molecules within a sample. These fragments can then be aligned to a genome to measure all the RNA molecules contained within the sample. In a standard RNA-seq data analysis, there may be more than 10 analytic steps (Conesa et al., 2016). Different tools often require different file formats, data conversion between these formats can cause the number of these steps to rapidly increase. A study which includes, and possibly combines, the analysis of multiple data types is referred to as a multi-omics or poly-omics study (Huang et al., 2017). Multi-omics studies can easily reach hundreds of analytic steps.

Traditionally, the process of managing the steps of an analysis has been simplified through the use of scripting languages such as Bash (Ramey and Fox, 2002), Perl (Christiansen et al., 2012) or R (Ihaka and Gentleman, 1996). Scripts written in these languages often have to be modified for each analysis and changed as software versions are updated. This is avoided in workflow tools as the software is packaged in a standard packaging structure. This standard structure enables all of the data and configuration which is required to run the tool to be swapped without editing the script. This design is a known pattern in software engineering and is referred to as a modular design (Gamma, 1995). A common alternative to this pattern is the monolith design in which all of the components are interwoven in a single

block. Modular designs enable far greater re-use and automation of components than monolithic software designs without having to know how the components work. As a result, pipelines can be quickly built up, reused and changed without an in-depth knowledge of all the components involved.

1.6.1.3 GeneProf: A Case Study Bioinformatic Workflow System

Workflow analysis systems capture and record a subset of the potential sources of bioinformatic discrepancy identified in **Section 1.5**. In order to capture all potential bioinformatic discrepancy, including that not addressed solely by a workflow system, the workflow system must be combined with other tools and built upon to add the missing functionality. The concepts of modularity and reusability of analysis tools which allow workflow tools to reproduce methods is a common pattern to all workflow systems.

GeneProf (Halbritter et al., 2012) is an example of a high-throughput workflow-based system. GeneProf automates the bioinformatic analysis of genomic next generation sequencing data. The specific data it is focussed upon comes from two technologies; RNA-seq (described in **Section 1.6.1.2**), and Chromatin ImmunoPrecipitation sequencing, ChIP-seq (Robertson et al., 2007). ChIP-seq identifies the sequences of DNA with which proteins of interest are interacting at a whole genome level. These sequences are then aligned to a reference genome. Where multiple fragments align to the same genomic region, this generates peaks of binding indicating the areas of interest.

The GeneProf system provides a pre-set and configurable workflow which guides non-technical users through an analysis. The pre-set GeneProf analysis uses a wizard-type structure which assists the user by stepping through the analysis process, picking a strictly-defined set of software tools with sensible defaults and configurations. This means the analysis can be standardised, and at each stage the resultant data is highly comparable with other datasets analysed using the same pre-set configurations. The benefit of this limited toolset is a database of standardised datasets with a well-documented and tested analysis methodology. GeneProf includes a database of analysed and curated publicly available data (Halbritter et al., 2014) which has been produced using these methods. These publicly available studies can be compared and mined for patterns across multiple organisms, cell types and conditions.

Data and analysis within the GeneProf system is structured around its internal concept of experiments. These experiments are one or more datasets combined with an analysis structure and results. These can then be annotated with meta-data such as cell type, organism type and other technical study details. An experiment provides an encapsulated

record of all the data, analysis tools and configuration details of a study. Experiments can be searched and filtered using their data and meta-data then combined. Statistical studies which combine the results of multiple other scientific studies in this way are referred to as meta-analyses.

On creation of a new experiment within Geneprof the first step is to add the required datasets. Data can be uploaded to experiments through a variety of means, including HTTP upload, transfer from an external File Transfer Protocol (FTP) server or external resource such as the SRA (Leinonen et al. 2011). Alternatively, data can be imported from another experiment already contained within GeneProf at any stage of the analysis. This uploaded data then becomes available for processing via the system's analysis tools.

The processing and analysis structure of the experiment is captured in an experiment page and workflow. An experiment page curates the output data from the workflow into a set of results presented to the user. The GeneProf system has a pool of analysis machines on which to run a workflow. These machines are usually high-quality servers pre-configured with the full set of software tools GeneProf offers. The system launches each analysis tool on a machine in turn, passing in the required data and settings and reading back the resultant output. Once the experiment datasets are processed by the workflow, a report is created by the system, detailing the results. The experiment parameters can then be modified and re-run until the point at which the user is happy with the workflow.

Once all the experimental analysis is completed, the results can be made publicly accessible from within GeneProf. Making an experiment public allows anybody to access it and locks it, blocking editing and keeping it as a record of the methods and settings used in the analysis of the data. A permanent hyperlink, or "permalink", is then made available which will always link users to this experiment. This permalink can be included in any publication of the study, linking a record of the bioinformatic analysis to the text. This link allows the readers of the published paper to immediately view analysis methods and the raw and processed data for each of the stages of the analysis and interrogate it themselves.

Making an experiment public also makes all the workflow details, results and methodology available to all GeneProf users. These public data sets can be used to identify patterns or 'mine' system wide information common to many datasets. Data mining can be carried out using the GeneProf data explorer, a tool to generate flexible plots from the public data using various plotting tools. The data explorer can produce histograms, correlation matrixes and principal component analysis plots with a variety of configurable options. Alternatively, gene-centric pages in the system show a specific pre-set range of gene-centric plots of the public data. External applications can also be used to interrogate the data using the GeneProf

Application Programming Interface (API) (Halbritter et al., 2014) a Representational State Transfer (REST) API (Fielding and Taylor, 2000). This API allows the download of different selections of public and private data in a range of formats.

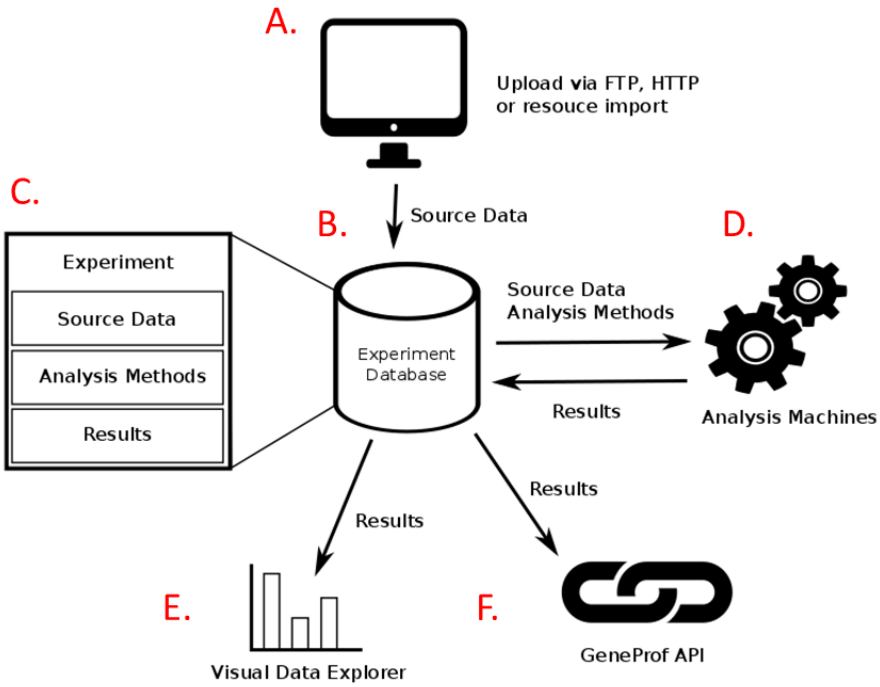

Figure 4 - The Structure of GeneProf: A schematic diagram showing a structural overview of the GeneProf system and its components. The arrows indicate the movement of data between the experiment database and the components of the system. (**A**) The user's computer from which they can upload their source data. (**B**) A central web application with a database of experiments. (**C**) A block diagram showing the structure of a single experiment with source data, analysis methods and results. (**D**) Analysis machines with a set of installed software tools. Source data and analysis scripts are sent to these machines and results posted back. (**E**) The GeneProf visual data explorer accesses experiments for use in visualisations and visual data mining tools. (**F**) The GeneProf Application Programming Interface (API) which provides programmatic access to the GeneProf experiment database.

The GeneProf system consists of a number of core components, shown in **Figure 4**. These are: an analysis infrastructure with a suite of tools, a database of private and public studies, and a web application and API which act as interfaces to both of these.

1.6.1.4 GeneProf Web Application

The GeneProf web interface is a set of web pages created with Java Server Pages (JSP) and Java servlets (Schildt, 2017). The web interface covers a range of features, but the two main types of pages are experiment-centric pages and gene-centric pages. Experiment-centric pages display information related to a specific experiment including the analysis data, methods and results. The gene-centric pages display data related to a specific gene in a

specific organism. This is both data from a selection of the published experiments and data from external databases such as GEO and BioGrid (Stark, 2006). Graphs and visualisations of data on pages are generated by a Java servlet which executes R scripts and returns the image generated by the script back to the web application. As parameters are changed new server requests are made which re-runs the R script and updates the visualisation. The web interface provides access to a range of other functions such as searching, documentation, administration and analysis configuration pages.

Workflows for experiments can be constructed using a web-based composing tool. This composition tool uses JavaScript to provide an interactive web interface that lets the user connect blocks representing analysis software together to form a workflow in the form of a flow chart. Several hundred analysis tools are available for selection from a toolbox within the interface called the ‘module palette’. The inputs of one analysis tool can be connected with the outputs of other tools if the data type of the input and output is the same. If, for example, one tool outputs sequence data and another tool can accept sequence data as an input, they may be connected together. This validation of the workflow gives the user some guidance as to which tools it is practical to use at each stage. Each tool has a set of sensible defaults for its execution options and these can be changed by the user to give a finer level of control over the analysis. The interface also offers a range of pre-set workflows defined as wizards, an example of which is shown in **Figure 5**. These allow the user to submit data of a certain type for a pre-configured analysis.

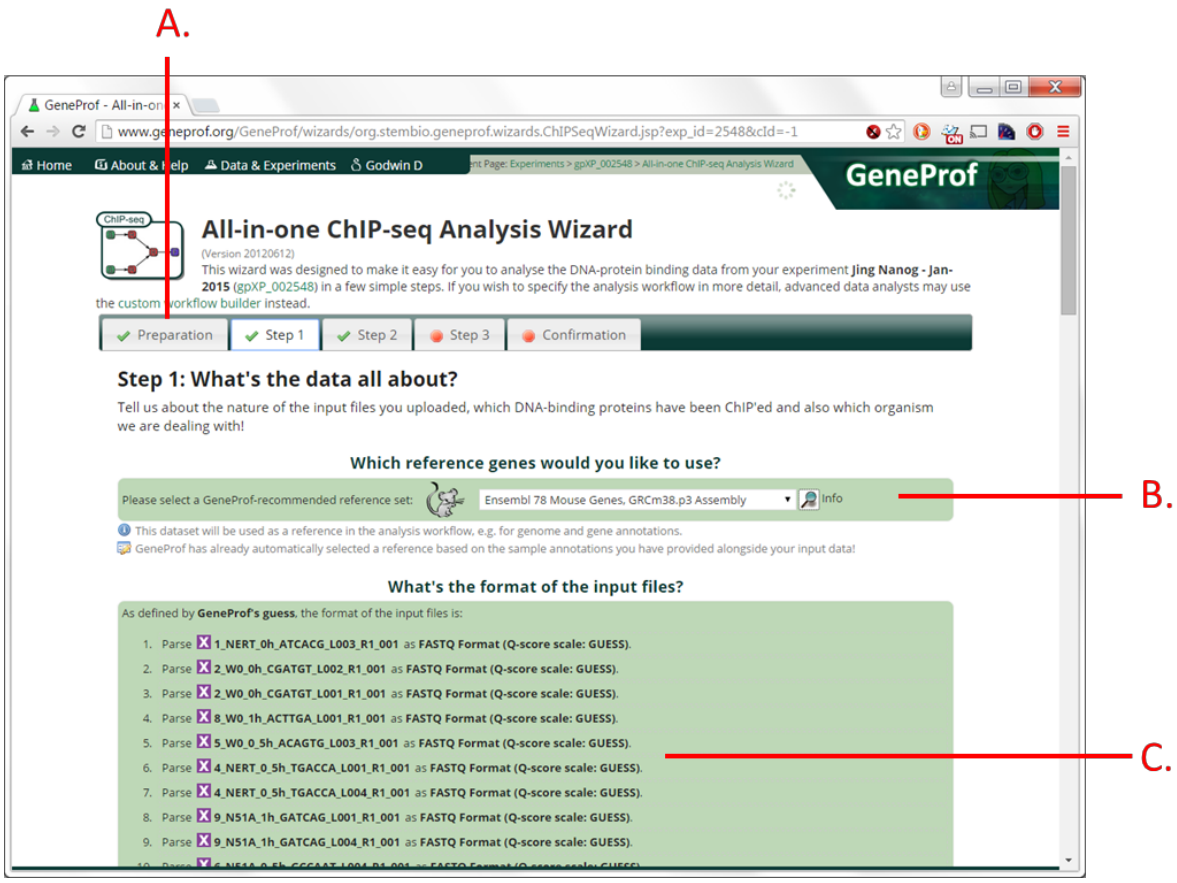

Figure 5 - The GeneProf All-in-one ChIP-seq Analysis Wizard: A screenshot of the GeneProf high-throughput workflow system showing the selection of a reference genome and input files for an analysis in pre-configured analysis wizard. (**A**) The selectable wizard steps displayed on tab interface components. (**B**) The reference genome selection combo box. (**C**) Input files for the wizard showing detected formats and the sample names.

1.6.1.5 GeneProf Study Database

At the time of writing, GeneProf contains 216 public experiments and 13,252 public datasets. GeneProf uses a Structured Query Language (SQL: Chamberlin & Boyce, 1974) database for experiment data with users and studies spread across several tables and schema. Users can browse studies through the web interface or access them through the GeneProf API. Different annotations defined when experiments are created allow the data to be segmented in different ways to answer specific questions. For example, experiments are tagged with information such as cell type and organism; this allows users to investigate variation in expression of genes in a specific cell type. With every new study published in GeneProf this information is further enriched.

1.6.1.6 GeneProf Analysis Infrastructure

Analysis in GeneProf is carried out using a group of specially configured analysis computers. These run a standard installation of CentOS (https://www.centos.org), a Linux based

operating system. In addition to this they each have all of the GeneProf analysis tools installed and a piece of constantly running software called a daemon which controls the execution of experiments. This daemon regularly checks for new experiments in the study database. When new workflows are submitted for analysis by the web interface, the experiment is marked as ready in the database. The daemon then finds the experiments which are marked as ready and claims them, changing their status and recording the identity of the computer carrying out the analysis. The daemon then launches a new Java workflow application which processes—one stage at a time—the outstanding stages of the workflow for the selected experiment. For each stage, the workflow application downloads the relevant data from the study database. It then runs the selected tool with the parameters saved for it. Once an analysis stage has been completed, the workflow application uploads the resultant files to the study database and marks the stage as done. If there are still outstanding stages the processing daemon goes on to process these, otherwise it marks the experiment as done and moves on to process other experiments.

In the current structure of GeneProf, the addition of new analysis tools to the available toolset is done by implementing a Java wrapper to interact with the tool. This Java wrapper is based on a custom GeneProf structure which provides standard methods for loading data, executing a tool and retrieving the results. Each of these methods must be filled in by a contributing developer to call the relevant methods with the correct parameters. This method of extending the system has significant problems as the amount of work to incorporate a new tool is non-trivial. It can take an experienced developer several days to weeks to implement a single tool with several thousand lines of code. In addition, the user implementing the wrapper needs to have a high level of knowledge of both Java and the inner workings of the GeneProf system. This provides a significant barrier to entry for many users wishing to contribute to the GeneProf system.

Each GeneProf module has full access to the experiment database and the analysis system with the ability to access both public and private experiments without requiring additional security credentials. This means all the code must be written to a high standard by trusted individuals. Finally, as all individual modules need to be executable from the same system, the analysis computers must be configured to run all the system modules simultaneously. As more are added the configuration of the analysis system becomes increasingly complex.

1.6.1.7 GeneProf: Summary

Throughout this thesis, the GeneProf system is used as a case study of a bioinformatic workflow system as it already provides some functionality to record the software tools used within an analysis. In addition, it provides a rich set of tools and presentations which enable

the user to easily run a fixed set of analyses. The GeneProf workflow analysis system records both the list of tools used and the parameters for running each. In order to reproduce a particular study, however, the user must already have the GeneProf tools installed and configured. This analysis environment can be complex to set up and configure and, beyond the name and version numbers of the tools used, there is little information provided. GeneProf provides no means of maintaining or recreating the analysis environment. In the following sections of the Introduction, the potential extension of the GeneProf system will be discussed, along with the tools available to carry this out.

1.7 Reducing Operating System and Hardware Configuration Variability

As discussed in **Section 1.5.3**, differing software libraries and dependencies can cause variability in the execution of an application. In order to standardise execution, the full set of dependencies including the operating system need to be controlled. In addition, as shown in **Section 1.5.4**, some differences in the low-level configuration of hardware can cause variation in the results of an analysis. A number of technologies have been developed which standardise computing platforms and create a generic environment which should always function in the same way. These technologies are detailed below.

1.7.1 Virtualisation & Virtual Machines

Virtualisation—in computer science—is the concept of creating a virtual as opposed to physical version of something such as a piece of electronic hardware and is a concept initially developed by IBM in the 1960s (Adair, 1966). In recent years, developments in generic hardware have expanded the use of virtualisation in creating virtual machines. These virtual machines are virtual versions of full computers, including memory, processors and interconnects running inside of a physical computer. Instead of interacting directly with the computer hardware, the software within a virtual machine interacts with the virtual machine acting as if it were the hardware. The virtual machine can then pass these commands on to the actual hardware. Running a virtual machine inside a physical machine in this manner, enables the abstraction of the software running on a virtual computer from the physical hardware itself. This software can then be moved to a virtualisation environment running on some other operating system and physical hardware. Differences in the underlying hardware and software can be ignored and the virtual machine will run in exactly the same way on this new environment as on the old environment. In addition to creating this abstraction between the virtualised software and its execution environment, virtualisation enables splitting up of physical hardware by running multiple, isolated virtual

machines on one computer. Each virtual machine can thus be given a percentage of the physical resources such as disk or CPU time, allowing one computer to be used as if it were multiple, enabling greater optimisation in the use of physical hardware resources.

In a modern virtual machine environment, all of the hardware functions of the computer such as networking, processing and storage are provided by a program called a hypervisor (Barham et al., 2003). In a standard computer, an operating system is the main program or set of programs which control programmatic access to the underlying hardware. With hypervisor virtualisation, this hypervisor interacts with the host physical machine taking over this role from the operating system. The software components, including the guest operating system running in the virtualized environment then interact with the hypervisor. The hypervisor can then control what percentage of the physical resources are being used by a particular virtual machine and regulate the access rights a virtual machine has, in order to stop it from carrying out malicious actions. The virtualised computers can, in this way, run without being able to interact directly with each other or the host computer. The operating system and the user often do not appreciate the system is running inside a virtual environment as the functionality provided by the hypervisor remains much the same as physical hardware.

One type of virtualisation is emulation, which is where a virtual version of different physical hardware to the underlying hardware is created by the hypervisor. The hypervisor then converts commands made to the emulated hardware into calls to the physical hardware. This translation means that applications will generally function more consistently across different hardware platforms. It can however, lead to emulation being somewhat slower than virtualisation without emulation. The use of emulation to create a reproducible system could better guarantee that the software it ran would execute in the same way across different hardware systems. The majority of virtualisation systems are not emulation systems and cannot run software which is not compatible with the underlying hardware on which it is running (Smith and Ravi Nair, 2005). One exception to this is the Quick Emulator (QEMU: https://www.qemu.org) hypervisor which does support hardware emulation for its virtualisation.

Separation of software from the physical machine allows entirely different software to be run on each of the virtual machines without interacting with each other. Different, incompatible software versions, libraries and even operating systems can be packaged up as a virtual machine and run concurrently on the same physical machine. Virtual machines can be started or stopped by a hypervisor within minutes which enables rapid repurposing of the physical machines' resources once a virtual machine is no longer needed.

The operating system and applications used by a virtual machine are usually stored on virtual disks. These take the form of a file or archive containing all of the files which would be present on a physical version of the disk and are referred to as Virtualised Disk Images (VDI). This file can be loaded by the hypervisor and presented to the virtual machine as if it were a physical disk. On starting a new virtual machine on a hypervisor, the virtual disk which is loaded defines all of the operating system and software used by the machine. For example, if a virtual disk containing the Linux operating system is loaded, the virtual machine will be Linux based, if a virtual disk containing Windows is loaded, the virtual machine will be Windows based. This means the virtual disk image defines a lot of a virtual machine's function: it is thus a blueprint for the running virtual machines.

1.7.2 Containerisation

Containerisation is a method of operating-system level virtualisation. It began as a concept in Unix-based operating systems (Ritchie and Thompson, 1978) which separates individual applications from the default structure of an operating system of a physical machine. A container is one or more software packages wrapped up into a separate, specially formatted package. When a container is executed, this package structure is opened up as a virtualised environment within the operating system. This environment has limited access to the rest of the computer, though select files and resources can be shared with the environment determined by the operating system. In this way, containerisation creates an area which is separated off from the standard execution area on the file system, in which software dependencies and data sources can then be allocated on a per-container basis. However, containerisation does not necessarily compartmentalise access to Random Access Memory (RAM) and physical devices (Turnbull, 2014).

Containers can enable a much more rapid deployment of a virtualised environment than even virtual machines, as the bulk of the operating system is not duplicated. Running an application within a container can be faster than hypervisor-based virtualisation, because this duplication by the virtualisation software can be slow (Beserra et al., 2015). Stopping or starting of a container is substantially faster than a virtual machine, taking only seconds. Containers are noticeably less secure, however, due to the lack of hypervisor and lower degree of isolation from the physical machine. The first stages of containerisation technology were built upon LinuX Containers (LXC: https://linuxcontainers.org/lxc). In recent years a second wave of implementations have been built. Two examples of these are Docker (http://www.docker.com) and rkt, which is part of the CoreOS operating system (http://www.coreos.com). Throughout this thesis the discussions on containerisation primarily

relate to Docker as this is the most mature of the available implementations at the time of writing.

Containerisation is a newer and currently far less mature technology than hypervisor based virtual machines. This immaturity means many functions commonplace in virtual machines are currently not available in containers which hinder its use in computing systems (Mouat, 2015). An illustrative example of this is system security. Security of computer systems is often challenging due to the many interacting layers of software and hardware. Developing technologies can often suffer problems with security due to oversights in design and bugs which will have been identified and eradicated in more mature technologies. Containerisation with Docker or rkt is significantly less secure than virtualisation. There are several recognised areas of weakness for containers (Mouat, 2015). These are:

- Kernel Exploits: The Kernel is the core of a Unix-based operating system which controls things such as interaction with hardware. The host operating system and all the containers share the same kernel, so any vulnerabilities in the kernel could allow exploitation of the host computer. Virtual machines use a separate kernel for each instance thus isolating this problem.
- Denial of Service (DOS) Attacks: Containers can monopolise resources such as disk access, blocking others from functioning. This is possible to some extent in virtual machines. However, as all resources are managed by the hypervisor, it can be limited.
- Container breakouts: A breakout is where a process running inside a container manages to escape the limitations of the container and gain full access to the physical hardware. Should an application be able to break out from a container using a kernel exploit, it will have all the administrative permissions it did inside the container. Allowing an application within a container to have full administrative permission is high risk. Breaking out of the environment is far more difficult to do within a virtual machine because management is done through a separate hypervisor application.
- Compromising Secrets: As all applications within a container share the same kernel and resources, sensitive information such as keys and passwords are potentially accessible by other containers. This is not the case with a virtual machine.
- Poisoned Images: It is important that the creator of a container is trusted. Containers loaded from public repositories can exploit any of the weaknesses listed here. The current default for frameworks such as Docker is to load untrusted containers from public repositories.

As virtual machines are less susceptible to these attacks, they are currently more suitable for running untrusted software. Many of the problems identified above can be addressed manually with technologies such as SELinux (Smalley and Loscocco, 2001), resource monitoring tools and extensive configuration. This example shows, however, that containers are still a relatively immature form of virtualisation in comparison to virtual machines. A lot more configuration and technical knowledge is required to install and configure them. In the future, containerisation is likely to be a viable alternative form of virtualisation but currently the additional complexity and security concerns outweigh the relatively little benefit of increased speed.

1.7.3 Cloud & On-Demand Computation

High Performance Computation (HPC) is computation of a category of problem which require amounts of computational power too large for a single computer. Commonly, large data centres of computers are used to perform a specific task or set of calculations. These groups of computers are often referred to as a computational resource.

The rise in availability of both virtual machines and containerisation has enabled a new model of access for HPC. A variable sized part of a large machine can be isolated using a virtual machine or container system for use by a user. As they require more or less resource, this can be increased or decreased without affecting other users or the system. Providers of large computational resources have used virtual machines and containers in this way to provide scalable, on-demand computation; this is commonly referred to as cloud computing.

Cloud platforms provide a way of instantly accessing remote computational resource using a standardised structure. The cloud is accessed by control software or a well-documented API over a private network or the internet. Cloud platforms generally allow users to select from a set of pre-defined disk images or load their own custom images. This provides a great deal of flexibility over the software and operating systems which can be run.

Generic control combined with increasing speeds of networks has meant the physical location of the cloud and its underlying hardware is less important than on traditional computational clusters. This allows large amounts of generic computational resource to be hosted by organisations and used by a large group of users. This scale—and the ability to repurpose the hardware within minutes with virtualisation—reduces the upfront cost of using large computational resources. There are a range of generally available commercial clouds such as Amazon Web Services (AWS: https://aws.amazon.com), Google Compute Engine (GCE: https://cloud.google.com/compute) and Microsoft Azure (https://azure.microsoft.com).

These systems are commonly referred to as public cloud due to their publicly available status. In addition to this there is a range of software to set up clouds on physical hardware such as OpenStack (http://www.openstack.org, Sefraoui et al., 2012) or Apache CloudStack (https://cloudstack.apache.org, Sabharwal and Ravi Shankar, 2103). These enable self-run and managed clouds privately or within an organisation and are commonly referred to as private clouds.

The widely available and generic platforms provided by virtualisation and clouds remove much of the variability that would otherwise be associated with specific hardware. It is relatively quick and easy for anyone to specify an amount of hardware resource, then start and run a virtual machine. This virtual machine can be started any number of times and will run in an identical manner to every other time it has been run.

1.7.4 Virtual Disks on Bare Metal

Bare Metal computation is the running of an execution environment on a physical machine without any virtualisation layer. This is supported by a number of cloud providers such as Packet (http://www.packet.net) or SoftLayer (http://www.softlayer.com). These companies provide an API to the physical hardware, similar to the API of other virtualised cloud environments. The bare metal machine can be started with a selection of virtual disks with operating systems and software used by the machine. A common API such as this allows users to select and launch a physical or virtual machine in the same way depending on their preference.

Where hardware variation is not a consideration, the overhead of running virtualisation can be eliminated through using a bare metal machine. Some applications may themselves make use of elements of virtualisation. These applications can also be run within a virtual machine and this is referred to as nested virtualisation (Ben-Yehuda et al., 2010). Nested virtualisation can often be complex to configure, and requires specific hardware to enable it. Using a bare metal execution environment avoids this complexity.

A common method of implementing this automated bare metal virtual disk deployment is with a Pre-eXecution environment (PXE) image deployment. PXE is an option provided by most modern computers which allows them to use a network device to prime the start-up procedure. When a machine initially starts up or ‘boots’, it looks for an operating system on the boot device; this is a hardware resource from which an operating system kernel can be loaded. PXE allows the boot device to be defined by a Dynamic Host Configuration Protocol (DHCP: Droms, 1997) server connected to one of the network devices. DHCP is a standardised network protocol for distribution of network configurations to computers on the

network. During a PXE boot, this DHCP server gives the booting computer the location of a Trivial File Transfer Protocol (TFTP, Sollins, 1992) server from which the operating system and boot script can be loaded. TFTP is a very simplified file transfer protocol for communication of small files without the overheads of a more complex system. The booting computer then downloads this script and operating system and starts up the computer with them.

This strategy of deployment of standardised images through PXE is also employed by containerised environments such as CoreOS (https://coreos.com/os/docs/latest/booting-with-pxe.html) whereby a minimal operating system is loaded to RAM. This minimal OS is then used to start containers.

1.7.5 Virtualisation Resources

Virtualisation has become a widely used tool with a wealth of available resources. Both virtualisation and containerisation systems have community-moderated, published databases of virtual images. These images give researchers access to analysis tools and environments without the difficulty of installing and managing dependencies and toolchains themselves. Docker Hub (https://hub.docker.com) is an application database for Docker containerised applications. There are several online databases hosting virtual disk images of various types by Oracle, Amazon and many of the other virtualisation platforms. For Oracle VM VirtualBox, for example, there is the VirtualBoxes database (http://virtualboxes.org). There are limited bioinformatics resources available, although several projects have started to utilise the benefits of containerisation. Nucleotid.es (http://nucleotid.es) is a catalogue of genome assembly tools packaged in Docker containers. The aim of the project is to assess and benchmark the wide number of sequence assemblers available in a reproducible manner. Statistics are provided for each tool and the assemblers themselves are available to download in their containers which provide easier accessibility to the tools. Bioboxes (http://bioboxes.org, Belmann et al., 2015) is a resource of bioinformatic tools available to download and use in pre-packaged containers. This again aims to simplify the download and local use of bioinformatic tools. These libraries provide tools packaged in standard virtual disk images which could be loaded into a variety of virtualisation-based systems.

1.8 Summary of Introduction and Hypothesis

The current state of reproducibility within published studies is poor, leading some commentators to refer to a state of “crisis of reproducibility” (Baker, 2016). As has been discussed, a significant percentage of this reproducibility problem is due to inadequate reporting of tools and methods. It is not enough to simply name the tools used in an analysis

in a publication (Duck et al., 2015). Capturing the analysis environment itself and providing this in a published study, as is common with source data, would eliminate these problems.

There are several tools currently available which would allow researchers to address some of the problems of bioinformatic reproducibility. As summarised in the table in **Table 1**, a combination of these tools could be used to capture some of the sources of potential discrepancy in a study, but this does not account for it all. At the moment, there is no one system, or combination of systems, which allows the complete reproduction of a study despite significant community efforts. To address these problems, this thesis documents the implementation of a new system which brings together the existing state of the art, and newly-created elements where these are insufficient or not available.

Source of Potential Discrepancy	**Existing Best Solutions**	**Effectiveness**
Analysis Methods	Recorded in workflow systems	High
Random Seeds	Include setting the random seed in the methods	High
Operating System and Hardware Variability	Virtualisation & cloud computation	High
Software & Versions	List tools in methods	Low (Duck et al., 2015)
Software Libraries and Dependencies	Unaddressed	None

Table 1 - Summary of The State of The Art in Analysis Reproducibility: A table summarising the conclusions of section **Reducing Potential Variability in a Bioinformatic Analysis**. The table shows the sources of potential discrepancy in a bioinformatic study, the current state of the art in dealing with this and the relative effectiveness of this solution.

The reproducibility of a bioinformatic analysis could be dramatically improved through greater recording and storage and then communication of all the elements of the analysis (Garijo et al., 2013). This thesis describes a system which combines and builds upon existing workflow systems to enable the full software ‘stack’, analysis parameters and all other information related to a study to be recorded and documented. In order to build upon an existing workflow system, GeneProf, will be used as a case study in this thesis. The software innovations performed on GeneProf in this work should, however, be applicable to most workflow systems. Where this is not the case, it will be highlighted as such. This work

enables future researchers to have full access to the analysis stage of a study, enabling them to investigate its results without the time and cost of working out how to reproduce it.

The remainder of this thesis will address the question: how can we use and extend existing tools and systems in order to incorporate the ability to fully record and share an analysis? Whilst addressing this, it will also investigate if it is possible to substantially reduce the burden of reproducibility for the publishing researcher through automation, and if it is also possible to similarly reduce the burden of reproducing a bioinformatic analysis for a second party. This work will be divided into the design and implementation of a system to capture and communicate the elements of an analysis. Using this system there will then be an evaluation of this method in comparison to existing methods.

2 Methodology and Experimental Design

As discussed in **Chapter 1**, a major barrier to the reproduction of a published study is the difficulty of adequately sharing bioinformatic methods. This chapter details the development of Cumulus; a method for capturing and enabling the reproduction of a bioinformatic analysis using a virtualised environment. In this chapter, the methodology behind the design of this software and the decisions which led to the design will be detailed.

2.1 Software Licensing

When software is written and then published it is generally released under a software licence which describes the terms under which it can be used. The terms applied in these licences can vary widely, but one commonly used category is Free and Open Source Software (FOSS) licenses. Software licensed with a FOSS license allows users to use and re-distribute the software without charges or license fees (St. Laurent, 2004). FOSS licenses which comply with the definition of Open Source, in that they allow software to be freely used, modified, and shared are managed by the Open Source Initiative (OSI: https://opensource.org). The OSI publishes a list of licenses for open source projects to use as standard with their software (https://opensource.org/licenses). As the aim of this work is to produce a system which enables reproducibility, it is important that third parties are able to use and integrate this work into their own software and systems. This may not be possible if a licence requires a payment or terms which a researcher is unable to accept. As such, this work will focus on using software libraries and tools which fall under this FOSS category.

2.2 Capturing the Sources of Potential Variability in an Analysis

In order to fully reproduce a bioinformatic analysis, the scientist reproducing the analysis must be able to entirely re-create the environment in which the analysis took place. **Section 1.5** of this thesis discussed the sources of potential variability within a Bioinformatic Analysis. Each of these sources of potential variability, must be captured and then communicated to the scientist attempting to reproduce the study. In addition, the scientist attempting to reproduce the study must have some way to take this information and use it to reproduce the same environment. The values communicated during this process will take the form of a set of configurations, software and data files which the scientist attempting to reproduce the study will then use to create their analysis environment. The value describing an element of potential variability will be referred to as an ‘experimental descriptor’ and the set of experimental descriptors fully describing an analysis as the ‘full experimental descriptor set’.

Of the sources of potential variability in an analysis, the recording of methods can already be addressed adequately by workflow-based systems. Similarly, virtualisation tools are already available which have the ability to standardise potential variation in hardware. Both workflow and virtualisation systems are highly complex and can take teams of software engineers many years to create (Afgan et al., 2016). Where possible this project will combine and augment existing software tools and systems and add missing additional functionality required to fully record an analysis.

In order to capture the full experimental descriptor set, an existing workflow system will be augmented with additional software to allow it to capture all of the experimental descriptors. The software created—Cumulus—will be an independent piece of software which will not only provide analysis tools for use through this workflow system but also record everything the researcher does with them. The record of the analysis can then be published with the resultant study. This will enable both the specification and communication of the full experimental descriptor set to the scientist attempting to reproduce the study. In addition, it will enable a scientist to take a descriptor set defined by the system and re-create the experiment and analysis environment defined by the reporting scientist.

2.3 Augmenting a Workflow System

This project will use the GeneProf system as a model of a workflow system and an example on which to build. The concepts developed here are generic, however, and will be applicable to other analysis systems. Any additions which are specific to the GeneProf system will be highlighted as such.

The software stack of a bioinformatic workflow system such as Galaxy or GeneProf follows a common structure. The workflow system is a web-server based application which interacts with a set of analysis tools installed on a base operating system as shown in **Figure 6**.**A**. The analysis tools and web server components are often run on different physical machines in order to enable the use of multiple computers and to isolate the impact of the analysis from the web interface. Some workflow systems can interact with their analysis tools through an API to an external computational infrastructure. This may be a Cloud or virtualisation API which allows them to use the functionality and scale provided by a Cloud environment as shown in **Figure 6**.**B**.

The additional areas of potential variability within an analysis which the Cumulus system will aim to capture are the names and version numbers of the operating system and analysis software used, together with all their respective dependencies. In order to capture this

information, Cumulus will require access to the analysis tools and the operating system on which they are run. In order to obtain this access, Cumulus will be an addition to the existing analysis structure within the workflow system. Cumulus will provide an API to request computational resource; it will then, in turn, make that request to a cloud or virtualisation provider. Whilst relaying these requests, it will record the interactions, the versions of software, the dependencies and the disk image used in the analysis as shown in **Figure 6**.**C**.

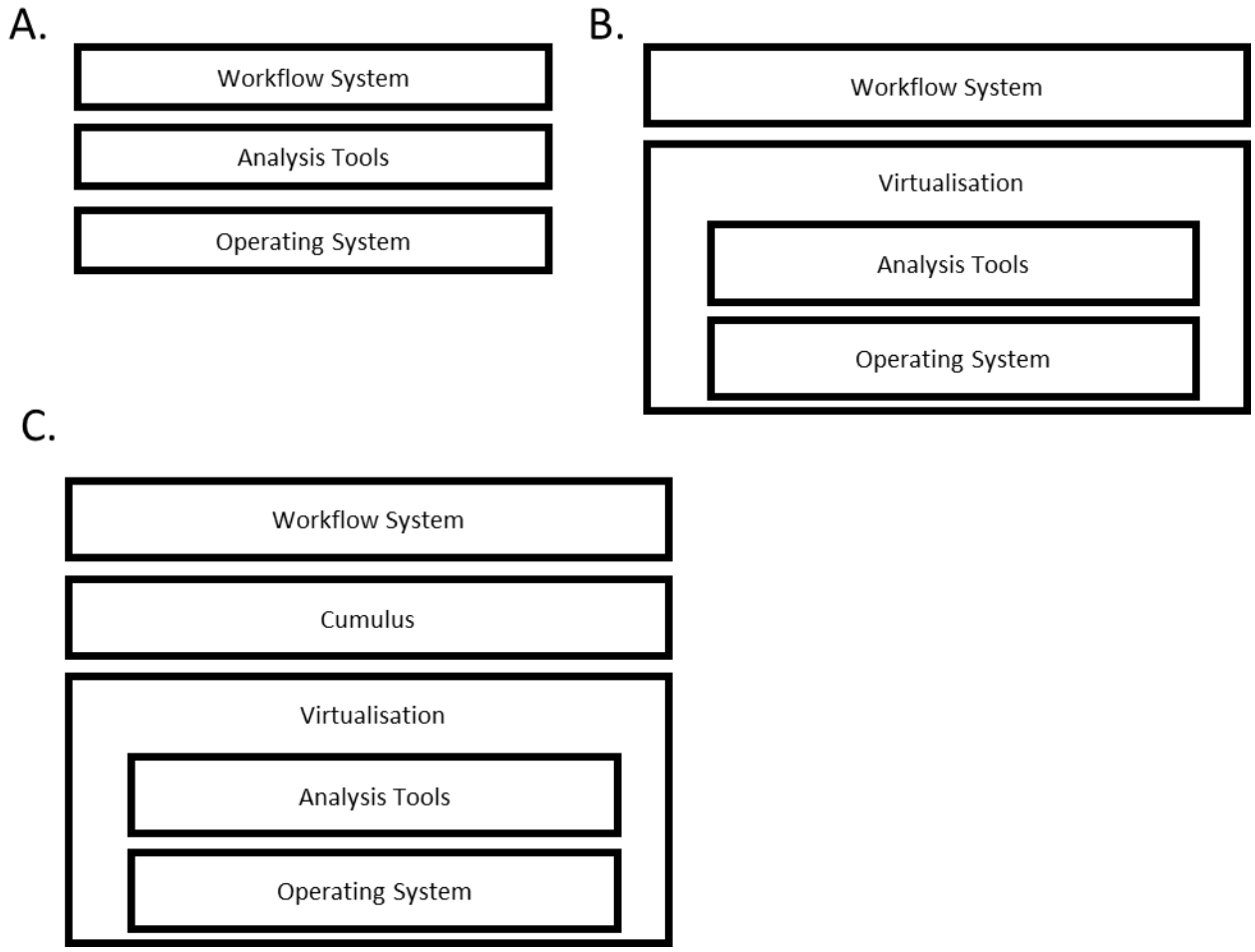

Figure 6 – Analysis Structures of a Workflow System: Three stack diagrams showing different workflow analysis system structures. (**A**) a simple workflow system structure, such as that used in the GeneProf system, in which the analysis tools are installed directly on a host operating system. (**B**) A workflow system which interacts with analysis tools on a cloud or virtualisation platform. (**C**) The proposed Cumulus structure, the workflow system interacts with an intermediate system, Cumulus, which then orchestrates the analysis on a cloud of virtualisation platform.

Adding a component to the analysis structure between the workflow system and computational infrastructure in this manner allows a separation of the concerns (Mitchell, 1990) of workflow, the method of analysis and the recording of this analysis. This allows software components to be recorded whilst minimising the impact and required changes to

the workflow system itself. In addition, a potentially wider range of workflow systems could be supported with minimal difficulty.

The end system will allow the user to share the data from each stage of the analysis workflow including all the methods, a list of all the software and dependencies used and a virtual disk image of the operating system which can be used to repeat or re-run the analysis.

2.4 Virtualisation

In **Section 1.7** it was noted that virtualisation is the main technology used for reducing the software variability associated with different operating systems and hardware. In addition, as discussed in **Section 1.7.5**, virtual disks are commonly used to manually package up an analysis tool and its environment in order to share it with the community. A workflow system may support hundreds of different analysis tools, often with multiple different versions. As a workflow system grows, it needs to be able to add new tools without disrupting those already installed. As discussed in **Section 1.6.1**, tools such as the Galaxy Toolshed manage these additions within a single analysis environment. This strategy can, however, hit the problem of conflicting dependencies. These problems make the process of installing and managing a single analysis environment highly complex. In order to reduce this level of complexity, the system will manage each analysis tool in its own virtualised environment. Wrapping each tool up in this manner will isolate the tools from conflicting with each other. A set of virtual environments may then be shared as a chain to define an analysis. **Section 1.7.1** detailed virtual machines and **Section 1.7.2** detailed containerisation, the two most commonly used virtualisation technologies. As was highlighted in **Section 1.7.2**, containerisation is currently an immature technology, lacking a lot of the functionality of virtual machines. The software created here will therefore use virtual machines as it's implementation of virtualisation.

Running each analysis tool within its own virtual machine reduces the complexity of each individual analysis component but increases the scale of difficulty in running and sharing a workflow. Whilst a researcher may manually be able to share a single virtual disk manually on publication, picking out a set of virtual machines containing tools they have used out of growing set of potentially thousands is a more onerous task. To facilitate this, the system will manage the virtual machines in a database and automate the process of running and sharing them. In order to run and share a workflow, the system will automatically select the correct set of tools used in the workflow which would then be run and shared *en-masse*. To do this, virtual machine management and a database of virtual disks will be incorporated into Cumulus as new components which will automate the process of creating and managing virtual analysis environments.

In order to create and manage an analysis environment using virtualisation, the Cumulus system needs some way in which to interact with hypervisors and virtual environments. There are many open source and commercially available virtualisation environments. Commonly used examples include the Xen project (http://www.xenproject.org, Barham et al., 2003), Linux Kernel based Virtual Machine (KVM: Kivity et al., 2007), VMware (http://www.vmware.com), QEMU (http://qemu.org, Bellard, 2005) and Oracle VM VirtualBox (https://www.virtualbox.org). In cases such as this, where there are a number of very different implementation choices, it is standard software development practice to loosely couple the components through an API (Yourdon and Constantine, 1979). In this instance, the scheduling component within Cumulus will connect to a Virtual Container API; a standardised interface for provisioning a virtual machine. This will connect out to the chosen virtualisation environment and provision the machine, as shown in **Figure 7**.

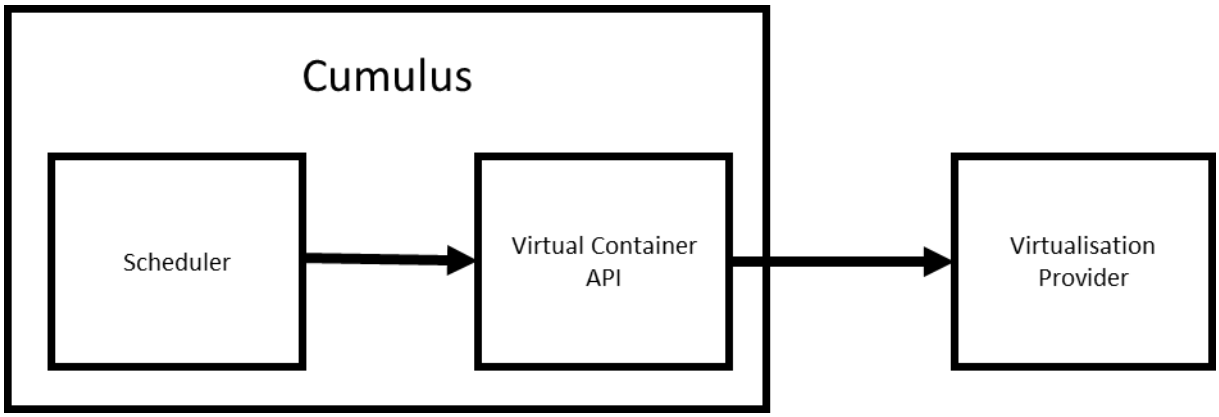

Figure 7 - The Cumulus Virtualisation API Structure: A flow diagram showing the communication structure between the APIs of the Cumulus system when a virtual machine is requested. The Scheduler API requests a virtual machine from the internal Virtual Container API. The Virtual Container API then converts this request into a request to an external virtualisation provider such as a cloud or hypervisor.

As previously mentioned, this work will only consider software licensed with a FOSS license. This license restriction eliminates VMware which is generally closed source and requires a license fee in most circumstances (Nie, 2013). QEMU can be used in conjunction with other virtualisation systems such as KVM or to provide hardware emulation to allow code compiled for a different target processor to be executed. This emulation of different physical hardware is a feature which is not supported on the majority of other virtualisation platforms. As discussed in **Section 1.7**, when functioning in this manner it can make QEMU significantly slower than the other virtualisation platforms. As the vast majority of computing systems and bioinformatic software is made for Intel x86 processors, this emulation is not needed and QEMU is therefore less suitable.

Xen can support a larger number of virtual machines per physical machine than KVM or VirtualBox as it has a highly-optimised structure and custom Linux kernel. It can, however, be complex to install due to the high levels of configuration that it requires and the fact that it needs a full replacement of the system kernel (Freet et al., 2016). Xen is enterprise-level software which requires substantial expertise to install and maintain. As the system being described in this thesis is an experimental work as opposed to a production system, the benefits of scale it provides are unlikely to be of use. Both KVM and VirtualBox are more lightweight with a very simple and streamlined installation and support by comparison (Freet et al., 2016).

KVM is part of the core of Linux and therefore restricted to running under a Linux environment. This is not true for VirtualBox, which has support on most modern operating systems. Cross platform support is useful in a system promoting reproducibility. The computing environment available to the researcher reproducing the study is an unknown quantity so the support of a wider variety of operating systems is beneficial. We therefore chose Oracle VirtualBox as the virtualization environment for initial implementations of the system. Note that the loosely-coupled structure described above will allow alternative solutions–such as KVM or Xen–to be implemented in the future should Oracle VirtualBox prove unsuitable or should the alternatives provide additional benefits or features to address future use-cases.

2.5 Clouds

One possible design for this system would be to use a cloud provider to supply the mechanism for virtualisation. Fundamentally, a cloud provider is an API in front of one of the virtualisation technologies listed above. The advantage it offers is that of scale. Many hundreds or thousands of virtual machines can be spun up simultaneously without the difficulty of physically managing that amount of server hardware. As this is an experimental work, this level of scale is unnecessary. In addition, a generic implementation which can be run in a variety of environments is much more use in aiding reproducibility between researchers who may have other computing environments and limitations. Relying on a specific cloud which may not be available to different researchers is unnecessary and potentially creates more problems. If, in a future production environment this kind of scale is required, the virtual container API could be extended to use libraries such Apache jclouds and interface with different cloud providers.

2.6 Application Data and Experimental Data

Application data is the code, compiled binary files, software dependencies and resources required to run an application or set of applications. This is distinct from experimental data which is the captured data from the experiment undergoing analysis. As is standard practice with a virtual machine, the virtualised application data will be stored on a virtual disk. The virtual disk is loaded and opened (mounted) whenever the virtual machine is started and the applications are required. This virtual disk can then be shared with the published study as an element of the experimental methods.

2.6.1 Virtual Disks

A virtual disk format is the format in which the analysis tools application data will be stored for access by the analysis virtual machine. There are a number of technologies available for disk virtualisation such as Virtual Hard Disk (VHD), Virtual Disk Image (VDI), Virtual Machine DisK (VMDK), with each offering functionality similar to the others. Historically, each of the virtualisation hypervisor implementations had its own virtual disk format, but most modern hypervisors now support a range of different disk formats, VirtualBox, for example, supports VHD, VDI and VMDK in addition to a number of others (Dash and Blaminsky, 2013). In addition to these virtualisation formats, several other file system formats are used for the exchange of disks in other situations such as raw image (img) and ISO 9660 disk format (http://www.ecma-international.org/publications/standards/Ecma-119.htm). These formats are widely used for operating systems and Linux 'live' disks and can provide a read-only bootable operating system.

The aim of the system is to produce a reproducible experiment. In this regard, the standards used in the experimental descriptors are important. The standards selected should be as compatible and interoperable with as many different tools as possible. If standards which limit this interoperability are used, it will be more difficult for the scientist attempting to reproduce the study to interrogate the experiment and its results. To this end, there are several advantages to the ISO 9660 format. As the default standard data file system for the Compact Disk Read Only Memory (CD ROM) it has widespread use and there are several tools available for working with the format. Operating system distributions commonly offer download in this format. In addition, the use of this format enables users to download images from Cumulus and boot these images directly from their own system either by burning them to a CD or writing to a commonly available Universal Serial Bus (USB) device (Sivaiah et al., 2014). Because of this, the system virtual disk images will use the ISO 9660 disk format.

As the ISO 9660 format has widespread use, an abundance of tools are available to create and work with its files. Most major Linux distributions offer releases using the ISO format and tools such as the Ubuntu Customization Kit (http://uck.sf.net) and Customizer (https://github.com/kamilion/customizer) allow the user to create and modify the distributions. These tools provide a simple workflow where the user can download any operating system ISO they require and log in to it. Any tools or software required can be installed or unused packages removed. The user can then exit the environment and the ISO is closed and re-packaged. This ISO can then be uploaded and registered on the Cumulus system.

Several projects exist which provide minimal Linux distributions. These are distributions which have had all but the core software and resources removed. Examples of these include: Tiny Centos, Ubuntu Mini Remix (https://launchpad.net/ubuntu-mini-remix), Tiny Core Linux (http://www.tinycorelinux.net) and Alpine Linux (https://alpinelinux.org). Minimal distributions can provide a good blank canvas on which only the software packages required for the specific analysis software are installed. Having only a core set of software reduces disk size and removes any unneeded complexity. Cumulus makes use of Ubuntu Mini Remix for its core set of tools.

As time passes and new iterations of the system's analysis tools are released the software on virtual disks will require updates. In order to do this, the user will need to mount an existing ISO, update the software and then re-submit it to the system. As disk images are not overwritten by the system, the versions of all of the images going back will be retained. When a user starts a new analysis, they can use the new version of a piece of software but the old will still be available.

2.6.2 Shared Experimental Data: Stembio Drive

Whilst virtual disks are suitable for application data, they are, however, less suitable for experimental data. Experimental data can be very large; often many gigabytes or terabytes in size. Virtual disks are generally copied for each virtual machine launch and storing this data on a virtual disk substantially slows down the launch speed of a virtual machine. In addition, application data stays relatively static, only being required during a specific part of the analysis of a study. By contrast, the flow of experimental data around a system can be quite complex. Each analysis step may require a number of different sources of data and produce a number of results. These may in turn be required by other experimental steps. This data flow is captured and managed by workflow applications which enable different analysis steps to be connected together. If this data were stored within a virtual disk, there would be the increased complexity of mounting and extracting the data from the virtual disk volume each time it is required by an analysis tool or the workflow system.

In addition to this data flow during an analysis, experimental data may need to be accessed and inspected at different stages, after the analysis has taken place. This may be for Quality Assurance (QA) purposes or to make graphs or visualisations in order to better understand the results. Additionally, the researcher may be required to submit the data to a third party such as a repository on publication of the study. Storing it in a closed virtual disk is unsuitable for these requirements.

An approach used in cloud-based environments such as AWS or OpenStack is to use a centrally-managed volume or object store—separate from the data processing structure—to hold user data. This is the model used by Amazon Simple Storage Service (S3) or OpenStack Swift. Users can import or upload source data from their own computer or external resources to this service. All of these resources can then be accessed by all the components of the system to access or deposit experimental data. Equally, the outputs of analysis are available to download to the user's computer or to share on to external services.

One potential problem with this separation of data is that the system should maintain a link between the experimental data and the application data to ensure all components of an analysis can be related to each other. This relation enables the correct disk images and data files to be published and reported for the correct experiments. In addition to this, once an experiment has been run and published, the data should no longer be editable. This is currently a feature of GeneProf which any new system should aim to maintain.

There are several technologies available from which to create a highly available and scalable file storage system. Examples of these are Hadoop Distributed File System (HDFS: http://hadoop.apache.org, Shvachko et al., 2010), Ceph (http://ceph.com, Weil et al., 2006) and Gluster File System (GlusterFS: https://www.gluster.org). These technologies provide large file stores which can be scaled up as data is added. They do not, however, provide higher-level data management features such as integration with workflow systems, a user interface to upload files or user-based access control. In terms of creating a reproducible system, the selection made for the underlying technology used for the actual storage is less important than the software for managing and connecting this storage within the workflow system. The work described here will concentrate on the data management level software and assume this is installed upon a scalable filesystem such as those listed above.

The requirements of a data store for a system such as Cumulus are: to store many terabytes of data, to move potentially large files to and from a virtualised environment, to lock specific data files so they cannot be edited, to be interoperate with a workflow system and to allow researchers to upload a variety of files and formats. There are a number of software packages with FOSS licenses which provide data management level access to cloud-based

storage. Examples of these are: OwnCloud (https://owncloud.org), Pydio (https://pyd.io), SeaFile (https://www.seafile.com) and sparkleshare (http://sparkleshare.org). These are all smaller scale applications and do not reliably handle the required size of file or communication protocols suitable for data analysis. The applications which do provide an API for programmatic access (OwnCloud) use the Web Distributed Authoring and Versioning (WebDAV) protocol (Network Working Group, 2007) for the retrieval and storage of files.

WebDAV is built upon HTTP to enable web content authoring and the communication of large files on the internet. Unlike protocols such as NFS, WebDAV does not allow for partial upload of files (Dusseault, 2004). NFS allows a small part of a file to be downloaded, viewed and edited (Shepler et al., 2003). However, WebDAV requires that the file is entirely downloaded and edited in-situ before being entirely uploaded again. This can become very slow for large files and requires the editing machine to have a large amount of cache available to store the intermediate files. In addition, should a failure occur during upload, the entire file must be re-sent. This makes WebDAV less suitable for editing large files than protocols such as NFS or CIFS (Dusseault, 2004).

In order to meet the experimental data storage requirements of such a system, a new component, Stembio Drive, will be implemented. This will enable both user-level access for upload and API-level access to allow the workflow system and analysis tools to interact with the study data. A single Network Attached Storage (NAS) style volume will be connected to by a web interface and all of a user's virtual machines. This NAS will implement communication protocols such as NFS and CIFS which are suitable for large file sizes and reliability.

Once analysis has been started on a data file, this will be moved to a protected area of the drive to avoid the user deleting or editing the data. The operating system of each users' virtual machine will mount their area of the drive as a block storage device. This allows the virtual machine to treat the Stembio Drive as if it were a local hard disk. As each analysis tool is then executed the inputs will be loaded from this drive and the resultant files written back. All of the files on the drive will be available for the user to download or for another virtual machine to access for the next stage of the analysis.

2.7 Indexing Experimental Software

New analysis tools are constantly being developed and updated. In order for the analysis tools within a workflow system to stay relevant, it is important that the system can be updated with these new additions. When a new or updated analysis tool is added, the tool and environment in which it runs need to be recorded. Recording the environment allows

other researchers to dissect the stack of software and dependencies used in a specific analysis and reproduce it. One analysis environment can consist of many hundreds of libraries, making the process of recording the versions and all the dependencies an arduous task. In a system in which new analysis tools are constantly being added, this can very rapidly become unfeasible. To address this issue, the system will automate the process of recording a new analysis tool and its environment.

The ISO 9660 disk images will first be uploaded to the system by an administrator and then extracted. As each virtual disk is imported into the system, a database of analysis tools within Cumulus will be populated with the software libraries and versions of all the installed packages on the disk. This will allow the system to select images which contain required pieces of software and enables the production of a report about the disk images and software used after any analysis. The system will do this by indexing all of the available software packages from the package manager database inside the image file. This could be a flat file database for a Debian package (DPKG) or a structured database for Red Hat Package Manager (RPM). The package manager database will be parsed into an array containing package name, description, project URL and version. For each record in this array the system will search for an existing package by the same name and if none exists it will create one. It will then search the existing versions of this package and again if none exists it will create one. Finally, it will link the software version to a software install record linked to the record for the disk image. This populates the software package database with each package linked to any existing software packages in the system. In addition, it will give both a structured tree view of a single image and also a stratified view of all the images linked by their shared packages and versions. This automatic loading of the dependencies and integration into the database cuts down on any manual curation or data entry of packages. There may be tens or hundreds of thousands of pieces of software in any particular operating system; Gentoo Linux for example has a reported 63988 dependencies between 14319 software packages (Uys, 2011). It would be unfeasible to index all of these dependencies manually.

With this system of automated indexing of packages on a virtual disk, a disk can be downloaded from Cumulus, the software updated, and the disk re-submitted to Cumulus. Cumulus will automatically be able to identify which parts have changed and how. An image can have many thousands of changes between versions, quite often very subtle. Manually tracking each of these changes over many hundreds of disk images would be time consuming and unfeasible without this kind of automation.

2.8 Bare Metal Computation

As previously discussed in **Section 1.7.4**, not all analysis tools are suitable for virtualised environments and bare metal servers are often used as an alternative to virtual machines for these types of workloads. In order to run these kinds of applications, the ability to write virtual disk images to a physical machine will also be added to the system. As detailed in **Section 1.7.4** the most common method of automating the start of a physical machine is through the use of a PXE. The system will use this PXE environment to re-write the computer's hard drive during boot whilst it is not in use by the operating system. The deployment process for this is shown in **Figure 8**. It will do this by hosting its own DHCP and TFTP server to control the first two stages of the PXE boot. The physical analysis machine will connect to the Cumulus DHCP server at boot and get forwarded to the Cumulus TFTP server hosting the boot script. Cumulus will then determine if the machine has had a re-write requested by a user. If not, and there is no rewrite required, then it directs the computer to a boot script which will start the computer normally. Otherwise the boot script it gives the machine will re-write the local disk. The script will download the correct Cumulus image from the image database to the local hard drive. It will then extract this to the machine's hard drive and the computer will be restarted with the new operating system. This allows Cumulus to provide an interface for the remote control and automatic deployment of virtual disk images to bare metal analysis computers.

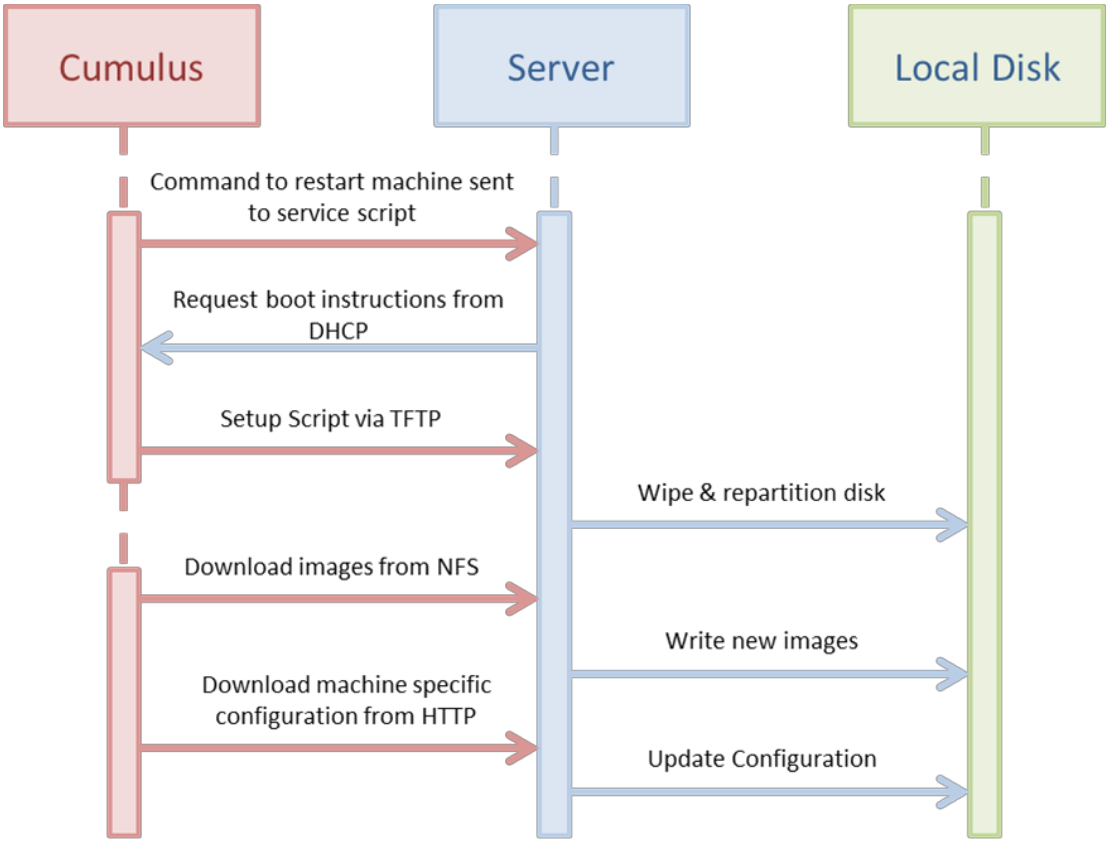

Figure 8 - The Cumulus PXE Disk Bare-Metal Deployment Process: A sequence diagram showing the stages of Pre-Execution Environment (PXE) deployment by Cumulus of a virtual disk image to the local disk of a server. The red arrows indicate actions and the flow of data from the Cumulus services. The blue arrows indicate actions initiated by the server.

2.9 Networking

The networking and connectivity between the elements of a virtualised or cloud computational system can be a highly complex topic (Denton, 2014). This system will aim to reduce and avoid these complexities wherever possible. The aim of this work is to investigate the creation of a system which enables reproducible analysis, not to reproduce the full functionality of a cloud. This is a major area of complexity for cloud providers and as it is outside the direct scope of this project will not be considered. Thus, connectivity between virtual machines beyond internet communication will not be implemented.

2.9.1 Network Setup and Configuration

In order to control certain elements of the virtualised operating system, Oracle VM VirtualBox provides an additional kernel module called the Guest Additions (Dash and Blaminsky, 2013). This kernel module must be installed into the operating system of the running virtual machine. Among other things, these Guest Additions enable VirtualBox to obtain the IP address of the running virtual machine. The Guest Additions are a proprietary product and

not licensed using a FOSS software licence. As described in **section 2.1**, this project will only use software licensed as FOSS, and so, they will not be used. As a replacement, Cumulus will maintain its own DHCP server to provide IPs to virtual machines as they are provisioned.

When provisioning a machine with VirtualBox, the system will specify a Media Access Control (MAC) address for the network card. The virtual machine is then booted and automatically starts making DHCP requests. These requests will be detected by the Cumulus DHCP server which will respond with a pre-determined IP address. Cumulus keeps a record of the IP address in the database and uses this to connect once the virtual machine has started. This mechanism allows Cumulus to maintain a mapping between the virtual machine and its network address.

To control the virtual machine, the system will need to connect and execute commands. There are several protocols which enable this such as Telnet (Carr, 1969; Postel and Reynolds, 1983) and Secure Shell (SSH - Ylonen and Lonvick, 2012a). Secure shell has become the de-facto standard for remote execution for Unix-based operating systems Linux and OSX as well as Microsoft Windows (Calvo, 2015). While SSH is most commonly used for the control of a remote machine through execution of commands, it also supports tunnelling and forwarding of ports and X Window System (X11) connections. In addition, it can transfer files using the associated SSH file transfer (SFTP) or secure copy (SCP) protocols. Because of these advantages the system will use SSH as the main channel to communicate and control a virtual machine.

In computer networking, a port is a component of a network address which allows a single network address to host multiple differentiated services. A port is a 16-bit number between 0 and 65535 with numbers below 1024 often restricted to use by only privileged users within Linux operating systems. Specific well-known services are often assigned to common port numbers such as HTTP to port 80. This allows predictable defaults so that the port number need not always be specified.

Port forwarding is a feature provided by the SSH protocol where a specified port on the server can be connected to one on the client via the SSH connection. Rather than starting a new connection between the specified ports, all communication between the client and server on this port is then routed through the existing SSH connection. This allows other applications and network protocols to be run through a single SSH connection (Barrett and Silverman, 2001). Cumulus will use this functionality to run all required network connectivity between the Cumulus system and virtual machines through SSH connections. The system will operate a simple Network Address Translation (NAT) type structure.

When a request is made to Cumulus for an open port to a virtual machine, it will link it to an open port on the web server. This will then be tunnelled through SSH to the target virtual machine. The system will maintain a pool of available ports which can be mapped dynamically as needed. This kind of NAT structure will enable users to run services on their virtual machines. For example, if a web server on port 80 is required, the service name and port will be defined in the configuration of the virtual machine. When the virtual machine is started, Cumulus will connect port 80 on the virtual machine to a random external port in the pool, port 8880 for example. The user will then be able to access the web server in the virtual machine from port 8880 on the Cumulus server. By default, each virtual machine will have the SSH port (22) connected. Additional ports can then be specified as required.

For many software tools, it is important to be able to connect to the internet. This may be because they download data from external repositories such as BiomaRt (Durinck et al., 2005) or to interface with external services. It is also important, however, that network access should not be unrestricted. Virtual machines should not be able to access internal components of the system nor other users' resources or virtual machines. In addition, the virtual machine should not have to access resources which may be transient to the extent that they impact upon the reproducibility of the analysis. To limit this, Cumulus will limit access to the internet using a HTTP and FTP proxy. All internet requests will be redirected through an SSH tunnel to the Cumulus system. There will then be a whitelist of allowed web addresses or blacklist of disallowed addresses restricting the virtual machine to a limited subset of the internet. This should block both malicious use and the incorrect use of transient resources by analysis scripts.

2.10 Summary of the System Structure & Analysis Database

The main structure of the Cumulus system will be a sever-based system with an API and web interface which will provide user and programmatic access to the systems functionality. The functionality provided by the system can be divided into four areas: management of the computational cluster of physical machines, the virtual machine sessions, the virtual disk images and the analysis tool database and associated workflows.

2.10.1 Computational Cluster

The computational cluster will consist of a set of physical machines to which the virtual container API will connect and provision virtual machines. Each of these physical machines will be pre-configured with an operating system and Oracle VM VirtualBox. From this set of physical machines there will be an active pool of available machines which can be

provisioned to. Machines can be added or removed from this pool to enable or disable new provisioning of virtual machines. As virtual machines are provisioned on these physical machines, the system will record the amount of hardware resources such as CPU cores or RAM that has been allocated. The remaining available resource will then determine the machine on which future virtual machines will be provisioned.

2.10.2 Virtual Machines

When a user makes a request for a virtual machine it will create a record for this request and provision the virtual machine from the computational cluster's active pool. Once a virtual machine is available the system will connect any virtual networking required for the request and record this in the database. The system will maintain a record of all virtual machines used, the disk images used for them and the interactions that have taken place.

2.10.3 Virtual Disk Images

The system will contain a database of virtual disk images. As previously described in **Section 2.6,** these will be used as the operating system disk for the analysis virtual machines. These can be added to, by uploading a new ISO format virtual disk image or re-configuring an existing disk. The system will store the disk file on the Stembio Drive shared storage so that they are available to all elements of the system.

Upon upload, the disk will be unpackaged and its repositories scanned as described in **Section 2.7**. The results of this scanning are stored in the Cumulus structured database. Each disk file will be linked to a database record for the virtual disk. This database record will link to the analysis tools, libraries and versions or everything installed on the disk.

The disks will be available to download individually, allowing the user to run them locally in a virtual machine. Alternatively, the user can write them to a CD-ROM or a USB boot device as described in **Section 2.6**. This ability to open the analysis environment using standard computer tools allows researchers to use the environment produced by Cumulus to reproduce the analysis without needing to install and maintain a Cumulus system.

2.10.4 Analysis Tool Database & Workflows

A database table of analysis tools will be maintained. Each analysis tool will have multiple version records and each of these version records will have one or more disk image records on which it can be found. This will enable a user to pick a specific tool and version and the system can then pick a suitable virtual disk image. As discussed in **Section 2.7**, the majority of this list of database tools will be automatically populated from the disk images. Some software packages, however, may require manual installation or are managed outside of a

package management system. In order to capture these packages, users will be able to register additional software packages and versions manually and mark them as being installed on specific machine images.

In order to manage the structural data related to the analysis tools and virtual disk images, the system will require a database. This database will store information about all of the elements of the system and link them system together. The structured nature of this data would be most suitable for a relational database. The GeneProf system already contains an SQL database, so Cumulus will adopt the same SQL database. GeneProf stores its workflow data within an experiment record. The Cumulus system will augment this experiment record with additional data about the tools and methods used.

2.10.5 Experiment Record Summary

In order for an experiment analysed through Cumulus to be reproduced by another researcher, they need to have access to the full experimental descriptor set. Studies are generally communicated in a scientific publication with a link to a set of files containing the raw and analysed results. Papers which used the GeneProf system, such as the paper by Festuccia et al. (2012), extended this to include a link to an experiment page which included intermediate data and the analysis workflow. The Cumulus system will extend this further by including the elements shown in **Table 2** in the experiment page. Cumulus will enable the researcher to download or resume the virtual machine used in the analysis and investigate the set of tools used.

Data Component	Data Type	Capture Method
Analysis Method	Script or workflow	GeneProf / workflow system
Analysis Parameters	Workflow configuration	GeneProf / workflow system
Analysis Tools	Report & Database	Cumulus Database
Analysis Environment	Virtual disk image	Cumulus
Dependency tree	Report & Database	Cumulus Database
Raw Data	Data files	Stembio Drive
Intermediate Data	Data files	Stembio Drive

Table 2 - Components of A Cumulus Experimental Record: A table showing the components of a Cumulus experimental record which is provided with the publication of a study. Each data element is shown along with the type of data contained in the element and the part of the system which captures it.

2.10.6 System Structure Overview

In order to create an analysis system which enables reproducible analysis, a new set of components will be created to augment the existing GeneProf workflow analysis system. The new components of the analysis system will fit in to the existing workflow system as a set of separate additional applications which communicate with each other through APIs. These application components and their relationships are shown in **Figure 9**. The main addition will be Cumulus; a software component whose primary function will be to store and manage analysis software tools, their dependencies and virtual disks. In addition, Cumulus will orchestrate the starting of virtual machines and the communication between these virtual machines and other elements of the system. Cumulus will maintain a database of software tools; these will be accessible to GeneProf through the Cumulus API so that workflows can be built with them. Once a workflow is scheduled to run, GeneProf will call the Cumulus API which will provision virtual machines sequentially for each stage of the workflow. As an analysis stage is completed, the result will be stored on the Stembio Drive by Cumulus and GeneProf updated.

Another addition will be the Stembio Drive; a software component which will provide file storage for analysis data and virtual disk images. Files can initially be uploaded to the system through the Stembio Drive. Each virtual machine will mount the user's private Stembio Drive area to access these uploaded files and deposit the results of analysis. The GeneProf system then has access to these results to present them as tables and

visualisations in the user interface. An additional component, Stembio Visualisation connects to the GeneProf API and the Stembio Drive to provide a framework on which to develop interactive visualisations. This is detailed in **Chapter 5.**

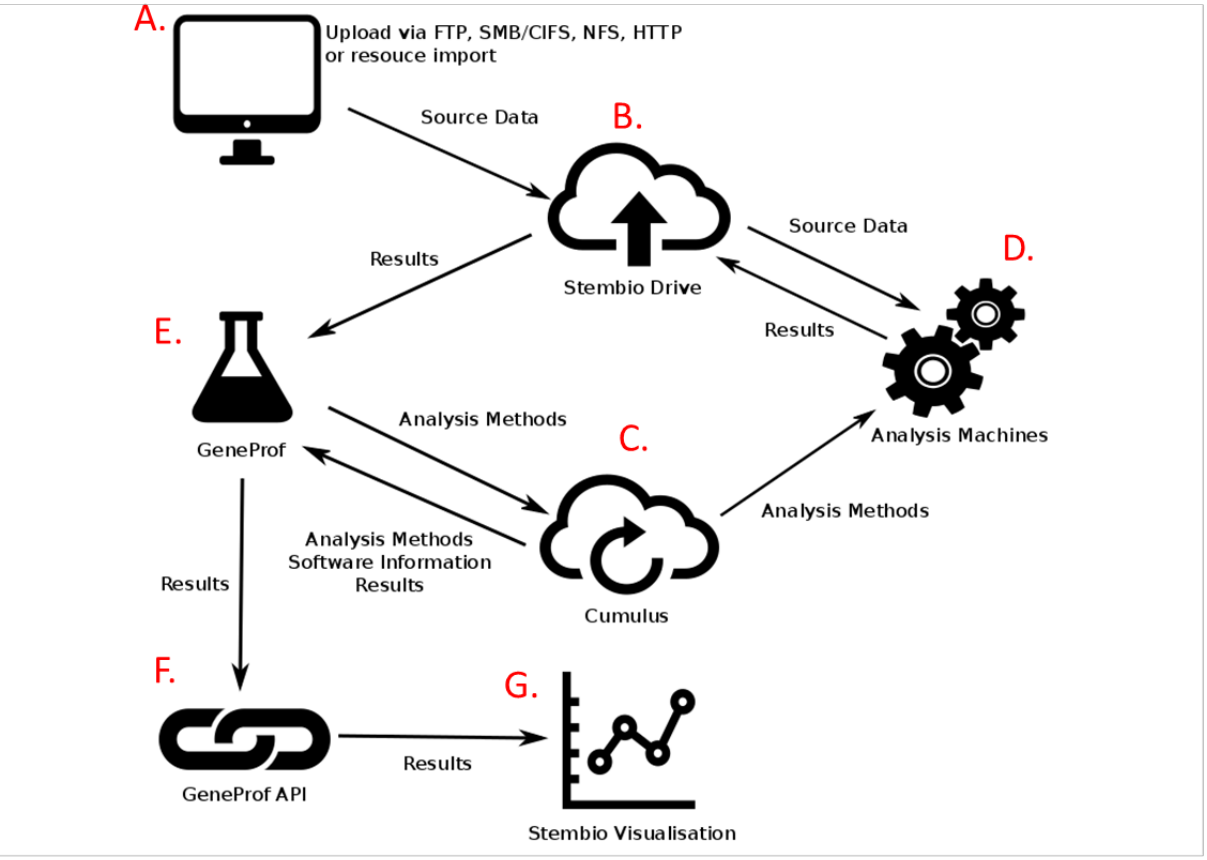

Figure 9 - System Structure Overview: A diagram showing the components of the new analysis system and the information flow between these components. (**A**) The user's computer from which they can upload their source data to the Stembio Drive. (**B**) The Stembio Drive stores source data and the results of an analysis which are accessed by other components of the system. (**C**) Cumulus stores analysis methods in the form of virtual disk images and a linked software tool library. The list of software tools and their details are shared with GeneProf which creates workflows from them and executes them through Cumulus. Cumulus orchestrates these workflows on analysis machines. (**D**) Analysis machines are sent virtual disk images which are started and controlled by Cumulus. Analysis data is loaded from the Stembio Drive and the results, post analysis, are saved back. (**E**) GeneProf retrieves lists of tools from Cumulus and constructs workflows with them. It then calls Cumulus to execute them. Results are retrieved from Cumulus and the Stembio Drive and displayed to the user. (**F**) The GeneProf API provides access to structured results from the analysed studies within GeneProf. (**G**) Stembio Visualisation accesses the GeneProf API to provide interactive visualisations of the results of an analysis.

3 Implementation of Cumulus

The following section describes the implementation of the Cumulus system. Firstly, the design and implementation of the Cumulus user interface is discussed. In following sections, the internal structures of the system such as networking and storage are detailed. Where the system makes use of existing technologies or tools, this is documented.

There are a number of high level programming languages which would be suitable for the implementation of a web server-based system such as Cumulus. The main areas of impact that the selection of a programming language has are: programmer productivity, maintainability, efficiency, portability, tool support, and interfacing with software and hardware (Spinellis, 2006). As most user interaction with the Cumulus system will take place through standard web APIs, the impact of software and hardware interfacing is minimal. The developed software will be a web-based server application. This allows the developer more freedom to specify the server requirements, operating system and environment in which it will run. Portability, therefore, is of less impact than during development of a client-based application which must be compatible with whatever the end user's environment might be. GeneProf and its supporting framework of applications have been written in Java, as this system will run within the same environment, Java will also be used to develop the Cumulus system. This will minimise the number of concurrent software platforms used, increasing maintainability and efficiency. Similarly, the GeneProf system uses MySQL (https://www.mysql.com)—a FOSS SQL based relational database—to store its structured data. The Cumulus system will use the same database technologies.

3.1 Application Overview

The Cumulus application shown in **Figure 10** is the central component of the system. This is a web-based application to access and manage a database of virtual disks and tools as well as an analysis infrastructure.

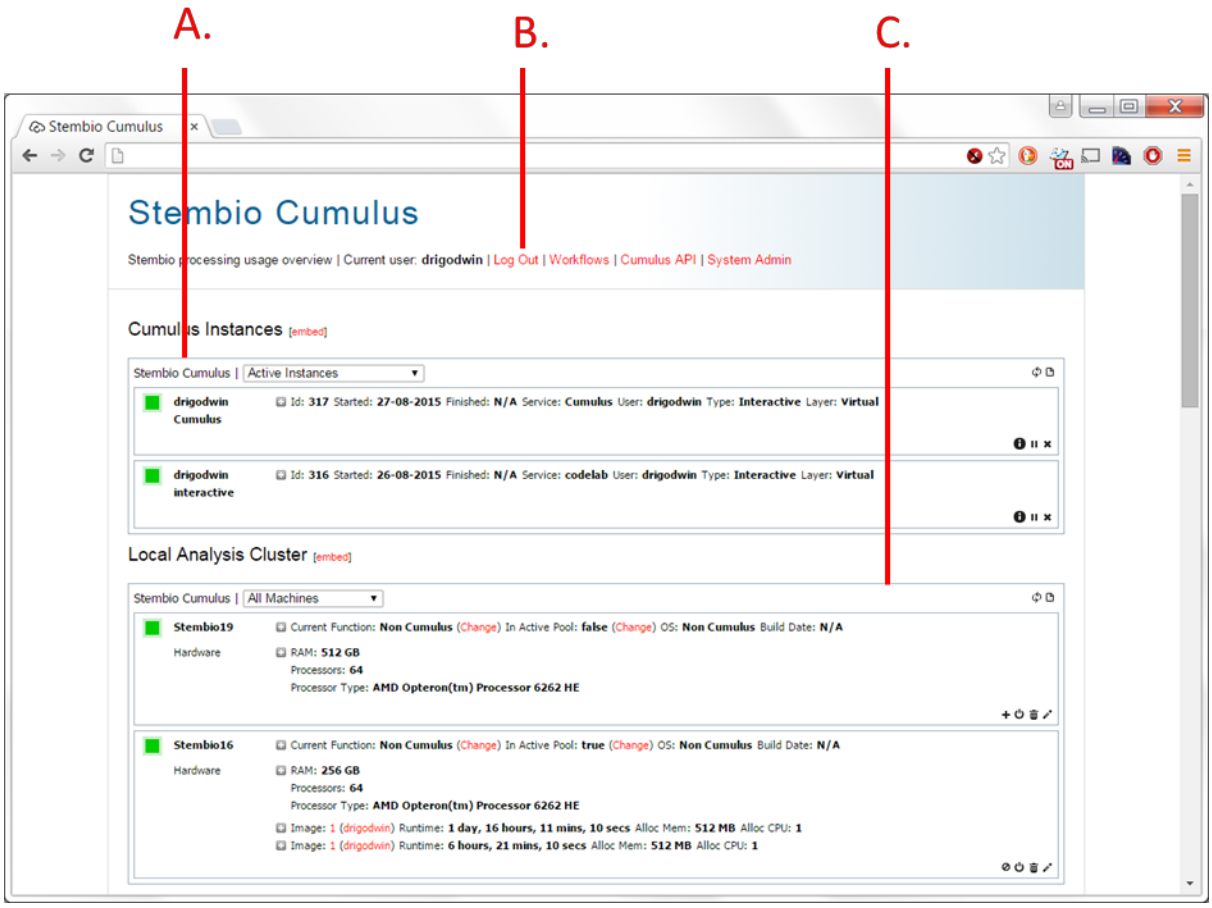

Figure 10 - Cumulus User Interface Overview: A screenshot of the main page of the Cumulus user interface showing (**A**) A user interface component detailing Cumulus virtual machine instances. (**B**) The application navigation menu. (**C**) A user interface component detailing physical machines in the local analysis cluster.

3.2 Application Platform

The system described in this thesis is made up of a number of different applications which work together through APIs. Each of the applications is based on the same underlying software platform. This platform is a set of programming tools which provide a lot of the standard functionality required by the system. This section will describe the toolset which make up this platform and how they interact.

Each application is based upon an integrated application framework. This framework enables software libraries which provide standard functionality, such as user management and database integration, to be configured and work together in a structured way. The implementation for this is provided by Spring (https://spring.io), a widely-used application framework for java. Spring enables a set of services, each providing a different piece of core functionality, to be assembled and then made available to the application logic. These services are all configured using Spring Configuration and secured using Spring Security

(https://spring.io/projects/spring-security). Spring Security allows users to authenticate themselves using a number of different mechanisms and then provides function level access and permissions to all of the services within Spring (Mularien, 2010).

A stand-alone Central Authentication Service (CAS: https://apereo.github.io/cas) is used for Single Sign-On (SSO) across all applications. It maintains a single user database across all elements of the system and enables the user to sign in on one of the applications and maintain the session across all other linked applications. For example, a user can access a GeneProf page with an embedded plug-in from Cumulus without having to separately log in to Cumulus. CAS integrates directly with Spring Security through an API.

An API service framework is a software tool which simplifies the process of writing and maintaining an API. Spring supports a number of API service frameworks; this system has been built with Apache CXF (http://cxf.apache.org). Apache CXF allows the programmer to specify a Java interface which can then be automatically translated into a variety of protocols such as Simple Object Access Protocol (SOAP) and REST. Other applications which wish to interact with the system can connect to these interfaces and run the functions specified in the Java interface. This API provides all of the functionality of the Cumulus web interface and is the mechanism by which GeneProf and other workflow systems can connect and use Cumulus programmatically.

In addition to an Apache CXF API, the system includes a Dynamic Web Remoting (DWR) API. DWR is an Asynchronous JavaScript and XML (AJAX) and reverse AJAX (Comet: Crane and McCarthy, 2008; Fette and Melnikov, 2011) interface and provides a standard way in which web interfaces can communicate with the Java server. Similarly to the CXF API, the programmer can create a Java interface which is then automatically translated into a JavaScript API available to the user's web browser. Calls made to this JavaScript API are communicated to the server by the DWR framework and call the equivalent Java function. In addition to this, DWR provides a Comet interface which allows the web server to call functions in the web browser of users who are currently browsing a page. This enables functionality such as alerting of events and call-backs to update a UI. This JavaScript API is also available to external websites and applications to interact directly with Cumulus.

Data persistence is the storage of the system state as structured data on disk. It allows the system to resume its state after it has been restarted. It is implemented using a relational MySQL database. The Java code interacts with this MySQL database using Hibernate (http://hibernate.org), an Object Relational Mapping (ORM) tool (Xia et al., 2009). ORM tools map Java classes and their member variables on to SQL database tables and columns. When a Java object is persisted, or retrieved, Hibernate marshals the conversion between

the Java object and the database table. Additionally, Java objects can be used to create queries which are converted to SQL by Hibernate. This allows data to be moved between Java and an SQL database without using SQL within the Java code. The deployment of the application can then use different database implementations depending on the situation. Hibernate also integrates with Spring allowing it to abstract and reduce the complexity of managing database connections (Fisher and Murphy, 2010). The Cumulus database structure, detailing the SQL to Hibernate mapping objects are shown in **Table 3**.

Object Name	Description	Fields
CumulusLog	The log of a virtual machine or PXE machine run by Cumulus.	id (PK), user (FK), nodeStarted, nodeStopped, nodeType, experimentId, created
Distribution	A type of disk image which can be used for PXE deployment.	id (PK), user (FK), softwareVersionInstalls (FK), distributionType (FK), executionRequest (FK), platform, init, kernel, name, buildVersion, buildDate, imageFile, readStatus, repoType, readLog, colour, imageUser, imagePassword, sshPort, pathVirtualBoxBin, pathTempFolder, vBoxHostOnlyNetworkOverride, vBoxStorageControllerOverride, internalVBoxControlPortPoolStart, defaultSSHEncryption, defaultSSHMACEncryption, cloudEnabled, mountPath, unmountPath, userAddPath, testCommand, driveMountPath, created
DistributionType	A table for linking distribution objects of the same type.	id, user (FK), name, distributionFunction, created
DriveReference	A reference to a file on the Stembio Drive	id (PK), user (FK), share, path, targetFile
ExecutionRequest	An object representing a request to execute a piece of software.	id (PK), user (FK), executionRequestLog (FK), machine (FK), workflowExecutionRequest (FK), workflowItem (FK), virtualContainer (FK), distribution (FK), softwareVersion (FK), name, uuid, description, exitStatus, service, sourceExecutionRequests, clearIntermediateFiles, requestedOn, serviceStarted, serviceTerminated, clearedOn,

		type, layer, executionStatus, requestedRam, requestedCores, requestedCPUCap, enableInternet, enableFTP, enableDrive, enableLogin, shareToMount, geneProfId, created
ExecutionRequestLog	The logs produced by executing a piece of software.	id (PK), user (FK), executionRequest (FK), outputConsole, errorConsole, created
ExecutionRequest-Parameter	An object representing a parameter passed to a piece of software in an execution request.	id (PK), user (FK), executionRequest (FK), softwareParameter (FK), value, created
HashTag	A tag allowing software and other resources to be grouped together.	id (PK), user (FK), value, created
HostedService	A service hosted by a specific virtual machine with the port numbers it requires.	id (PK), user (FK), serviceName, serviceDescription, portRange
Machine	An object representing a physical machine which Cumulus can use to run virtual machines or launch a PXE image.	id (PK), user (FK), distribution (FK), macAddress, ipAddress, name, status, overrideUser, overridePassword, overrideVBoxManage, overrideTempFolder, overrideNetcat, desctiption, installDisk, networkCard, restartNextTime, rewriteNextTime, lastUpdated, inActivePool, processors, availableRAM, availableStorage, processorPlatform, ioProfile, hardwareAccelerationEnabled, graphicsAccelerationEnabled, graphicsCardType, processorType, physicalLocation, created
ProcessingPool	An object representing a group of machines.	id (PK), user (FK), machine (FK), poolname, created
Service	A service which can access the Cumulus API.	id (PK), serviceName, serviceDescription, notificationURL, notificationEnabled, serviceIPMask, apiKey
SoftwareOutput	This is the output from a software package once it is run, such as analysis result files.	id (PK), user (FK), softwareParameterMap (FK), name, description, dataTypeRich, expectedFileName, expectedFileNamePattern, created
SoftwarePackage	An object	id (PK), user (FK), hashtags (FK),

	representing a software package. This may have multiple versions and exist in multiple disk images.	interpreter, packageName, description, systemLibrary, packageType, alterativeNames, webAddress, type, bioResourceIndexId, initiallyAdded,
SoftwarePackageAltName	This is an object for an alternative name for a package, created when packages, named differently under different Linux distributions are merged.	id (PK), user (FK), softwarePackage (FK), packageName, created
SoftwareParameter	A parameter for a piece of software.	id (PK), user (FK), softwareParameterMap (FK), softwareParameterType (FK), name, description, dataTypeRich, dataType, prefix, postfix, defaultValue, switchValue, maxFiles, optionList, upperRange, lowerRange, required, hideFromUser, created
SoftwareParameterMap	This is a mapping of the inputs and outputs of a piece of software.	id (PK), user (FK), created
SoftwareRequirements	A hardware requirement of a piece of software.	id (PK), user (FK), ramRequirement, cpuCoreRequirement, hddRequirement, created
SoftwareVersion	A version of a software package.	id (PK), user (FK), softwareRequirements (FK), softwareParameterMap (FK), softwarePackage (FK), experimentsProcessed, versionNumber, releaseNumber, codeName, releaseDate, blockRecommended, initiallyAdded, created
SoftwareVersionInstall	A software version installed on a virtual machine image or distribution	id (PK), user (FK), softwareVersion (FK), distribution (FK), virtualContainer (FK), installPath, interpreterPath, interpreter, created
User	An object for a Cumulus user account.	id (PK), user (FK), username, password, passwordSalt, alternatePassword, alternatePasswordSalt, apiKeyActive, apiKeyHash, created
VirtualContainer	An object representing a virtual disk image.	id (PK), user (FK), containerType (FK), platform, containerName, versionNumber, buildDate, availability, unavailable, imageName, readStatus,

		repoType, readLog, imagePath, diskPath, tempPath, mountPath, unmountPath, curlFTPFSPath, userAddPath, driveMountPath, testCommand, username, password, sshPort, sshTimout, sshInterval, sshAttempts, sshKeepAlive, internalMapRetry, internalSharePostfix, ramAllocation, cpuCoreAllocation, cpuUsageAllocation, ipDetectionAttempts, ipDetectionTimeout, defaultRAMAllocation, defaultCPUCoreAllocation, cpuUsageAllocation, defaultSSHEncryption, defaultSSHMACEncryption, created
VirtualContainerInstance	An object representing a virtual machine.	id (PK), user (FK), executionRequest (FK), machineImage (FK), machine (FK), containerId, serialisedVBox, outputConsole, errorConsole, status, started, stopped, lastResumed, cumulativeTime, internetEnabled, ramAllocationOverride, allocatedCPUCoresOverride, allocatedCPUCapOverride, macAddress, ipAddress, bootState, uuid, created
VirtualContainerType	An object representing a group of virtual containers.	id (PK), user (FK), name, virtualContainerFunction, created
Workflow	A workflow object is pointed at by multiple workflow items to constitute an analysis workflow.	id (PK), user (FK), name, description, published, publishedOn, coreTechnologyWorkflow, technology, created
WorkflowExecutionRequest	An object linking a set of execution requests to a workflow to represent the running of that workflow.	id (PK), user (FK), workflow (FK), created
WorkflowItem	A representation of one item in a workflow. Linked as a list to next and previous workflow items via dependents and dependencies.	id (PK), user (FK), workflow (FK), softwareVersion (FK), dependents (FK), dependencies (FK), overriddenName, overriddenDescription, created
WorkflowItemParameter	A set parameter of a piece of software in a	id (PK), user (FK), softwareParameter (FK),

	workflow item.	workflowItem (FK), overriddenName, overriddenDescription, userEditable, required, value, created

Table 3 – The Cumulus Database Structure: A table showing the objects within the Cumulus database, a description of each object and the fields the object contains. Fields which are primary indexing keys to the table are marked with (PK). Foreign keys which link the object to the primary key of another table are marked with (FK). Foreign keys are given the name of the table to which they are linked unless indicated otherwise in the description column. All tables are linked to user which indicates the user that created the object.

3.3 User Interface

The User Interfaces (UI) for the components of the system have been implemented as a set of JavaScript jQuery (https://jquery.com) plugins. jQuery is a JavaScript framework which provides an advanced set of JavaScript tools and enables cross-browser functionality (Duckett et al., 2014). Each plug-in is an interface built using Hypertext Mark-up Language (HTML) and Cascading Style Sheets (CSS) and controlled using jQuery. The jQuery plug-in uses the DWR interface to interact with the server-side functionality and retrieve data. This structure is shown in **Figure 11**. These jQuery plug-ins can then be embedded into web pages or external sites, enabling components of the system to be used and included in other systems. Using this structure, parts of the Cumulus system can be included in the interface of the GeneProf workflow system without changing the GeneProf code. This gives two mechanisms by which external applications can integrate with the system; via the CXF-based API or by inclusion of the jQuery plug-in its own UI.

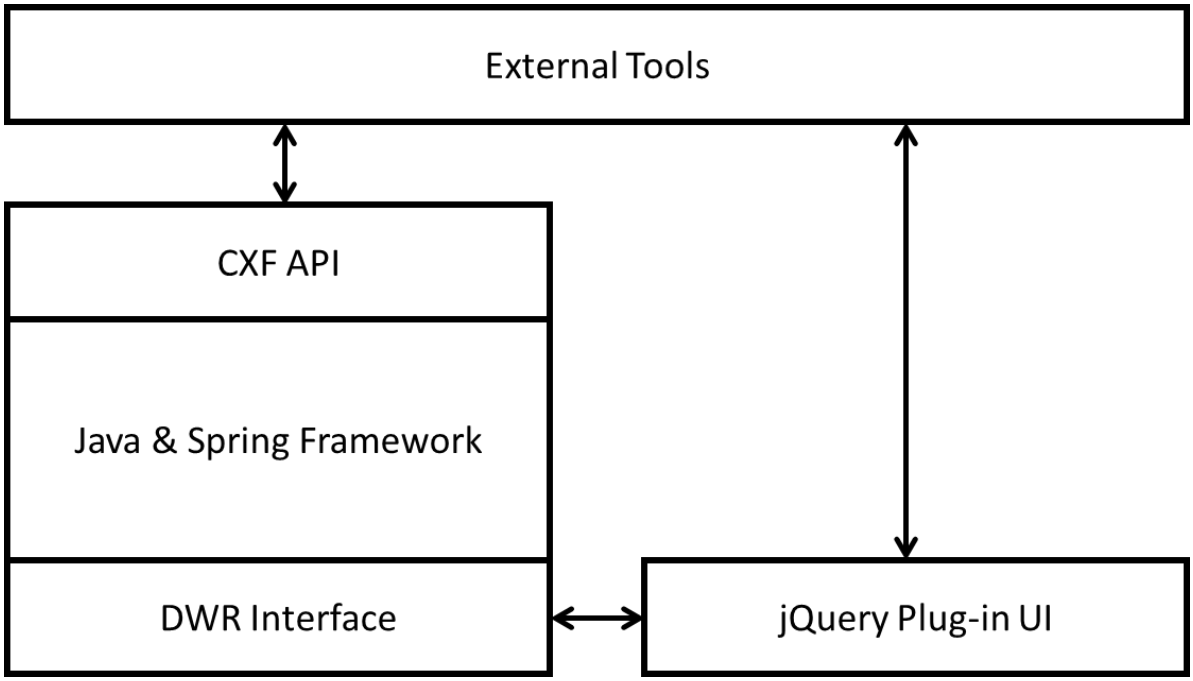

Figure 11 – The Cumulus Application Platform Structure: A block diagram showing the structure of the Cumulus application platform, a programming platform which is the basis of all the applications in the Cumulus system. The blocks represent application components and the arrows represent interactions via a HTTP based API. Adjoining blocks represent interaction via a Java programming interface.

The user interface design is minimal in order to allow the host applications to skin or theme the components as required. The jQuery plug-ins or modules have a standard layout shown in **Figure 12**, which has been extended in some modules where required. Each module is a paged tabular format with a filter and search option. There are a selection of controls at the top right for creating a new record or refreshing the records from the server. The records themselves contain the record details, any sub records and a set of buttons for associated functions such as deleting the record or restarting a server. Creating a new record or editing an existing record opens a new dialog box with input and form options to allow the user to enter records. Changes made through these dialogs can then be saved to the server, auto-refreshing the table.

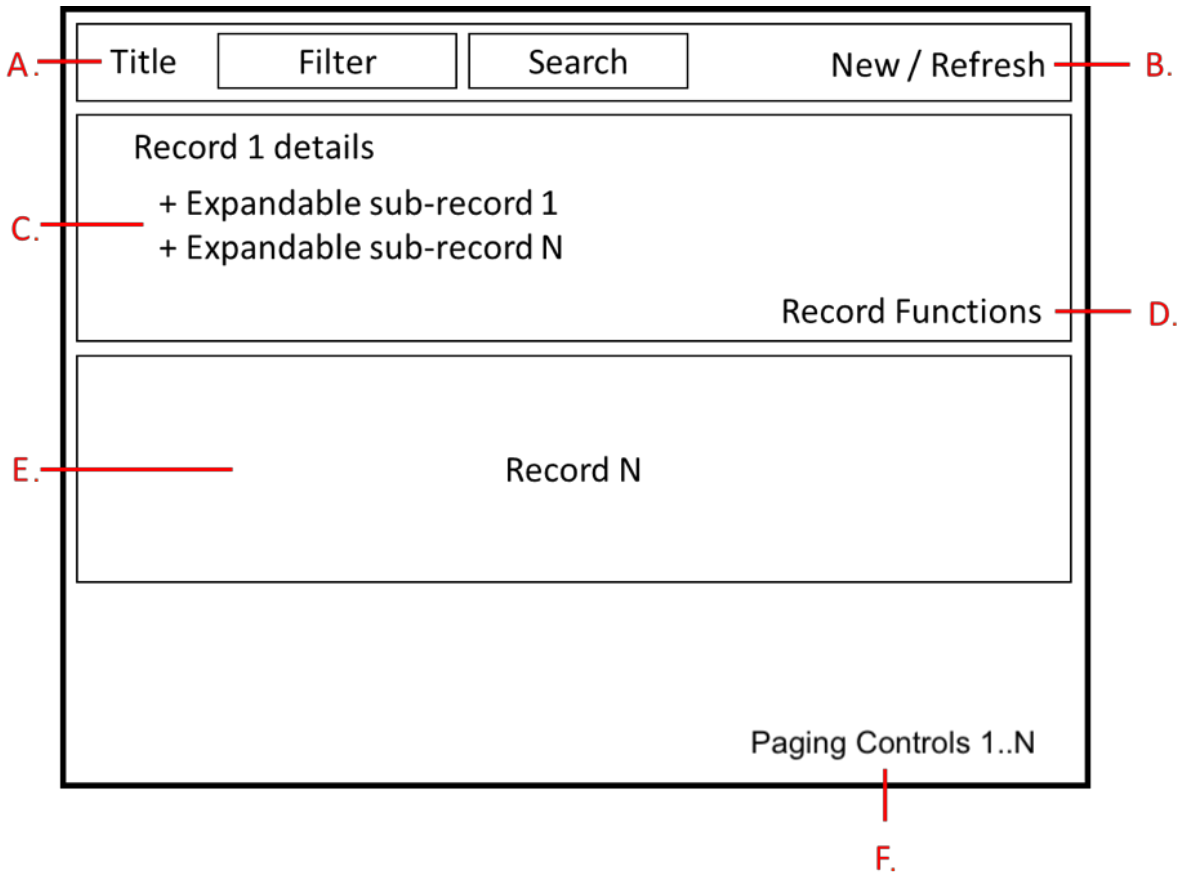

Figure 12 - Design of The User Interface of a Cumulus UI Module: A wireframe diagram showing the generic design of a Cumulus UI module. This generic design is extended with custom controls for the different modules. (**A**) The title bar and controls for filtering and searching the records in the list. (**B**) Controls for creating a new record and refreshing the records from the server. (**C**) A record entry which displays the fields of the record. Any sub records are shown as a summary with an expand button to get the full information. (**D**) Buttons for functions specific to the record, such as editing. (**E**) Additional records on the page will be displayed in the same manner as the first, building up as a scrollable list in the main display panel. (**F**) Paging controls to move between pages of records and controls to set how many to show per-page.

3.4 Cumulus User Interface Plug-ins

The main Cumulus page shown in **Figure 13** contains a functional list of all the plug-in modules available. These plug-ins can be used *in situ* on the page or embedded into other applications. The following section details these plug-ins.

3.4.1 Cumulus Instances

The Cumulus Instances plug-in displays and manages details of the currently running virtual machines. For each virtual machine, the plug-in provides information such as the hardware resources it has access to and the machine reports. There are two categories of virtual machines, interactive and batch analysis. Batch analysis virtual machines are running a specific task and will be stopped and produce an output once this task has been completed. Interactive virtual machines will run until stopped by the user or the system. The user has the

ability to stop, start or pause any virtual machine they have access to as well as get access details to log on.

3.4.2 Local Analysis Cluster

The Local Analysis Cluster plug-in provides details on the physical machines Cumulus has access to. Each machine has its attributes stored (such as number of processors and type and the amount of RAM allocated) and the user can restart or apply a new physical disk image via a PXE boot. Machines can be added or removed from the active pool. When a physical machine is in the active pool, virtual machines will be provisioned on it.

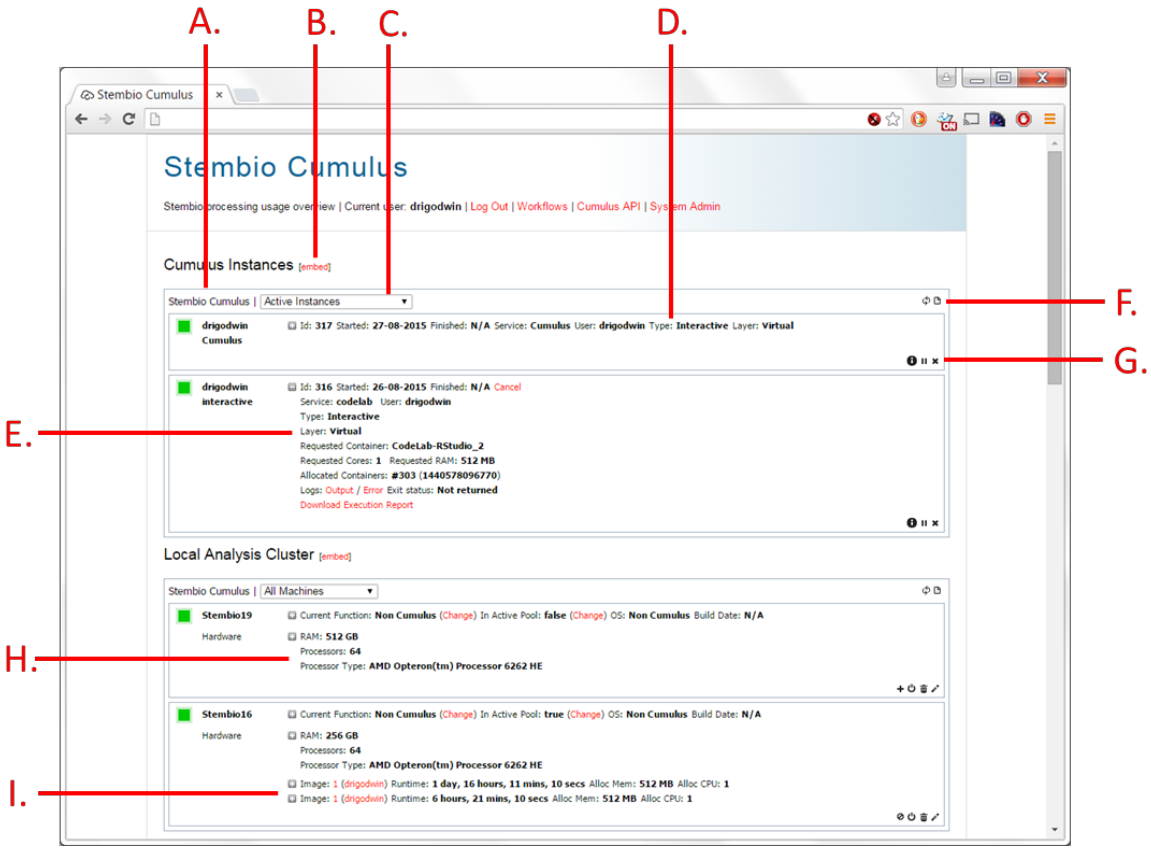

Figure 13 - Cumulus User Interface Plug-ins: A screenshot of the main Cumulus user interface showing (**A**) A user interface plug-in displaying Cumulus virtual machine instances. (**B**) The controls to embed the plug-in into an external application. Clicking this will open a pop up with instructions and embed code. (**C**) A filter combo box to filter the list of virtual machines shown in the plug-in. It is shown filtered to active instances which will limit the currently displayed virtual machines to those which are active. (**D**) A record overview, the compacted form of a record shown by default to reduce the space used. (**E**) A full record, an expanded record showing all of the information and controls related to the item. (**F**) All-record controls: controls which apply to the full set of records such as new and refresh records. (**G**) Record-specific controls: controls which apply only to one record such as delete or pause. (**H**) A physical machine record showing the hardware information of the associated machine. (**I**) Sub records showing virtual machines which are running on the physical machine record. These are displaying the amount of hardware resources each is using and the amount of time it has been running.

3.4.3 Analysis Tool Library

The Analysis Tool Library plug-in provides the records for all analysis tools available in Cumulus. These are automatically recorded when a virtual disk image is imported or can be entered manually through the UI. The plug-in shows the available versions of the software and the virtual disks on which each version is stored. Each piece of software can have parameters specified through a package parameters dialog. When the user runs a specific module, Cumulus will pass the values entered as parameters. Alternatively, in workflows, Cumulus can connect together modules by passing the output of one module as an input to another.

Parameters and tool outputs can be specified for each version of a piece of software. When a new version of the software is added, it will automatically inherit the parameters and outputs from the previous version. These inherited values can be edited if the new version is different. The user interface for this is shown in **Figure 14**.

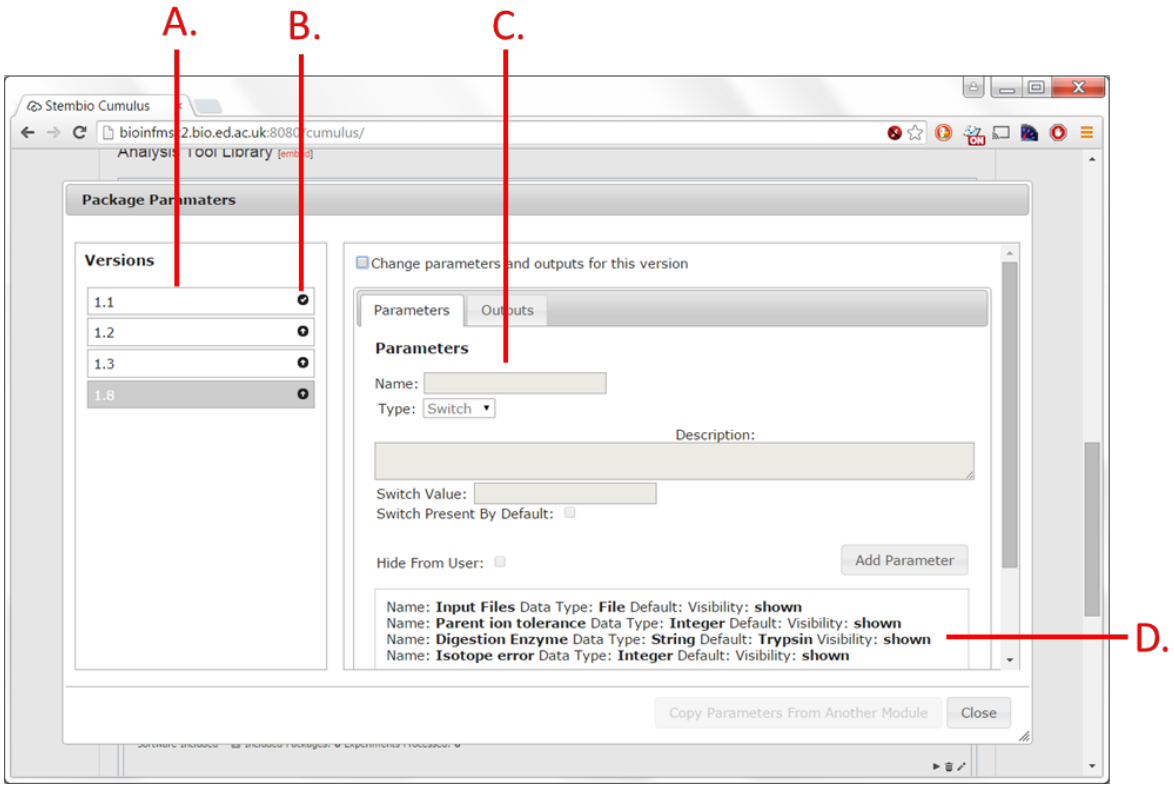

Figure 14 – Cumulus Software Tool Parameters: A screenshot of the Cumulus user interface showing the dialog box through which the parameters for a specific piece of software can be edited. (**A**) The selectable versions of the software; each can be selected and edited. (**B**) The inheritance icons, tools with an upwards arrow inherit the parameters of the previous version (default). Tools with a tick change these inherited parameters to add or remove parameters from the previous. (**C**) The parameters form, showing controls to create a parameter for the currently selected version. (**D**) A list of the specified software parameters for the currently selected version.

Users can select a software tool and version and run it directly from the plug-in. They are then presented with a dialog—shown in **Figure 15**—with which to set parameters which will subsequently be used to run the selected software tool. File parameters can be selected with a file browser which shows the files on the user's Stembio Drive. Any files a user has uploaded to their Stembio Drive will be displayed and can be selected. Once start is clicked, a new virtual machine is started which contains the version of software requested. The requested software is run, and the result returned as a Cumulus Instance output.

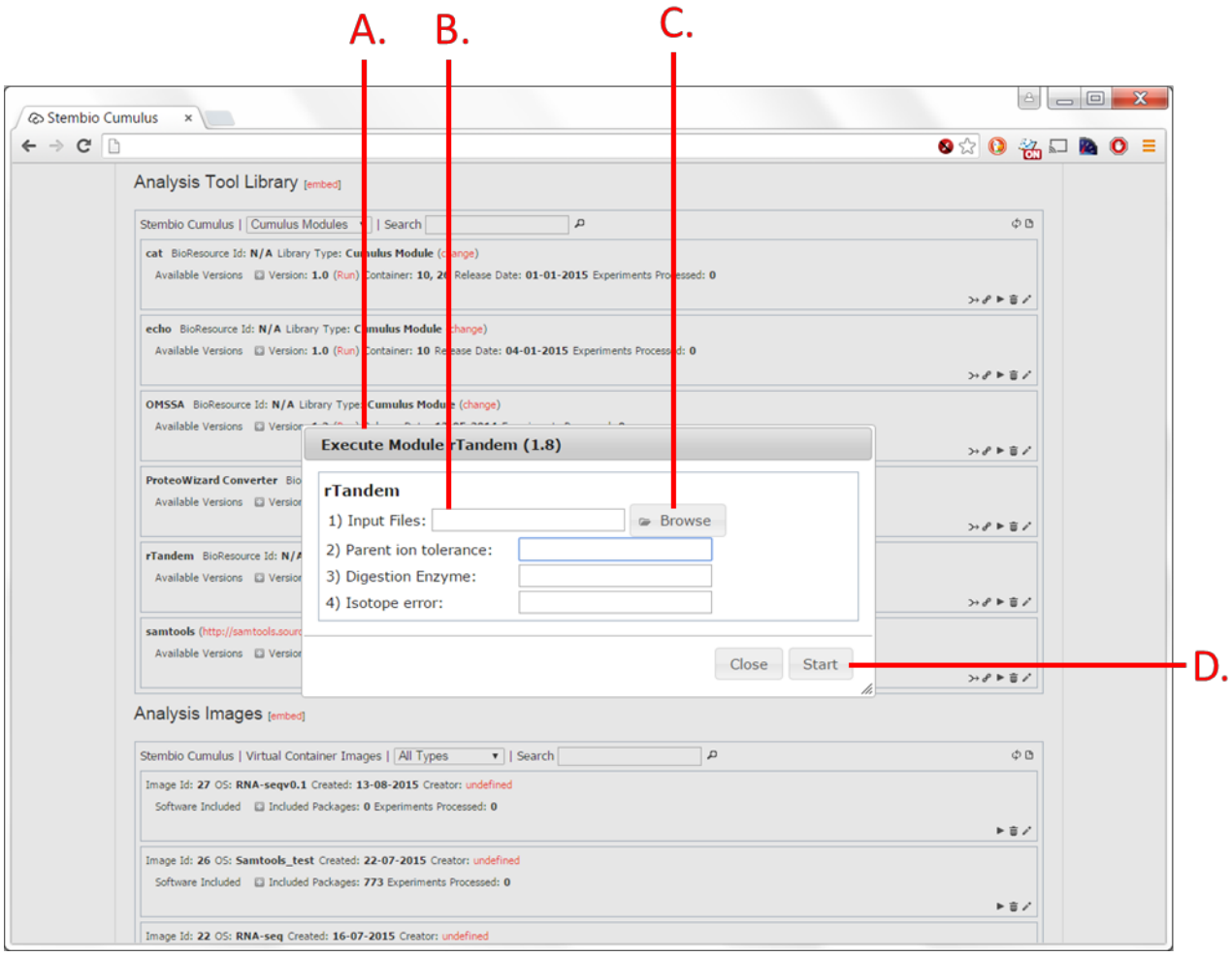

Figure 15 - Cumulus Execute Software Tool: A screenshot of the Cumulus user interface showing (**A**) The Cumulus execute software tool dialog box with rTandem version 1.8 selected. (**B**) Text and file input boxes which allow the user to specify the value of parameters for execution. (**C**) A file selection input box with a browse button. This will open a further dialog box displaying the contents of the users Stembio Drive. A file from this can then be selected. (**D**) A Start button to launch the specified piece of software on a new virtual machine.

3.4.4 Analysis Images

The Analysis Images plug-in shown in **Figure 16**, shows the database records for the virtual disk images. Each record shows the packages included on the virtual disk as well as information on what the disk can be used for. A 'play' option is available which starts the disk as a new interactive virtual machine.

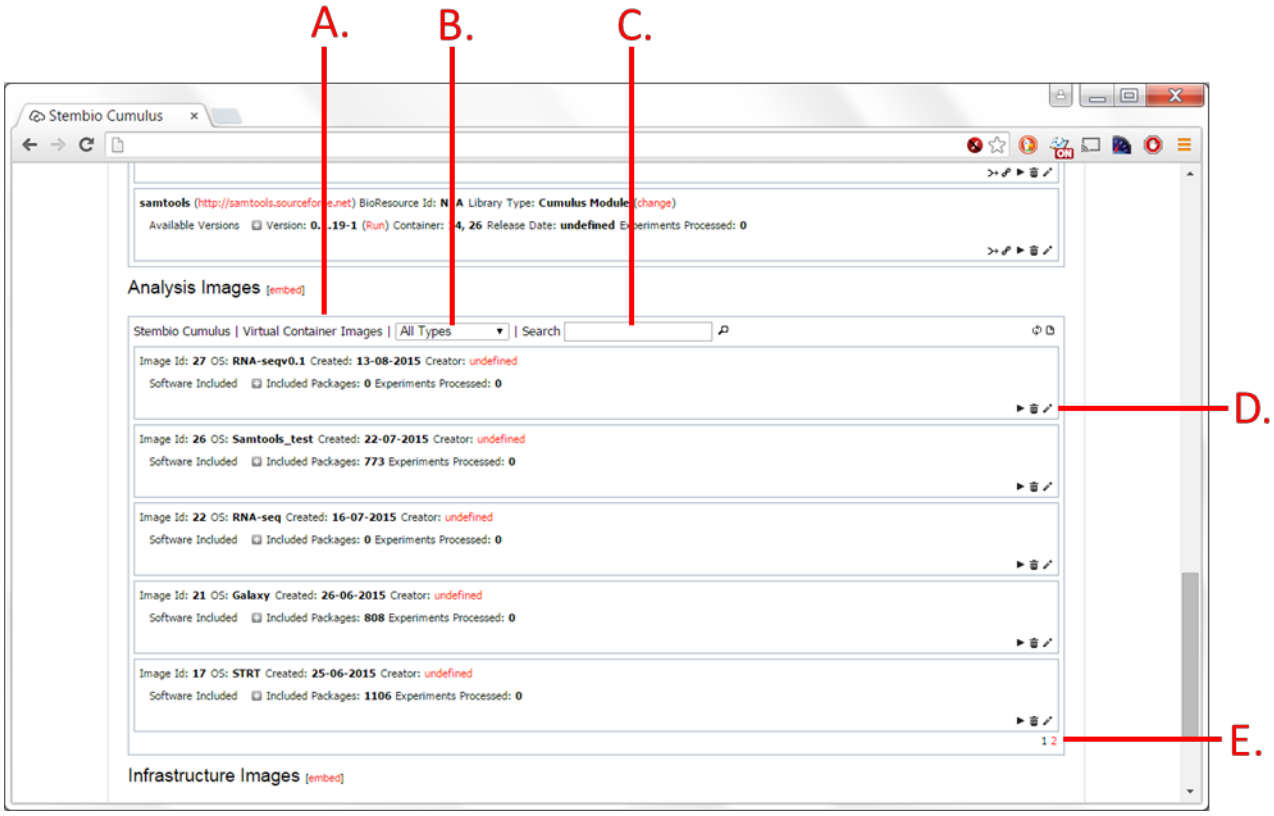

Figure 16 - Cumulus Analysis Images: A screenshot of the Cumulus system showing (**A**) The Cumulus plug-in for control of virtual disk analysis images. (**B**) A filter combo box to limit the plug-in to only showing hardware or virtualisation specific images. (**C**) A search box to search for disk images by one of its fields. (**D**) Record-specific controls to edit, delete or start a specific image as a new virtual machine. (**E**) Page controls to move between different pages of records.

New virtual disks can be selected from the user's Stembio Drive. These are copied into the Cumulus database and indexed for software packages. These packages are added to the Analysis Tool Library. Additional analysis tools can be specified through the edit image dialog box shown in **Figure 17**.

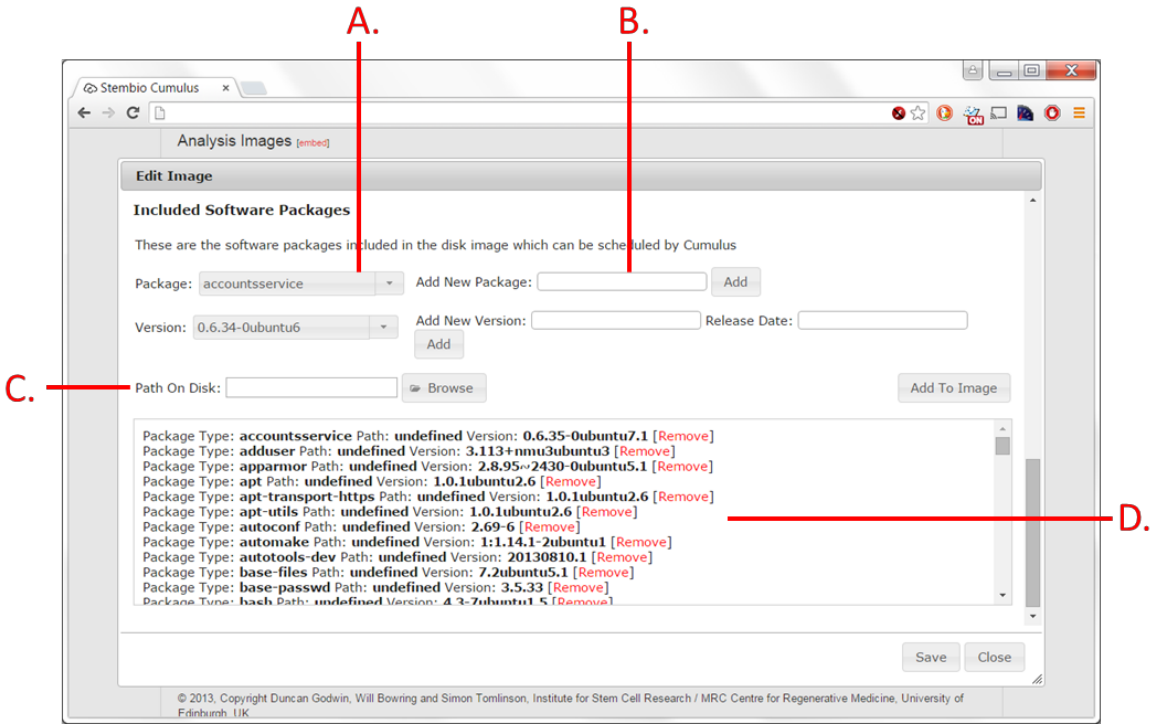

Figure 17 - Cumulus Analysis Image Software: A screenshot showing the dialog box to edit a virtual disk analysis image. (**A**) A combo box to register that an already-specified software package is installed on the virtual disk. (**B**) Input boxes to specify a new software package is installed on the virtual disk. (**C**) A file selection box to specify the location of the executable on the virtual disk. If the browse button is selected it will open a file selection dialog listing the contents of the virtual disk allowing the user to pick from the files. (**D**) The list of software packages already registered as being installed on the analysis image. This is pre-populated when the analysis image is first added.

3.4.5 Workflows

Cumulus contains a Workflows plug-in shown in **Figure 18**, which enables the user to create a workflow from the analysis tools in the Analysis Tool Library. Each analysis tool is represented by an Analysis Block. A workflow is represented by a number of these Analysis Blocks chained together. The parameters for these analysis tools can either be specified or linked to the output of a previous analysis block. These workflows can then be run which will start a chain of virtual machines producing an output.

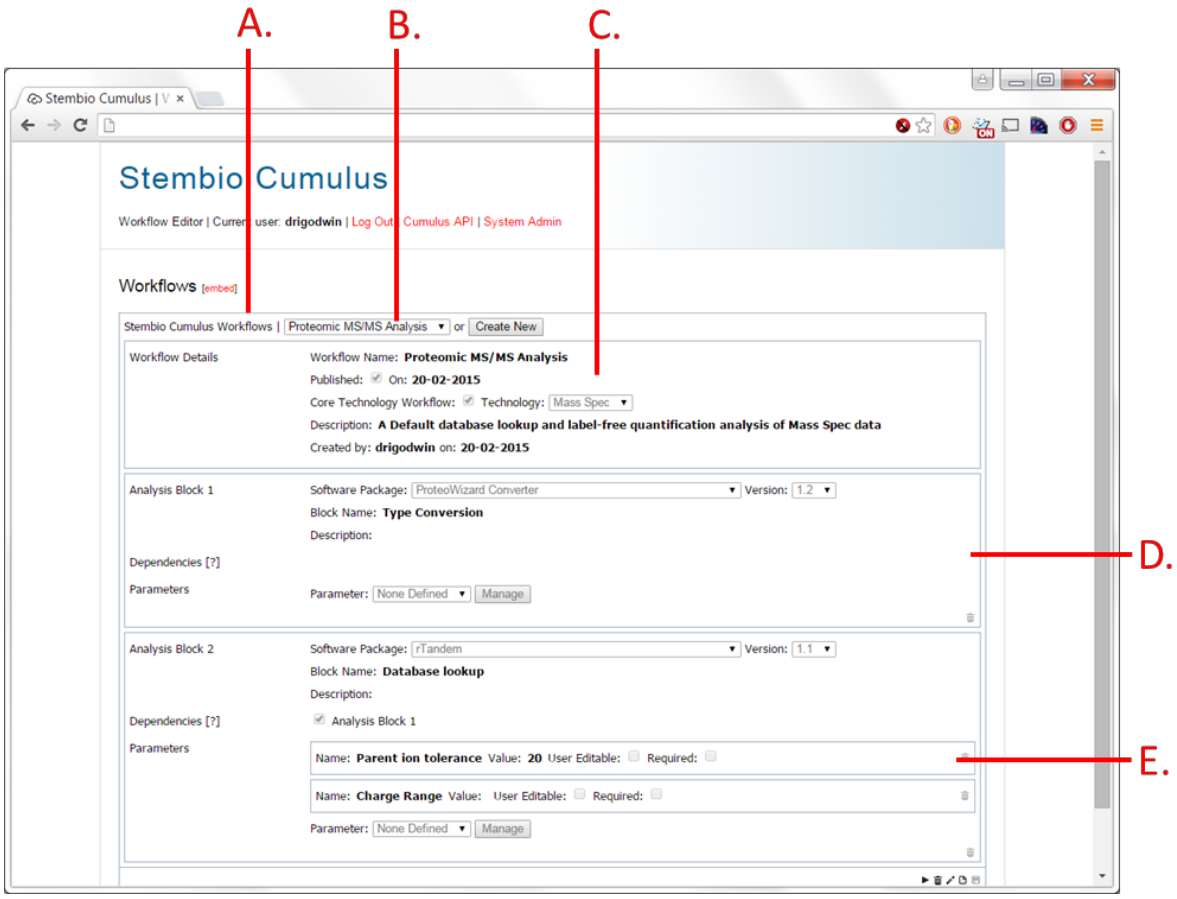

Figure 18 - Cumulus Workflows: A screenshot of the Cumulus system showing (**A**) the workflow plug-in, for creation and management of workflows within Cumulus. (**B**) A control to load an existing workflow into the editor or create a new one. (**C**) The workflow details box: these are controls to view and edit the workflow details such as name, descriptions and core technologies used within the workflow. These are used for categorisation and searching of workflows. (**D**) A workflow analysis block: this specifies a single stage of a workflow. A software package and version can be specified which will be run during this stage of the analysis. (**E**) Software parameters for the software tool. These can be specified as static values or linked to the output of a previous analysis block.

3.5 Storage

The Stembio Drive described in **Section 2.6.2** is a storage application which enables data files to be uploaded to the system and moved between virtual machines within Cumulus. It is implemented using the same Java Spring Framework previously described for Cumulus. The file sharing protocols are implemented using the Alfresco JLAN library (https://community.alfresco.com). A custom user management layer has been implemented upon this to implement quotas and connect it to the Spring framework's user management.

The user interface for the Stembio Drive shown in **Figure 19**, consists of an admin interface providing the various shares available, the user's quota and access information for connection via the different protocols. In addition, it allows the user to specify access for different external users and applications and download data from external repositories. The interface is again implemented as a jQuery plug-in which can be included in external applications.

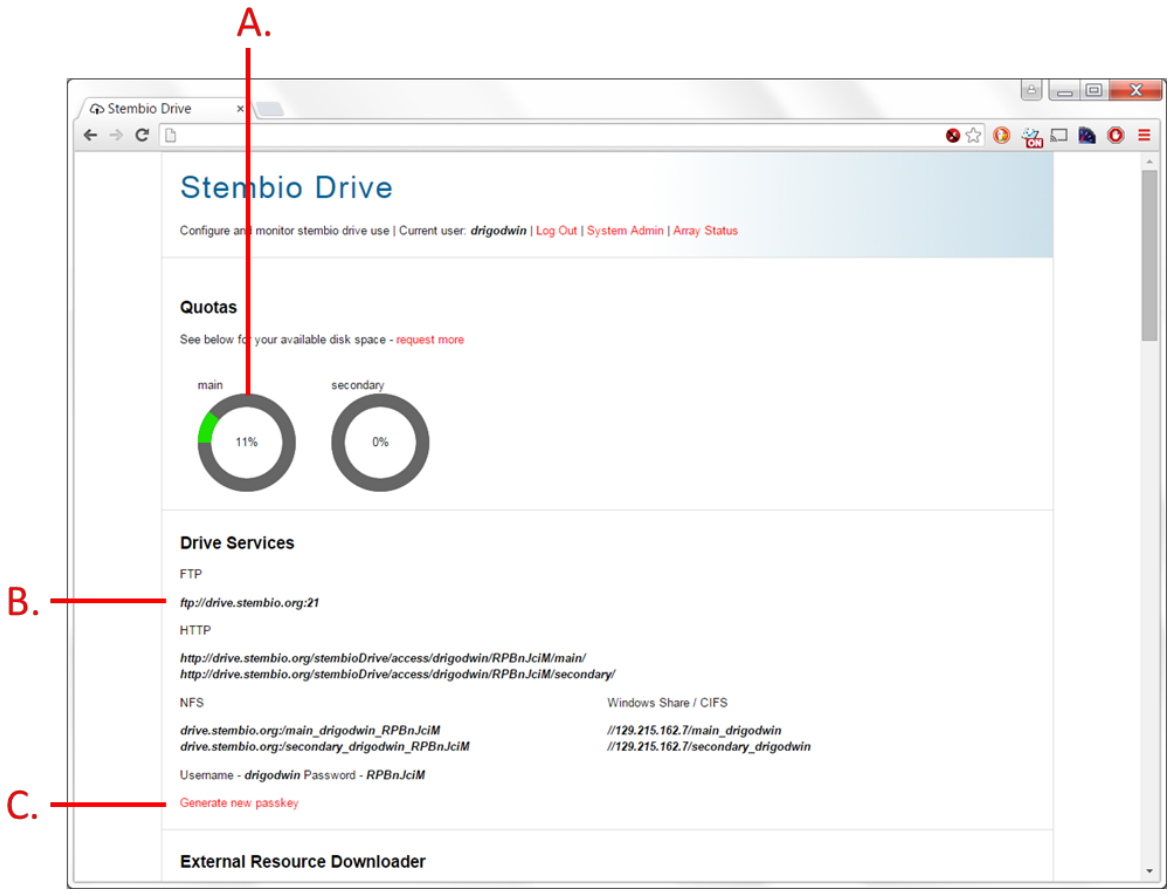

Figure 19 - Stembio Drive User Interface: A screenshot showing the Stembio Drive management user interface. (**A**) The percentage of disk space quota that the user is currently using, displayed as a pie chart. This is shown for each share within the drive. (**B**) Details for accessing the drive services such as URLs, username and passwords. (**C**) A control to reset the password for the share.

For administrative users, the Stembio Drive provides a mechanism for attaching new physical storage servers and checking the health of the disks on these servers. It also provides API details for enabling external applications to access the drive programmatically. These interfaces are shown in **Figure 20**. In order to monitor the disk health, a custom driver has been implemented in C. This component has been implemented in C because it is a requirement of the developer tools. In order to work on different operating systems the driver will require re-compilation on the target platform. The Java server connects to this driver to retrieve the physical device information.

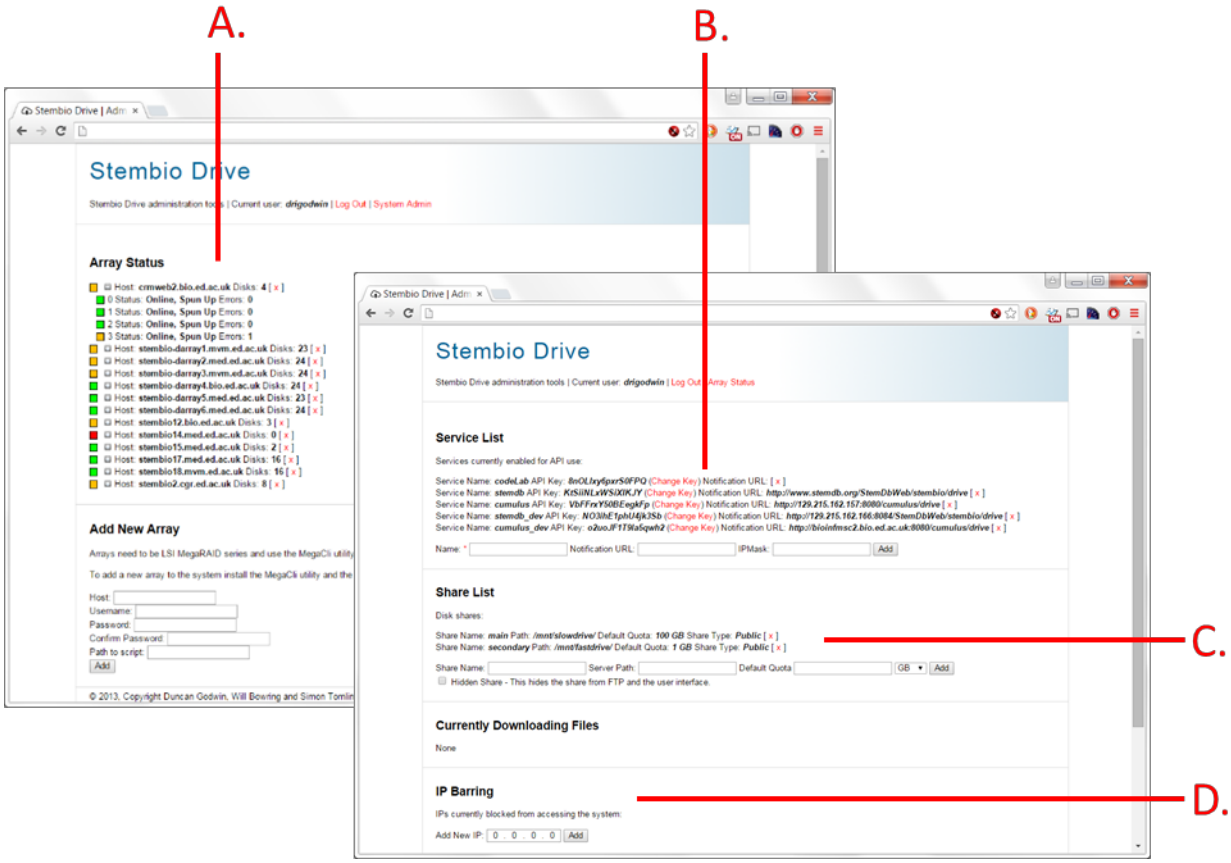

Figure 20 - Stembio Drive Further Functionality Overview: Two screenshots of the Stembio Drive showing (**A**) the physical disk array management interface. This is a list of physical disk arrays and disk drives within these arrays. Green blocks represent a healthy working drive or array, orange represents a reported problem with a drive or array and red represents non-functioning of the drive or array. (**B**) API controls: to allow and disallow select applications to access the Stembio Drive API. (**C**) Controls to create or edit a share. This maps a physical mount point to a virtual share within the Stembio Drive. (**D**) Access controls and IP barring for specific internet addresses. These enable different access rights to select computers, such as those within the Cumulus network.

3.6 Networking

Virtual machines launched through the Cumulus system are configured to use the VirtualBox Host-Only network. A Host-Only network is a virtualised network on which the only two devices are the physical processing machine—or host machine—and the virtual machine.

This Host-Only network is the only network available to the virtual machine and the virtual machine can only connect to the host machine through it. Cumulus connects using SSH to the host machine on which the virtual machines are running. Using this connection, Cumulus then connects via the Host-Only connection to the virtual machine itself, again using SSH. This connection between Cumulus and the virtual machine provides a channel for Cumulus to control the virtual machine and connect services such as the Stembio Drive.

3.6.1 SSH Optimisation

The system makes heavy use of SSH tunnelling, detailed in **Section 2.9**, for several areas of networking. Traffic in and out of the virtual machine is all tunnelled through the controlling SSH connection. This includes user connections, internet communication and disk access for analysis data on the machine. Each connection from the outside can be tunnelled multiple times, firstly through the connection to the physical machine and then through the connection to the virtual machine. This multiple tunnelling caused application errors during testing. The errors indicated applications were timing out during file reading and writing. To preliminarily investigate this, files were read from the virtual machine, host machine and through the Cumulus connection and the speeds compared. Reads from the host machine through a single SSH connection were slower than from the local virtual machine storage. Reads from Cumulus through multiple SSH connections were slower again and caused failures due to the low speed rate.

By default, SSH uses encryption to transfer data between the two machines (Ylonen and Lonvick, 2012b). With multiple layers of tunnelling, there are multiple layers of encryption. Each piece of data is encrypted again for each layer with a different key leading to the same piece of data being encrypted multiple times. There are no benefits to this additional encryption and, since Cumulus operates on a private network, the benefit of encrypting data at all is limited.

In order to avoid this costly increase in encryption, the system has used High Performance Networking Secure Shell (HPN SSH: https://www.psc.edu/hpn-ssh, Rapier and Bennett, 2008). HPN SSH is a patch on the standard SSH library which is required on both the server and client. The HPN SSH library both applies a number of networking optimisations to the SSH protocol but also enables a number of additional encryption protocols. These additional protocols are different and often faster implementations of cryptography including an option to turn it off altogether. Encryption cannot be turned off on the standard implementation of SSH. As the communication between the server and the virtual machine is running inside a private network already, this encryption is of limited use. In using HPN SSH and disabling

encryption, the speed of an SSH tunnelled connection improves to similar levels to that offered by a standard Transmission Control Protocol (TCP) connection.

To investigate the impact of encryption on the speed of network transfer through a Cumulus SSH connection, a 225MB file was copied through an SSH connection on a virtual machine with and without encryption enabled. A virtual machine was provisioned through Cumulus with two cores and 2GB of RAM. A randomly generated file was then copied from the virtual machine with the SCP command. This SCP was tested both on the internal Host-Only networking to the shared storage and externally through Cumulus to the internet. **Table 4** shows the results, with almost a doubling of the speed over an external network and almost a tripling in the internal network.

Encryption Enabled	Internal Transfer Speed	External Transfer Speed
Yes	16.1MB/s	3.9MB/s
No	45.0MB/s	6.8MB/s

Table 4 – Network Transfer Speed Over SSH: A table showing the transfer speeds of a 225-megabyte file over SSH from a Cumulus virtual machine to an external device in a variety of circumstances. 'Encryption enabled' relates to if encryption of the data over SSH was enabled for the transfer. Internal transfer speed relates to the speed of transfer to the Stembio Drive, a device on an internal network within Cumulus. External transfer speed relates to the speed of transfer to an external server on the internet.

All access to the virtual machine, including internet traffic and access to analysis data on the Stembio Drive, is tunnelled through these SSH connections. To carry out an analysis on a file, the file in question must first be copied to the virtual machine. This is a blocking step and the analysis cannot start until this happens. Any optimisation which can be made to speed up this step decreases the total duration of analysis and improves the usability of the system. In order to disable encryption on SSH, both the client and server must explicitly disable it. Standard SSH clients do not support disabling encryption. Because of this, SSH connections from external sources such as users connecting from the internet will still be encrypted by default. This allows external connections to remain secure despite the removal of encryption internally.

3.6.2 Dynamic Host Configuration Protocol Service

In order to first connect to a virtual machine, the Cumulus service provides it with an IP address over DHCP. In order to provide this, Cumulus uses its connection to the physical processing machine and captures the User Datagram Protocol (UDP) DHCP packets of the starting virtual machine. These DHCP packets are sent from the starting virtual machine over the Host-Only connection to allow it to initially configure networking. SSH does not support tunnelling of UDP packets (Barrett and Silverman, 2001), so Cumulus firstly converts

the UDP packets to TCP packets on the physical processing machine. This conversion is carried out using the tool Socat (http://www.dest-unreach.org/socat). These converted packets are then tunnelled back from the processing machine through SSH to the Cumulus web server.

When DHCP packets are received by the Cumulus server, they are interpreted with Dhcp4java (https://www.dhcp4java.com). This library translates the DHCP to a java object which includes the MAC address of the starting machine. The Cumulus system looks this MAC address up from the database and assigns an IP address accordingly. The IP is wrapped in a DHCP packet by Dhcp4java and returned to the physical processing machine. The physical processing machine sends this DHCP packet over the Host-Only connection to the virtual machine to allow it to set its IP address. The virtual machine then responds with a confirmation DHCP packet and starts using the specified IP. Cumulus is then able to SSH to the provisioned virtual machine.

3.6.3 Internet Proxying

Virtual machines within the Cumulus system have limited access to web resources. This has been implemented using: RabbIT (http://www.khelekore.org/rabbit) and FTP Proxy (https://github.com/c960657/ftpproxy), two Java-based web—and FTP—proxy libraries respectively. The Cumulus server starts these libraries which listen on two separate ports on the server machine. Each virtual machine then has a port on its local machine forwarded with SSH to connect to the open ports on the server machine. Cumulus configures the operating system proxy server settings of the virtual machine to point at this local port. The two proxy systems then have a list of allowed addresses (whitelist) or a list of disallowed addresses (blacklist). Should an address be allowed, the request is forwarded through allowing the virtual machine to access the requested resources. Otherwise the virtual machine will receive an error message that the site is blocked.

3.7 Reporting & Tools

For each study, the system generates a PDF report and set of files to be published with the study. The Java objects for each experiment, the tools used, the analysis tool execution information and virtual disk image identifier are merged by Cumulus into a HTML document. The PDF report is then generated from this Cumulus HTML document with the library iText (https://itextpdf.com) which enables the conversion of the HTML with CSS styles into PDF (Lowagie, 2011). These reports are then available for download from the Cumulus system with both the virtual disk images used and the workflow information. GeneProf allows publishing of experiments which gives a publicly available page and link that can be included

in a journal publication. With Cumulus, when the GeneProf public page is generated, GeneProf requests the links to the additional Cumulus materials from the Cumulus API and augments the GeneProf public page with the links.

4 Stembio CodeLab

The previous chapters have investigated how to enable the publishing of a bioinformatic analysis which can be reproduced and interrogated with a workflow system. These kinds of systems allow the user to specify an analysis using a modular set of tools which can be assembled into a workflow. The analysis system then schedules this analysis for processing with the results returned at some point in the future. An alternative model is an interactive analysis; this is used in tools such as R where an interactive environment allows the user to manipulate the data in real time. This section details Stembio CodeLab; an example application to investigate how reproducible methods can be applied to an interactive analysis tool.

An interactive analysis environment, unlike the batch analysis environment of a workflow system, is time dependent. It refers to environments which provide the near immediate execution of user commands in an on-demand fashion. Through this immediate response, interactive analysis tools enable rapid testing and prototyping of different analysis strategies. A bioinformatic investigation is a research-led process. In order to tease out information from a dataset, a wide variety of different data analysis techniques can be tried and a lot may offer inadequate results. This makes the investigation an iterative process where preliminary patterns are found and refined over many stages of processing. The toolsets used by bioinformaticians reflect this need for rapid iterations and, as such, are commonly interactive. Tools such as R / Bioconductor, MATLAB (http://uk.mathworks.com/products/matlab) and even the Linux shell, provide command line, interactive environments for analysis.

Several web-based interactive analysis environments have been developed in recent years such as RStudio (Racine, 2012). RStudio is an Integrated Development Environment (IDE) for R available both as an installable application or an HTML application hosted on a web server. Both connect to a version of R on the computer or server on which RStudio is installed and provides an interactive console and set of tools. The interactive console emulates the standard R console and is surrounded by tabs providing additional functionality to perform tasks such as load and save data, view objects in the live environment, view plots and help files. The web version of the application is written in C++, HTML and JavaScript and contains a built-in web server to host the web interface and marshal the communication between these components. The application installs a Linux service and works on a one to one basis with an install of R. This link to the installed R makes configuration of multiple non-conflicting environments or users complex. It is also designed to run on a single server rather than a clustered or high-throughput configuration.

The European Bioinformatics Institute (EBI) R Cloud Workbench (http://www.ebi.ac.uk/Tools/rcloud) is another example of an interactive analysis environment. It is a Java application which uses Biocep-R (Furht & Escalante, 2010)—an R virtualisation library—to run R within the EBI cloud infrastructure. R Cloud workbench is more limited in functionality than RStudio but it provides all the standard functions of an R instance. R scripts entered into the system are cleaned of unsupported functions and libraries; in this way it can maintain a list of safe but limited functions. The R Cloud Workbench is high-throughput system which can support many hundreds of users through the EBI infrastructure. The software available in the R Cloud workbench is the latest version of R with Bioconductor installed. This software set whilst functional and kept up-to-date, cannot be extended with new or non-core libraries by non-admin users.

Whilst interactive environments are commonly used by bioinformaticians, they can be problematic in a high-throughput context because of the substantially increased complexity of running them at scale. This complexity occurs for a number of reasons, firstly because of the added demand of a rapid response means that computational resource has to be available as needed. In batch analysis, the work can wait until the processing resource becomes available and spikes in demand can be smoothed out with periods of lower demand. This is not the case with interactive analysis where capacity must be available as it is required. This can cause large fluctuations in the amount of resource required at any one time and so, enough spare capacity must be kept in order to meet the potential demands. Virtualised environments reduce this problem as a single physical machine can be split up and used multiple times simultaneously without the environments being able to affect each other.

CodeLab will use virtualisation through Cumulus to provide interactive, recorded analysis environments. CodeLab will do this by providing an interface to allow other applications, such as RStudio to run within the virtual machine. The UI from these applications will be wired through from the virtual machine using the tunnelled networking—detailed in Section **3.6**—and embedded within an external UI. This will allow the user to install and use web-based interactive analysis tools in the virtual machine and use them from outside the virtual machine. Through recording the environment and interactions with these tools, CodeLab will capture the variable elements of the analysis. This will enable an interactive analysis to be reproducible in the same way as a workflow analysis.

CodeLab will keep a reference to a list of Cumulus virtual disk images which contain specific interactive tools. When a user wants to run one of these tools it will request, through the Cumulus API, a virtual machine running this virtual disk image. Cumulus will have these

interactive services registered with the networking ports they require. Once the virtual machine is launched, these required ports will be tunnelled through to a publicly available port. CodeLab will identify the mapped public port for a service running on the virtual machine then embed the application in its user interface (UI).

Interactive analysis environments such as R provide history so that the user can step back through the steps taken. This history is, however, difficult to communicate to other users. CodeLab will record this history during an analysis enabling it to be communicated in the same way as other elements of variability in an analysis.

This mechanism of automating the process of accessing interactive services on a per virtual machine basis and then recording them enables them to be versioned and recorded in the same way as other analysis software within Cumulus. Should some analysis through R be carried out, the same virtual machine, saved state and history can be started by another user who can then reproduce what they have done.

4.1 CodeLab: Implementation

CodeLab is an example application built to work with the Cumulus API to demonstrate how different types of analysis tools can use Cumulus to enable reproducibility. Like Cumulus, CodeLab has been implemented as an HTML, web-based Java application. It provides access to interactive services such as R, on-demand and in the web browser. The user interface for this is shown in **Figure 21**. Services can be launched within seconds and remain active between different web browsers, browser sessions and devices. CodeLab supports the integration of third-party web-based tools. In addition, several example programming environments have been implemented for CodeLab. These include bash scripting, Perl (https://www.perl.org), Python (https://www.python.org) and R. This section will focus on the R environment.

Once a user logs on and opens an environment, CodeLab makes a call to the Cumulus API to retrieve or start an instance on their behalf. This virtual machine is available and can be managed within the Cumulus interface in the same way as any other instance. The provisioning of this instance may take 30 seconds to a couple of minutes, depending on the configuration a user has selected. CodeLab uses the Cumulus functionality described in **Section 3.2** through its API to manage its software libraries and computational requirement. CodeLab launches a fully-connected virtual container, with both internet and Stembio Drive access, per individual user. CodeLab users can select an image and hardware profile from the Cumulus library which will be used for the CodeLab session. Alternatively, users or groups of users can be restricted to pre-defined images and hardware profiles. This means

standardised software configurations can be set up in Cumulus and used for separate CodeLab sessions. For example, if a tutorial were to be set up to teach microarray analysis, the required tools to carry out this analysis can be set up on an image and this image locked to the class of users.

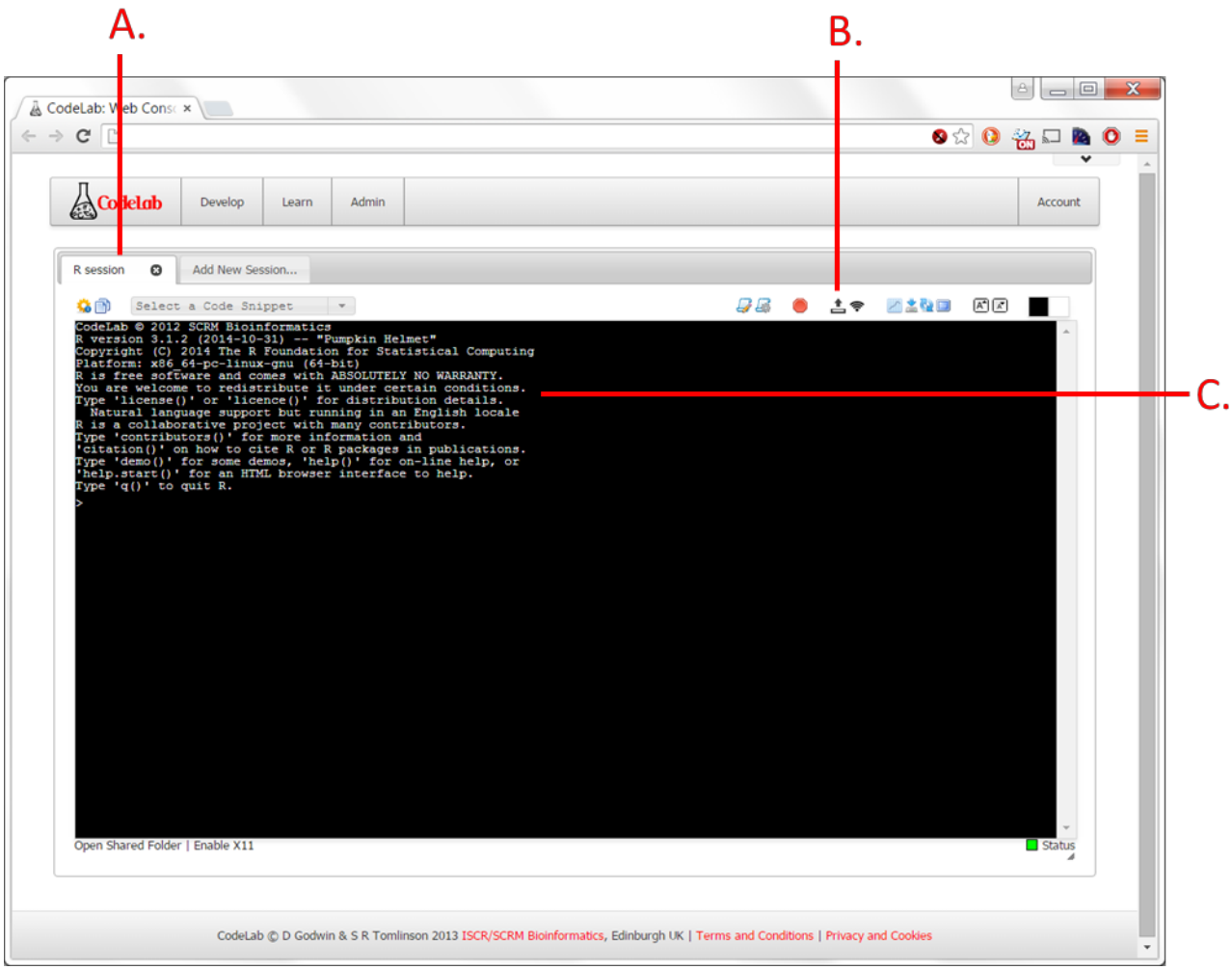

Figure 21 – Stembio CodeLab Live R Console: A screenshot showing the interactive R console user interface within CodeLab. (**A**) Tabs to navigate between the currently running interactive sessions and an 'Add New Session' button, which will create a new interactive session. (**B**) Controls to upload, create and edit scripts, broadcast the current interactive session to other users or monitor resource usage (**C**) The CodeLab interactive R console showing the R starting text. The user can type commands here which are executed on a virtual machine within Cumulus. The results are shown in the browser console window, emulating the functionality of an R console in a web browser.

The CodeLab instances are launched with internet access and the home directory set to the Stembio Drive. Any files the user then creates will be saved by default to the users shared space and will be retained when the CodeLab instance is ended. This container is kept running and connected to networks until the user ends it or CodeLab identifies the user is no longer using the system. CodeLab identifies unused instances by the processing load of the virtual machine and the size of instance required. In general, a virtual machine will be closed

a few days to weeks after it was last used. Once CodeLab has started a running instance, it connects to it via SSH. Any required web interfaces are tunnelled through the SSH connection to the running applications. A single, linked web interface and interactive programming environment are collectively referred to as a CodeLab Session. Multiple CodeLab Sessions can be run simultaneously by the user on a single Cumulus instance and the user can swap between them.

4.1.1 CodeLab Integrated Tools

An initial selection of embedded tools have been installed and configured for CodeLab including:

- shellinabox (https://github.com/shellinabox/shellinabox), an alternative shell implementation.
- R Studio (Racine, 2012) an IDE for R.
- Tailon (https://github.com/gvalkov/tailon), a web-based tool for displaying and searching through log files.
- NoVNC, a web based Virtual Network Computing (VNC) client (https://github.com/novnc/noVNC).

In addition to these installed tools, a number of example programming environments have been implemented for CodeLab through a component called the CodeLab Live Console.

The CodeLab Live Console is a jQuery plug-in which emulates a Unix-based command line environment in a web browser. Each console is a CodeLab session and relates one-to-one to a particular execution of a command line tool on the running instance. The command line tool can also be run through Screen, a Linux tool which enables persistence of sessions between disconnections. This allows the Cumulus or CodeLab systems to be stopped and started without ending the CodeLab Sessions.

Commands can be entered by the user into the CodeLab live console plug-in, the web interface to CodeLab, and these commands are relayed to the associated running session as standard input. As text output is produced by the application in the running session, this is sent back to the console plug-in through the use of Comet and displayed in the plug-in window. Comet is a web application communication model used for server to client communication. Web requests are normally initiated by the web browser which can make it quite difficult to deal with events initiated by the web server. In this model, long running HTTP requests are held open by the client and when the server is ready, it sends a message back through this still open HTTP request. This produces the effect of a live, responsive, interactive console window in the browser. The CodeLab Live console is built as a jQuery

plugin so that it can be included on other external sites. A website can include a short script which embeds the console onto the HTML page. This then acts in the same way as in the CodeLab application with a live console linked to the user's active instances. Other scripts on the website can also interact with the console or run scripts within the user's instance. This allows external services to use the full functionality of CodeLab in their own services.

Each environment type can have special commands defined which are captured and relayed separately to the environment. Ctrl+C, for example, stops the currently running process in the bash shell. Command histories can be replayed through pressing the up and down keys. These small pieces of functionality aim to make the emulation of the command line environments as realistic as possible. The aim of CodeLab is to enable reproducibility and scale whilst using known bioinformatics tools and working practices. Ensuring that these tools function in the same way in the CodeLab platform means there will be no additional complexities to learn for the user and they will be able to carry out the same standard of analysis.

4.1.2 CodeLab Graphics

The CodeLab Live Console aims to emulate standard command line environments as closely as possible in web browsers. Graphics are a key feature of command line tools such as R and in order to emulate this functionality, CodeLab provides X11 sessions. X11 is a method to interact with graphical user interfaces in UNIX based operating systems. It is a protocol which allows the graphical interface of an application to be displayed and interacted with on any computer whilst the application itself can be running elsewhere (Scheifler, 1987).

CodeLab has a Java applet which provides X11 display support based on WeirdX (http://www.jcraft.com/weirdx) which runs in the user's browser. This communicates through the Comet HTTP connection of the CodeLab web page connecting to an X11 proxy server hosted by Cumulus. This proxy server links up each X11 connection to the correct Cumulus instance, relaying data between this and the web browser. This applet enables low frame rate, interactive graphics in the browser for the console sessions. The browser Comet connection encodes the binary X11 data to text whilst it is communicated between the separate parts of the application. This conversion lowers the bandwidth for communication to less than standard X11 leading to a graphical refresh rate of around 3-4 frames per second (FPS). Higher resolution and more complex applications suffer from this lower frame rate and are less usable. For the primary use of emulation of the display and editing of graphs in the R environment, however, this frame rate is workable. When the user enables the X11 applet, any graphics which are then opened by the server-side applications produce a responsive Java window in the browser which displays them, shown in **Figure 22**.

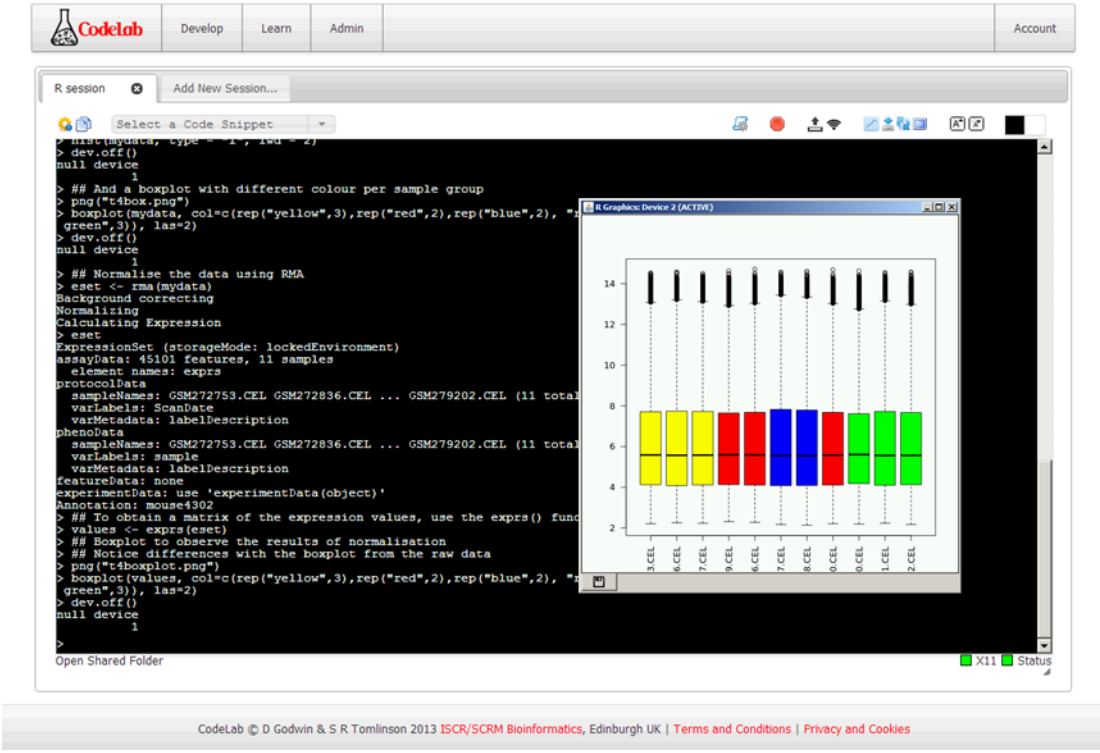

Figure 22 - Stembio CodeLab Integrated X11 Graphics: A screenshot showing the CodeLab console in which a microarray analysis has been run. This microarray analysis produced a box and whisker quality plot. R has displayed this on the standard graphics device. Cumulus has captured this request and opened the graph in the X11 browser window.

4.1.3 Snippets and Code Blocks

Snippets or Code Blocks are user-created scripts stored within CodeLab to enable the automation of different functions. Snippets are short sections of code, run from a drop-down menu in the console itself and provide relatively simple pieces of automation such as loading or formatting data. This kind of automation is used in CodeLab to create user-defined functions to aid analysis. The aim of these is to simplify the analysis process. Code Blocks are longer sections of code which may form part of, or a whole analysis. Users can save their own code blocks or snippets or contribute them for public use. Both types of scripts allow defined inputs connected to variables in that script. This allows the system to run the script presenting only input parameters or, alternatively, users can select the inputs and also edit the script. The user interface for this is shown in **Figure 23**.

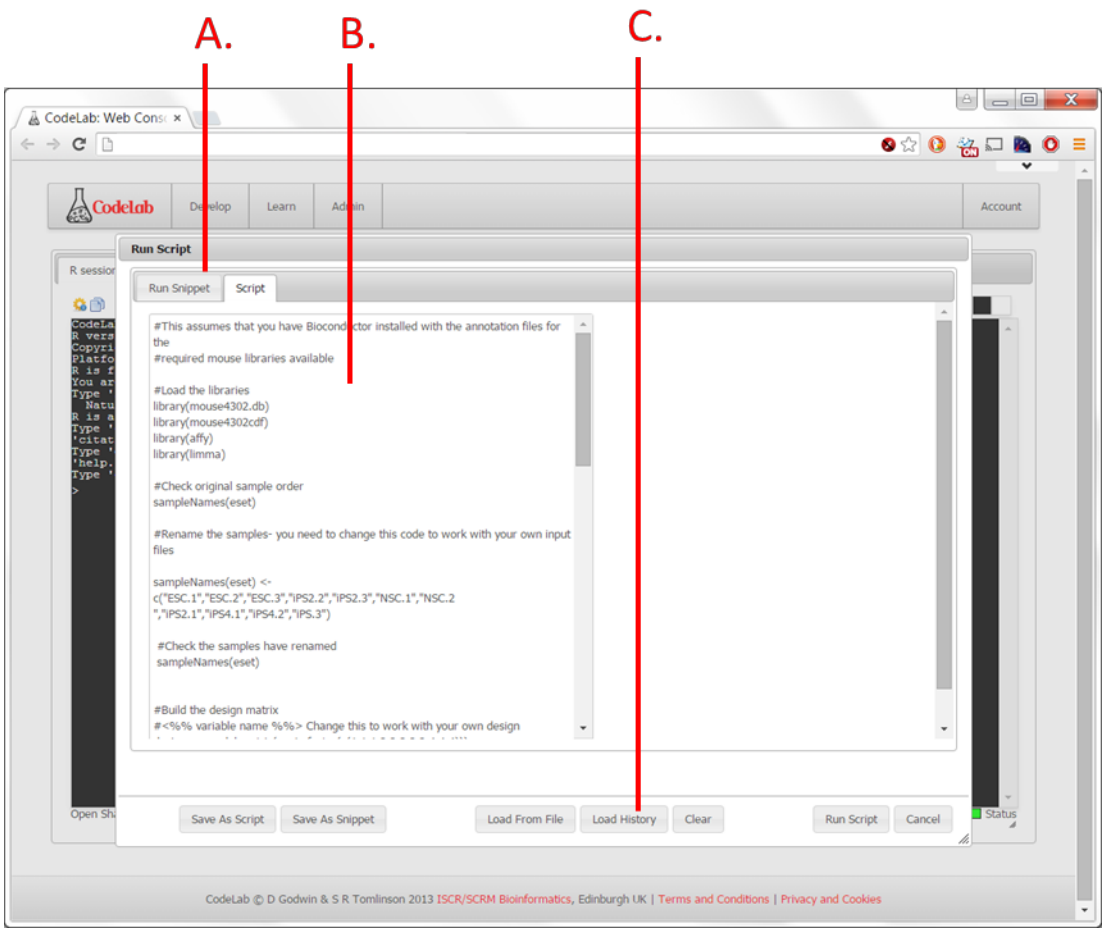

Figure 23 - Stembio CodeLab Snippets: A screenshot showing the editing and running of a code snippet of a microarray analysis within CodeLab. (**A**) Tabs to swap between execution and editing of the script. On the 'Run Script' tab, input fields are shown for embedded markers within the script. The values entered are then swapped for these embedded markers when the script is run. (**B**) A window to edit the script before execution. (**C**) Controls to load a script from a file or from the history of another analysis.

An interactive analysis is stored by CodeLab as a script of all the commands entered and a log of the resultant output from the execution of that script. The script or output for each session can be viewed by the user and re-used in part or in full as code blocks or snippets

4.1.4 Process Monitoring

CodeLab has the ability to monitor the CPU and RAM resources actively used by a virtual machine. When monitoring is enabled on an instance, the amount of these resources used is logged every few seconds. The user can view this through a real time updating graph shown in **Figure 24**, available from a button on the console component. This gives a window onto both the live activity of an instance, but also historical data over the whole of an analysis. Commands entered by the user into one of the sessions are automatically mapped onto this, showing the computational impact of different parts of a script.

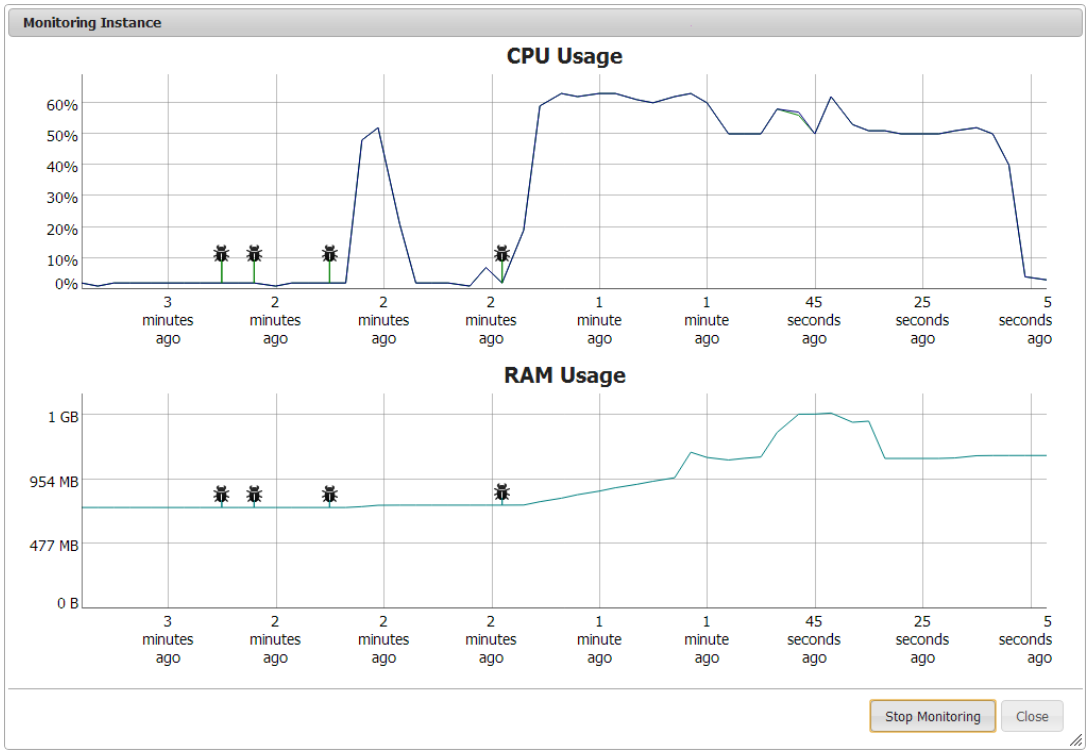

Figure 24 - CodeLab Instance Monitoring: A screenshot showing the monitoring interface for CodeLab. The interface shows two-line graphs. The top shows CPU usage on the CodeLab virtual machine. The bottom shows RAM usage. The live console interactions are mapped to the instance usage profile and appear as 'bug' markers. As commands are typed in they can be related to changes in CPU or RAM usage.

4.1.5 Reproducible Research Enabled by CodeLab

One of the main aims of both the Cumulus and CodeLab systems is to produce analysis applications with reproducibility as a core function of the system. At every stage of the analysis, the actions, tools and inputs used are recorded. As CodeLab sits between the user and the running process it allows CodeLab to log all of the details of the interaction between the user and the running process.

The commands typed by a user, or the options selected at each stage of the analysis, are a variable component which need to be recorded during an interactive analysis. Reproducibility is maintained within CodeLab by recording these interactive inputs and outputs of a process in real time. As a command is entered through the keyboard or text is returned to the user this is logged, producing a history of the entire process. This can then be combined with the other variables captured by Cumulus such as software and hardware versions to give a complete record of the analysis. The user interface for this part, with an example list of session records is shown in **Figure 25**.

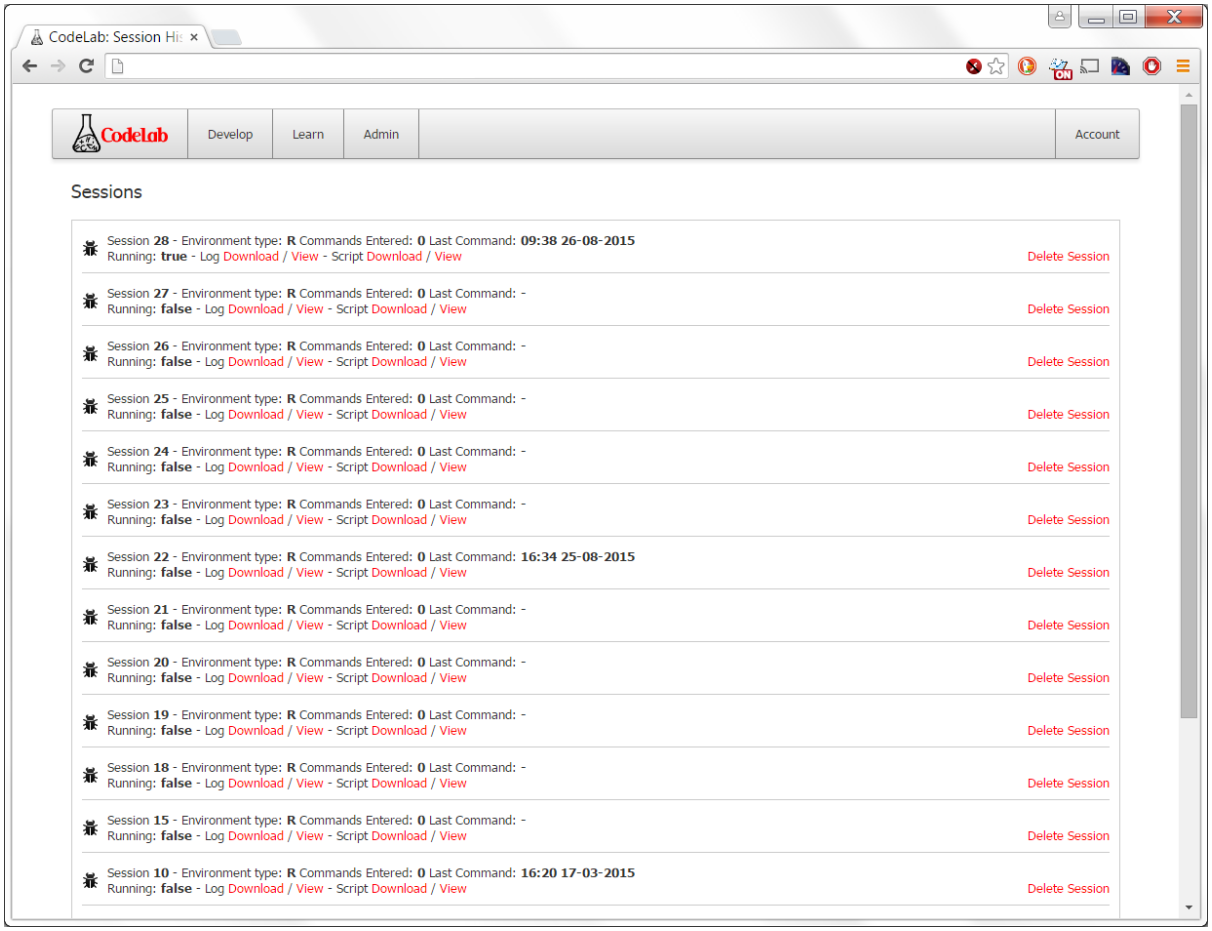

Figure 25 - CodeLab Session History: A screenshot showing the histories of a user's interactive R sessions. The history contains a log and a script. The script is all of the commands entered into a session. The log is all of the commands and the resultant output from R. Each session history can be viewed, downloaded or used as the basis for a new analysis in a code snippet.

5 Stembio Visualisation

Chapter 2 described the design and **Chapter 3** the implementation of Cumulus, a system to enable reproducible analysis within workflow systems such as GeneProf. GeneProf includes a number of visual, web-based, interactive data mining tools which allow the user to further analyse processed datasets. Cumulus provides a number of new APIs which enable applications to interact with data sources such as the Stembio Drive. The Stembio Visualisation framework has therefore been developed to further investigate and facilitate the development of new visualisations using the Cumulus system.

5.1 Stembio Visualisation: Implementation

The Stembio Visualisation framework provides a set of tools which enable developers to create, share and publish interactive web-based visualisations. This structure builds on the application platform described in **Section 3.2**, allowing researchers to combine a small amount of their own visualisation code with stock methods that reduce the repetitive work involved in the creation of a new visualisation. These stock methods include API generation and server-to-browser communication detailed in **Section 3.2**, in addition to a set of APIs for accessing and using data from the Cumulus and GeneProf databases. The developer creates their code using these stock methods and specifies what types of data it will display. The visualisation framework stores their work in a database of visualisations which can be accessed by other tools.

Once packaged in the visualisation framework, the visualisations can be used in a generic way throughout tools such as Cumulus and GeneProf. These tools specify a Picture Frame in which a certain type of data may be displayed. Visualisations which are able to display this data-type can then be dynamically selected. For instance, GeneProf may display a picture frame and provide a gene name, one visualisation may display a three-dimensional representation of the gene structure, another may show relationships between this gene and other genes. The visualisation framework controls the relationship between this picture frame and the components which are displayed within it. In this way, new visualisations can be written and incorporated into tools such as GeneProf without re-writing GeneProf itself.

Like Cumulus, the Stembio Visualisation framework has been implemented as an HTML, web-based Java application. In order to create a new tool within the framework, a developer must implement their own server-side Java object and JavaScript presentation object with the remaining structure provided by the framework.

In order to further enable development, a development environment and a set of development tools have been created for use by the programmer. The development environment is based on a custom, structured project for the Eclipse IDE (https://eclipse.org). Apache Ant (http://ant.apache.org), a Java automation tool, is used to automate the build process of the visualisations.

The main component of the Stembio Visualisation framework runs as a server-based database application which allows users to upload projects using Ant and then hosts them on a web server. Ant compiles and uploads the visualisation components from the project to the correct place on the server. Once Ant has been configured for the project, it allows the developer to rebuild and deploy visualisations automatically. This structure is detailed in **Figure 26**. Each visualisation has a web page on the server with debugging tools and logging for the server-side Java. These visualisation tools are presented as a database by the Stembio Visualisation framework which can be embedded into other applications.

Visualisations can load data using the Java server component from the Stembio Drive, the Cumulus API or the GeneProf API. The data loaded from these sources can be combined or modified and sent to the web browser through DWR as described in **Section 3.2.**

A web resource describing gene expression in the developing HSC niche has been produced using the Stembio Framework (http://agmniche.stembio.org, McGarvey et al., 2017) thus validating the effectiveness of the approach and the usefulness of the visualisation framework

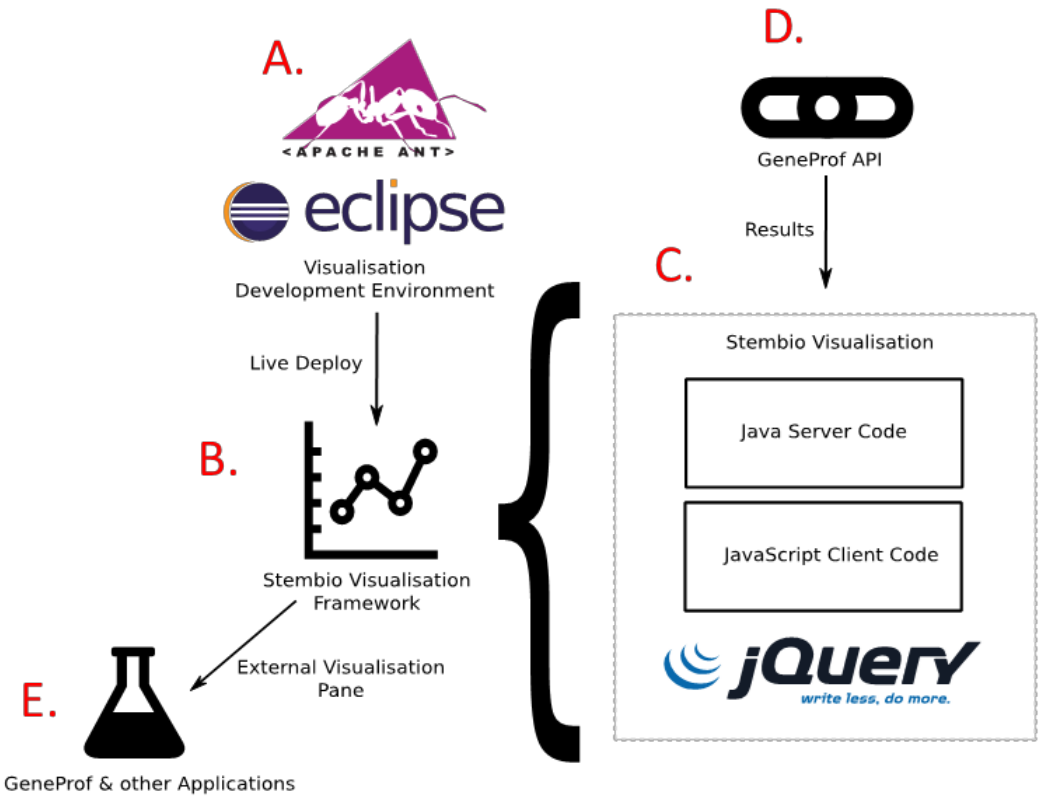

Figure 26 The Stembio Visualisation Framework: A flow diagram showing the structure of the Stembio Visualisation Framework. (**A**) A development environment is provided for Eclipse based on Apache Ant. (**B**) Each Eclipse project is linked to a project on the Stembio Visualisation framework web application. (**C**) The project structure is a Java object which runs on the server to format and provide data, and JavaScript code which runs in the web browser and receives the formatted data. (**D**) The Java object or the JavaScript can read results from the GeneProf API, Stembio Drive or other applications. (**E**) The visualisations can be directly embedded into external applications or websites.

6 Reproducing an Analysis

The overall goal of this work is to investigate how to enable the publishing of a bioinformatic analysis which can be reproduced and interrogated by a third party. The Cumulus system enables a researcher to carry out a bioinformatic analysis and then simplifies the process of reproducing that analysis. It accomplishes this by capturing the elements of potential variability within the analysis. These elements are packaged so they can then be published with the results of the experimental study. This chapter will discuss and demonstrate a novel example in which this work enables and improves the ability of researchers to carry out a bioinformatic analysis and interrogate the analysis from other scientific studies.

This thesis uses GeneProf as an example workflow system to integrate with Cumulus and, in combination, enable users to carry out a reproducible experiment. At an early stage of this project, a Cumulus virtual disk image was created for the GeneProf system which contained all of the tools and dependencies required for a specific GeneProf analysis. In order to compare the reproducibility of a study created in GeneProf alone to one with the Cumulus system, the process of installation, configuration and execution of these tools will be followed for an analysis in each system.

In the paper describing GeneProf (Halbritter et al., 2012)., the authors combined a re-analysis of two previous studies (Chen et al., 2008; Guttman et al., 2010). The Halbritter analyses were documented in GeneProf with accession numbers gpXP_000168 and gpXP_000012, available from the GeneProf site. These studies were used as examples to investigate the functionality of the Cumulus system. At this point of carrying out this investigation, these re-analysis experiments are six years old. Using the two systems, what comparative difficulties are there in reproducing these studies as they appear in the publication?

At the point of publication for the Halbritter et al. (2012) paper, the pipeline used for these studies was also used to create a Cumulus virtual disk. The installation for this was carried out using the published GeneProf installation instructions on a Cumulus virtual disk build environment. This disk image has since been stored in the Cumulus database and has been used to carry out the comparison detailed here.

6.1 Evaluation Methods

All of the RNA-seq analysis reproductions in the following section used the same analysis methods aside from the alignment stage. Firstly, the raw samples were downloaded from the SRA records listed in the publication using the SRAToolkit (Leinonen et al., 2011) version

2.1.16. The adapters were removed from the reads with Trimmomatic (Bolger et al., 2014) using the default supplied adapters file TruSeq3-PE-2.fa. The *Mus musculus* reference genome, Ensembl, assembly NCBIM37, annotation version 58 was downloaded (ftp://ftp.ensembl.org/pub/release-58/fasta/mus_musculus). For each aligner, the reference genome was used to create an index using the default mechanisms. Alignment was then carried out with HISAT2 version 2.1.0 (Kim et al., 2015), STAR version 2.5.2b (Dobin et al., 2013) and TopHat version 1.2.0 (Trapnell et al., 2009). FeatureCounts, part of the Rsubread library version 1.26.1 (Liao et al., 2014) was used to assign reads to features within R version 3.1.2. All tools used are listed with their versions and any specified parameters in **Table 5**.

Tool	Version	Specified Parameters
SRAToolkit	2.1.16	
Trimmomatic	0.36	SEED_MISMATCHES = 2 PALINDROME_CLIP = 30 SIMPLE_CLIP = 10 PALINDROME_MAL = 4 PALINDROME_KBR = true LEADING = 3 TRAILING = 3 WINDOW_SIZE = 4 WINDOW_QUALITY = 15 MINLEN = 36 HEADCROP = 1
HISAT2	2.1.0	MAX_ALIGN_PER_READ = 10
STAR	2.5.2b	
TopHat*	1.2.0	-r 140 --mate-std-dev 30 -g 10 --segment-mismatches 2
R	3.1.2	
featureCounts (Rsubread)	1.26.1	isGTFAnnotationFile = TRUE GTF.featureType = "exon"

Table 5 - Analysis Tools and Parameters: A table showing the tools and the versions used in the evaluation. Any parameters specified beyond the defaults are listed in the 'Specified Parameters' column. * the g parameter was changed to g=1 for the modification of the analysis in **Section 6.8**

6.2 Test Environment

The test environment for this comparison was a Dell x86 server with Ubuntu 16.04 LTS as the OS. This platform was chosen as the original GeneProf system required an x86 server

and an unspecified Linux distribution. Ubuntu 16.04 is the currently available long-term support (LTS) version of the Ubuntu operating system.

In order to reproduce a study in the original GeneProf system, there are two available procedures. The user can run their analysis on a hosted version of the system, or setup and configure their own version of the GeneProf system. These two methods can be regarded as the same, as both a hosted version and independently-run version of GeneProf require someone to install and configure or maintain the system. GeneProf has a set of instructions and scripts for doing this in which it lists the set of tools used in the analysis. These tools are shown in **Table 6**.

6.3 Software Versions

Software Name	GeneProf Version	End Of Life (EOL)	Latest Version (2017)
Java	1.6 (6)	February 2013	9
Apache Tomcat	6	December 2016	8.5
MySQL & Connector	5.1.12	December 2013	5.7
R	2.12		3.4.2
ImageMagick	6.5.7-8	May 2011	7.0.7
TeX Live	2011	June 2012	2017
GraphViz / dot	2.26.3		2.40.1
BowTie (Langmead et al., 2009)	0.12.3		1.2.1.1 / 2.3.3
BedTools (Quinlan and Hall, 2010)	2.10.1	December 2014	2.26.0
CCAT (Xu et al., 2010)	2.0 ‡		N/A *
FASTX-Toolkit	0.0.13		0.0.13
MEME (Lesluyes et al., 2014)	4.8.1		4.12.0
MACS (Zhang et al., 2008)	1.4.2		2.1.0
SAMtools (Li et al., 2009)	0.1.19		1.5
SISSRs (Narlikar and Jothi, 2012)	1.4 †		1.4
SRA Toolkit (Leinonen et al., 2011)	2.1.16		2.8.2
TopHat	1.2.0		2.1.1

Table 6 - GeneProf Software Requirements: A table showing the software requirements of the GeneProf system reproduced from the GeneProf installation manual (http://www.geneprof.org/GeneProf/help_advancedtopics.jsp#chapter:AdvancedTopics), the end of life (EOL) date (where specified) on which the GeneProf version stopped being supported and the current active version of the software.
‡ The GeneProf system also applies a Mersenne Twister algorithm to improve the random number generation

* The CCAT published URL is no longer available.
† The GeneProf system applied a custom patch to the SISSRs install.

Several of the tools required by GeneProf have reached end of life (EOL). Under these circumstances, the organisation releasing the software no longer fixes problems which arise or ensures compatibility with available operating systems. In some cases, especially with a perceived risk of system security, organisations can entirely block the release of software. Java 1.6 for example is one of the core requirements of the GeneProf system; the manual states “Sun/Oracle VM recommended”. This version of Java has reached its end of life and is no longer available to install without a purchased support agreement with Oracle. Subsequently it has been removed from all public software repositories (Andrei, 2017). In addition, where organisations do still release old versions, compiled, binary versions of the software are often not available. The SRA toolkit for example only releases binary versions of the latest releases. This means that users need to compile from the source or download from third parties. Compilation of source software can often be a somewhat more complex process than downloading a pre-compiled binary file. The user needs to install and manage the toolchain used by the developers of the system and manually install an often-complex set of dependencies.

As can be seen in **Table 6**, software libraries and tools are developed at substantially different rates. Less-frequently released packages are often funded through academic research with less resource. Unfortunately, due to their smaller user base and less widespread use, these tools can sometimes disappear or become unavailable. The tool CCAT, for example, was unavailable at the published URL at time of writing. Studies are reliant upon the continued availability of the software tools they have used in order to be reproducible. When a tool such as this is hosted on a small website and becomes unavailable, it is very difficult to obtain a copy in order to repeat an analysis.

6.4 Installation

Each of these tools must be downloaded and installed on to the target system. For this, GeneProf provides a number of scripts which download and configure the build process where required for each piece of software. In addition, patches are applied to specific programs such as SISSRs and CCAT which is compiled with the Mersenne Twister with improved initialization (http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html). A number of these scripts have become invalid over time. Repositories for different tools have moved between hosting platforms. SAMtools, for example, was hosted on the now-redundant Google Code (https://code.google.com/archive)

but has moved to GitHub. These changes, detailed in **Table 7**, have reduced the usability of these scripts to reference material for installation.

One feature of the list of software specified by GeneProf as shown is the limited extent of these specifications. No operating system distribution or version other than ‘Linux’ is specified, for example. Lack of system specification is problematic for installation of the system in two respects. Firstly, different operating systems will have different versions of underlying software libraries. As discussed earlier in this thesis, different libraries and code can have different requirements, function differently and give different results. Secondly, installation of software libraries can often be more difficult on operating system versions from a different time period.

Software	Installable Binaries	Source Compiles	Executes
Java	Partial		✓
Apache Tomcat	✓		✓
MySQL & Connector	✓		✓
R	✓		✓
ImageMagick	×	✓	✓
TeX Live	✓		✓
GraphViz / dot	✓		✓
BowTie	✓		✓
BedTools	×	✓	✓
CCAT	×	×	×
FASTX-Toolkit	✓		✓
MEME		✓	✓
MACS	✓		✓
SAMtools		✓	✓
SISSRs	✓		✓
SRA Toolkit	×	✓	✓
TopHat		×	×

Table 7 - The Current Status of GeneProf Software Dependencies: A table listing the GeneProf software dependencies sourced from the GeneProf installation manual and their current state of usability. A tick in the 'Installable Binaries' column indicates that there were compatible binary files available for download from the authors. A cross indicates that binary files were not available. A blank space indicates that GeneProf requires these files to be built from source. 'Partial' indicates that although binary files were not available, suitable alternatives could be found from other sources. A tick in the 'Source Compiles' column indicates the source files compiled successfully. A cross indicates that they did not compile or that compilation did not result in executable binaries. A space indicates that binary files were available for download so compilation was not attempted. A tick in the 'Executes' column indicates that the downloaded or compiled binary files executed successfully. A cross and pink shading indicates that they would not execute, did not function properly or were not available due to a problem encountered during a previous step.

Although Oracle Java 1.6 is unavailable, other implementations of Java 1.6 such as OpenJDK (http://openjdk.java.net) are available for download, and work to the same specification. ImageMagick binary files are no longer distributed for version 6.5.7; however, the source binary files can be built and installed. The same is true for MEME, the SRA toolkit, BedTools and SAMtools and these were successfully built from source. MACS deb installation did not work due to dependency problems though, again, compilation of the

source files generated usable software. This software stated that it was last tested on Ubuntu 10.04 and requires a Python 2.6 interpreter which is not available by default. These are all relatively minor complications but would still require both significantly greater time to resolve and a user with a relatively good knowledge of Linux.

TopHat is compiled by the GeneProf script with a link to SAMtools. This process failed, both using a binary download, or compilation from source. The compilation from source failed with errors relating to undefined functions. This error implies an incorrect configuration of build files or missing dependencies. It was a significant barrier to using the tool and blocked further use. Whilst these problems could undoubtedly be solved with significant further time and expertise, each of these problems complicates the process of reproducing the study. These complications make it unfeasible to reproduce the study within a reasonable timeframe.

The GeneProf study used a 2010 paper by Guttman et al. to demonstrate the RNA-seq component of the system. TopHat was one of the main tools used within this RNA-seq pipeline and the inability to run this implies the published analysis is not reproducible. Incompatibilities such as these develop over time, with increments in operating systems and application interfaces. It is this divergence which makes the process of reproducing an analysis process on a more modern system extremely difficult.

6.5 PXE Disk Deployment

The Cumulus system provides two methods of reproducing the GeneProf analysis environment. The analysis can be reproduced using the Cumulus API to provision and connect to a virtual machine or a virtual disk image can be written to a physical disk using the PXE image deployment. In this instance, the Cumulus system was used to re-provision a GeneProf analysis environment and connect to it as a GeneProf analysis machine.

The first stage of deploying the GeneProf tools using Cumulus is to enable booting from PXE on the physical machine. Booting a server from PXE is an option in the BIOS, commonly accessible by pressing a key on start-up. The next stage is registering the server with Cumulus and entering the MAC address of the physical network device. This MAC address is also shown in the system BIOS, at system boot or in any installed operating system. Once this is done, the disk image is selected in the Cumulus user interface for the physical machine and the machine restarted. The physical machine then downloads the image from Cumulus and installs it to its local disk. Cumulus then provides SSH log-in details for the physical machine and it becomes available as a GeneProf analysis machine.

After carrying out the PXE deployment process and logging on to the physical machine, all of the libraries specified for the analysis were already installed and available. The results for testing their execution is shown in **Table 8**. This demonstrates that—using Cumulus through GeneProf—the experiment can be re-opened in GeneProf and the analysis environment still be available as it was when the experiment was first run.

Software Dependency	Installed Version	Executes
Java	1.6	✓
Apache Tomcat	6	✓
MySQL & Connector	5.1.12	✓
R	2.12	✓
ImageMagick	6.5.7-8	✓
TeX Live	2011	✓
GraphViz / dot	2.26.3	✓
BowTie	0.12.3	✓
BedTools	2.10.1	✓
CCAT	2.0	✓
FASTX-Toolkit	0.0.13	✓
MEME	4.8.1	✓
MACS	1.4.2	✓
SAMtools	0.1.19	✓
SISSRs	1.4	✓
SRA Toolkit	2.1.16	✓
TopHat	1.2.0	✓

Table 8 - The Current State of The GeneProf Cumulus Virtual Disk: A table showing the GeneProf software dependencies installed on the GeneProf Cumulus virtual disk image. A tick in the 'Executes' column indicates that the software tool executed successfully.

Thus, using the Cumulus system to store the analysis environment of the study has enabled the analysis environment and all of the tools to still function six years on from the original publication. Since it proved impossible within the timeframe of this project to reproduce the environment de novo from specified components, it demonstrates the usefulness of the Cumulus system to replicate analyses.

6.6 Analysis Reproduction with Modern Tools

To determine to what extent the ability to replicate the original analysis affected the output, a reproduction of the Guttman et al. 2010, RNA-seq re-analysis in GeneProf was carried out with equivalent, currently available aligners. These aligners were used to reproduce the TopHat alignment carried out in the original GeneProf analysis. Firstly, HISAT2 version 2.1.0 was tested. HISAT2 is an aligner released by the same research group as TopHat, which has largely superseded it. Secondly, STAR version 2.5.2b was tested. STAR is a different, fast and commonly-used aligner. The default settings for both aligners were used. The reference annotation used for both was the same as the GeneProf analysis, detailed in **Section 6.1**. Reads from these aligners were then assigned to features of type 'exon' in the annotation and grouped by gene_id using the tool featureCounts. HISAT aligned 96.45% of reads, 65.8% of these were then assigned to features. STAR aligned 92.1% of reads, 63.8% of which could then be assigned to features.

The table of the top 20 most expressed of these features for ESC are shown in **Table 9**. This allows us to identify how similar the alignments produced by each of the alignment methods are. The similarity of this alignment shows how easily we can reproduce the alignment using these modern tools. As **Table 9** shows, the original analysis gene list is not recreated by the modern aligners.

Rank	GeneProf	HISAT	STAR
1	Eef1a1	Eef1a1	Eef1a1
2	Hsp90ab1	Hsp90ab1	Hsp90aa1
3	mt-Co1	Hsp90aa1	mt-Co1
4	AC092404.2	Hspa8	Hsp90ab1
5	AL840626.1	mt-Co1	Npm1
6	AL606724.1	Npm1	Ncl
7	Ncl	Ncl	Hspa8
8	Hsp90aa1	Rpl4	Pabpc1
9	Eef2	Eef2	Eif4g2
10	AL646054.2	Pabpc1	mt-Cytb
11	AC110186.1	Dppa5a	mt-Nd5
12	mt-Co3	Atp5b	Eif2s2
13	Dppa5a	Pkm2	Rpl7
14	Atp5b	Rpl7	Rpl4
15	AL833805.4	Rplp0	Hnrnpu
16	Gnb2l1	Rps4x	Hspd1
17	Tpt1p	Gnb2l1	Hnrnpa2b1
18	Rpl4	Actb	Atp5b
19	Tubb5	Tubb5	Dppa5a
20	AC124976.1	Eif4a1	Tpt1

Table 9 - Top 20 Embryonic Stem Cell (ESC) Genes Re-Analysing Guttman et al. Using Different Aligners: A table showing the top 20 Embryonic Stem Cell (ESC) genes ordered by rank as reported by different aligners when re-analysing the data described by Guttman et al. (2010). The 'GeneProf' column indicates the data from the original published GeneProf study which used the TopHat 1.2.0 aligner.

The mismatch of the top 20 genes shown in **Table 9** could be caused by a slightly different ordering of the similarly expressed genes. To check for this, the top 20 genes from the GeneProf analysis were compared to an increasing number of the ranked genes from the other aligners. This is shown in **Figure 27**. Whilst re-ordering of the gene lists does account for some difference between the lists, the overlap in the aligners is still significant. STAR aligner only finds 11 of the top 20 genes found by the GeneProf alignment in its top 1000. It has previously been shown that different aligners do not give the same results for the same dataset (Nookaew et al., 2012) so this result is not unexpected. These results show that to reproduce the same experiment using modern tools and get a similar result is far from trivial and, in some cases, it may not be possible.

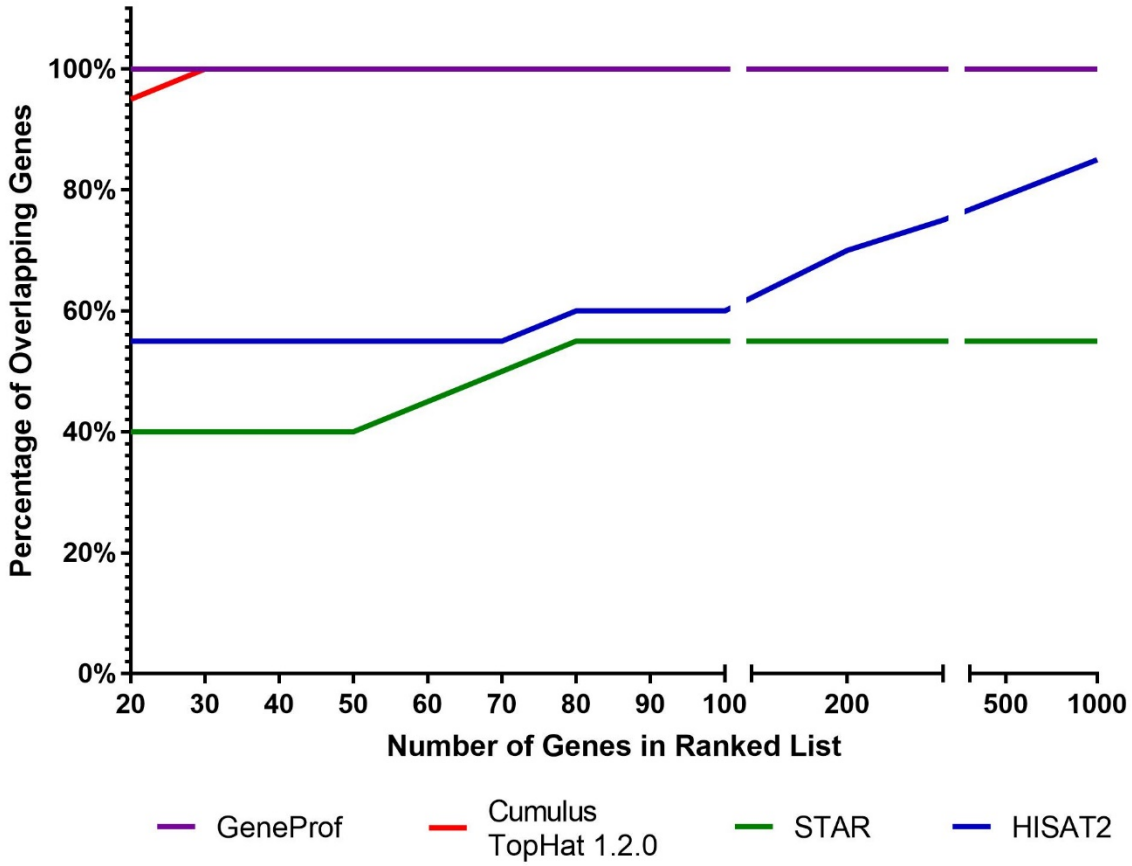

Figure 27 – The Overlap of Aligned ESC Gene Lists Comparing TopHat 1.2.0 in GeneProf to TopHat 1.2.0 in Cumulus and Other Common Aligners: A line graph showing the percentage overlap between the top 20 ESC genes from the published Guttman et al. GeneProf re-analysis and an increasing number of the ranked gene lists from TopHat 1.2.0 in Cumulus, STAR aligner 2.5.2b and HISAT2 2.1.0.

6.7 Analysis Reproduction with Cumulus

In order to investigate how well the Cumulus system can reproduce a study, the same RNA-seq alignment carried out in the Guttman et al. GeneProf re-analysis was repeated in Cumulus. As detailed in **Section 6.5**, the Cumulus system has TopHat 1.2.0 in the software tool library. This version was executed using the default aligner parameters and the *Mus musculus* genome from Ensembl, assembly NCBIM37, annotation version 58, in the same way as the GeneProf experiment. Using the same aligner and configuration through Cumulus as was used in the initial analysis resulted in a much better match to the original analysis than the modern aligners in **Section 6.6**. The results of the overlap of the top 20 GeneProf genes with the Cumulus TopHat 1.2.0 alignment are shown in **Figure 27**. The small discrepancy between the Cumulus alignment with the original GeneProf result is

possibly due to the TopHat random seed. The random seed was not recorded for the original analysis and thus differed in the re-analysis which can have an impact on the results (Trapnell et al., 2009). These results demonstrate that using Cumulus to reproduce a study provides far greater degree of reproducibility than can be easily achieved through reproducing a pipeline manually or using alternative tools. As has been shown, analysis tools can often not be exchanged without affecting the results of a study substantially. Using Cumulus to reproduce the same toolset gives a far closer reproduction of the original analysis.

6.8 Modifying an Analysis with Cumulus

In **Section 6.6** it was shown that re-analysis of the original GeneProf study using modern tools gives a different output. Eight of the top ten genes could not be found in the top one hundred genes from either the STAR aligner or HISAT2. These genes are: AL840626.1, AL606724.1, AL646054.2, AC110186.1, mt-Co3, AL833805.4, Tpt1p and AC124976.1. The majority of these genes are pseudogenes. Pseudogenes are segments of DNA that are related to real genes but have lost at least some functionality, relative to a complete gene (Vanin, 1985). The genome contains large areas of high repetition (de Koning et al., 2011). It has been shown that when an aligner attempts to map a read in one of these regions, it can map to multiple areas and incorrectly increase the number of reads assigned to pseudogenes (Kim et al., 2013). The STAR and HISAT2 aligners account for this effect in their methods and so these genes may not appear as highly ranked in their results. The default settings used by the GeneProf system allows these multi-mapping genes to be assigned to up to 20 different locations. The effect of multiple mapping to pseudogenes could be the cause of some of the discrepancy between the original study and the re-analysis using modern aligners.

To investigate the impact of multi-mapping reads, the alignment from the GeneProf re-analysis was re-run through TopHat 1.2.0 in Cumulus. The Cumulus system allows a user to select a specific tool and run it against a dataset with selected inputs. In this instance, in order to limit the mapping of reads to only one location, the “-g 1” parameter was set (Trapnell et al., 2009). As is shown in **Figure 28**, making this correction brings the TopHat 1.2.0 result far closer into line with the results of STAR and HISAT2. All of the top 20 genes reported by TopHat 1.2.0 overlap with the top 100 reported by HISAT2 and there is a substantially better overlap with the STAR alignment results.

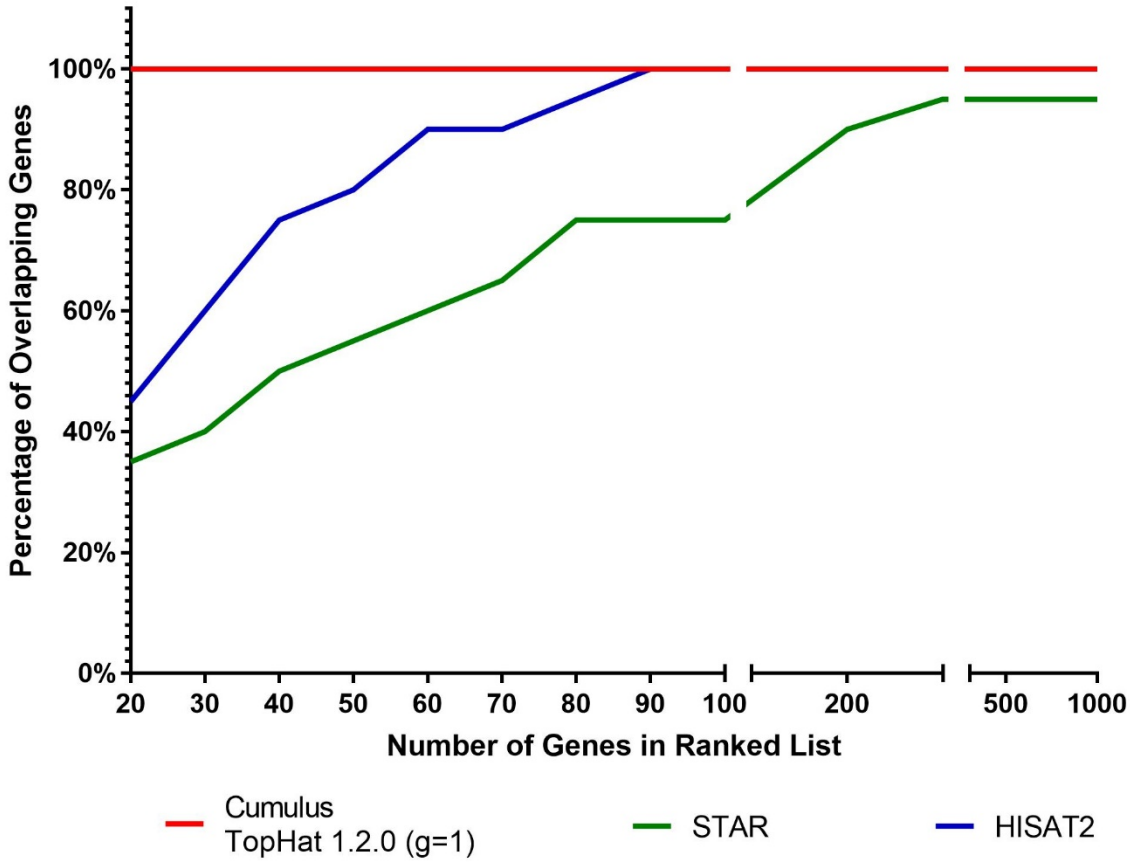

Figure 28 - The Overlap of Aligned Gene Lists Comparing a Corrected TopHat 1.2.0 Alignment to Other Common Aligners: A line graph showing the percentage overlap between the top 20 ESC genes from the Guttman et al. re-analysis aligned with TopHat 1.2.0 in Cumulus and an increasing number of the ranked gene lists from STAR aligner 2.5.2b and HISAT2 2.1.0. Multiple mapping of Genes has been disabled in TopHat 1.2.0 with the g=1 option.

Using the Cumulus system, a problem has been investigated in a published study. The workflow for this published study has been re-opened and corrected. The published re-analysis for this study is six years old at the time of writing and the tools do not easily function outside of the Cumulus environment. This demonstrates that the Cumulus system can be used to reproduce, compare with updated methods, and then re-analyse the results of a published study. Where a discrepancy occurs, the same study analysis environment can be altered to identify the cause of the discrepancy. Without the original analysis environment provided by the Cumulus system, it would be a technically complex process to identify and then validate problems such as this.

6.9 Summary of the RNA-seq Analysis Reproduction

The GeneProf system was initially published with the re-analysis of two example studies reported in Guttman et al. (2010). Here it has been shown that this re-analysis cannot easily be reproduced using available tools. Firstly, one of the applications used by GeneProf—the

TopHat 1.2 aligner—can no longer easily be compiled and run using the GeneProf install instructions. Using an alternative aligner in place of TopHat—such as STAR or HISAT2—gives a different result. By capturing this analysis pipeline in Cumulus, the tools used to analyse the study can still be run enabling the analysis to be reproduced to a far greater degree.

7 Integration of a New Analysis Workflow

Chapter 6 of this thesis demonstrated the capacity of the Cumulus system to improve the ability of a researcher to reproduce an existing RNA-seq study. In addition to the reproduction of studies, the Cumulus system also enables the development of new reproducible analysis workflows. New analysis tools and methods are constantly being developed and in order for an analysis system such as Cumulus to continue to be relevant, it is also necessary for it to be able to capture these newly developed methods. To evaluate the process of development of a novel workflow within the Cumulus system, this chapter will investigate a pertinent biological question which requires the development of an additional analysis workflow. Using the requirements of this data, a process of identification, development and then capture of a novel analysis workflow using the Cumulus system will be detailed.

7.1 Investigation of the Pluripotent State

During early mammalian development, the embryo develops from a ball of cells into an organism made up of trillions of cells and hundreds of different specialised cell types. This developmental process is possible because the cells which comprise the early embryo are pluripotent. A pluripotent cell is a cell which has the ability to differentiate into any cell type that occurs within an organism. It has been shown that the pluripotent state of an embryonic stem cell (a pluripotent cell type derived from early embryos that can self-renew in vitro) is maintained and regulated through the action of a few key transcription factors, including the protein Nanog (Chambers et al., 2003).

Mullin et al. (2008) demonstrated that dimerization of the pluripotency regulator Nanog is essential for its interaction with other binding partners. This dimerization is associated with the tryptophan repeat region, a part of the Nanog protein in which every fifth residue is a tryptophan. Deletion of this domain, essential for the formation of dimers, resulted in the abolition of cytokine-independent self-renewal; one of the defining abilities of embryonic stem cells. In a further study, the authors showed this dimerization of Nanog to be essential for its interaction with another transcription factor, Sox2, and consequently the orchestration of embryonic stem cell self-renewal by the interactions of the Nanog/Sox2 complex on the genome (Gagliardi et al., 2013).

In order to dissect a complex process such as transcriptional regulation by Nanog at a genome scale, it is important to be able to understand all of the components of the biological

machinery involved. Different techniques can provide information on different parts of the whole system. Proteomic analysis is the large-scale study of proteins (Anderson and Anderson, 1998). As shown in **Figure 29**, proteomic analysis can be used to identify the binding partners of a specific protein **Figure 29**, (**B1**). ChIP-Seq analysis can be used to identify neighbouring locations on the DNA where a protein and its partners bind **Figure 29**, (**B2**). RNA-seq analysis can be used to identify the downstream transcriptional effects of this binding to the DNA **Figure 29**, (**B3**). Application of all of these techniques in combination allows the system and its components to be mapped out in detail. This type of combinational analysis is referred to as a multi-omics analysis.

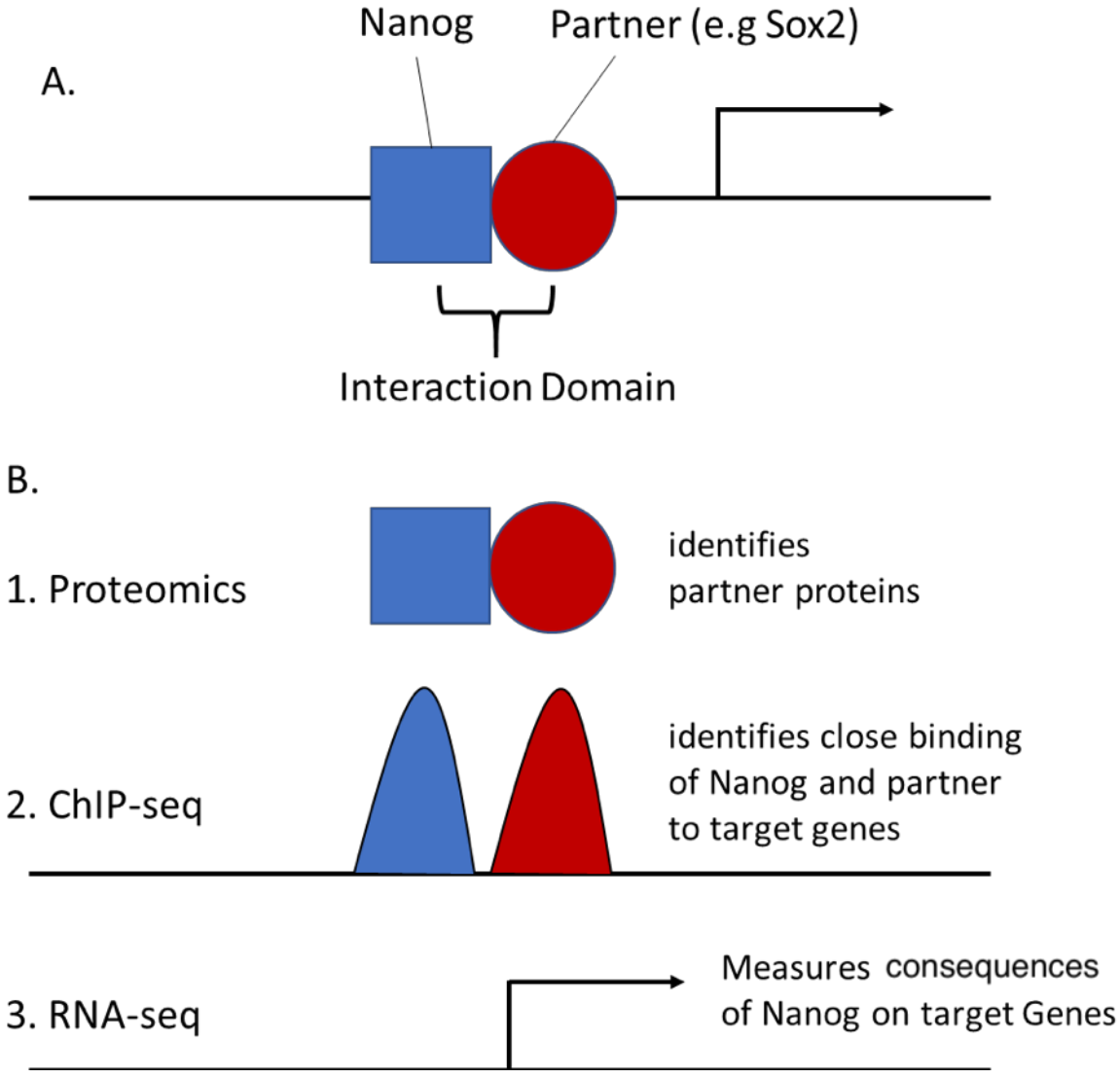

Figure 29 – A Model of an Integrative Multi-Omic Study. A diagram showing (**A**) The model of translational regulation by Nanog through binding to partner proteins such as Sox2 then binding to DNA influencing downstream regulation. (**B**) The different data types which enable the dissection of the system in A. (**1**) Proteomics data to identify the partner proteins with which Nanog interacts. (**2**) ChIP-seq data to identify locations on the DNA at which these partner proteins closely interact with the DNA. (**3**) RNA-seq data to measure the consequences on transcription of the genes activated by the interaction of partner proteins with DNA. Blue identifies the Nanog protein and its related DNA binding peak. Red represents a Nanog partner protein such as Sox2 and its DNA binding peak.

The GeneProf system already contains workflows for the analysis of ChIP-seq and RNA-seq data. However, it does not contain any facility for the analysis of Proteomic data. To facilitate the type of multi-omic analysis shown in **Figure 29**, a new proteomic analysis workflow will be created in the Cumulus system. This workflow and its tools will be made available for analysis through GeneProf to create a reproducible multi-omic analysis system.

7.2 Creation of a New Proteomics Analysis Workflow

High-throughput proteomics has become synonymous with the use of Mass Spectrometry (MS) to identify *en-masse* all of the proteins within a sample (Blackstock and Weir, 1999). In order to evaluate the process of creating a new analysis pipeline within the Cumulus system, a new workflow based around the analysis of high-throughput mass spectrometry has been implemented. There exists a well-recognised process for the analysis of proteomic data, described by projects such as the Trans Proteomic Pipeline (Keller and Shteynberg, 2011, http://tools.proteomecenter.org/wiki/index.php?title=Software:TPP). This process is shown in **Figure 30**. Data from one of a range of commercial, closed source mass spectrometers is converted into a standard format accepted by peptide and protein identification tools. These identification tools then identify the peptides and then proteins from a sample. This list of identified proteins can be further interrogated to provide insight onto the function of a biological system.

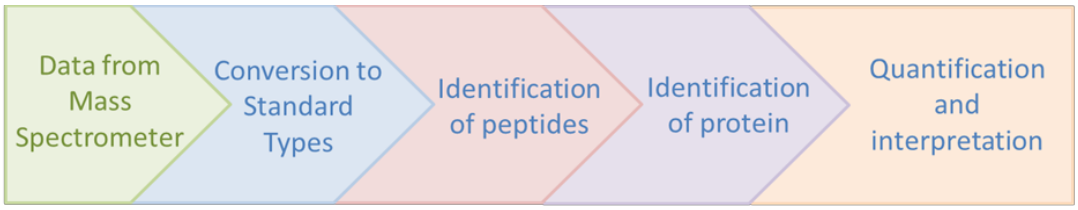

Figure 30 - Proteomic Analysis Process: The stages of the process for analysis of proteomic mass spectrometry data used by analysis pipelines such as the Trans Proteomic Pipeline (Keller and Shteynberg, 2011), a pipeline for analysis of high throughput mass spectrometry proteomic data.

In recent years, stand-alone 'simplified' proteomic pipeline tools have been developed such as SearchGUI (Vaudel et al., 2012). SearchGUI provides a graphical user interface to several protein identification packages. In addition, it can interface with proteomic data type conversion tools if they are already installed. A graphical user interface of the type provided by SearchGUI can simplify the process of analysis for the user as it provides 'point and click' options for running an analysis.

Whilst simplifying the process of analysis, GUI tools are unsuited for inclusion into a larger workflow system. In order to integrate a tool into a reproducible workflow system, the workflow system has to be able to communicate with the tool. Workflow systems communicate with tools through an API rather than a graphical user interface. SearchGUI provides SearchCLI, a command line interface (CLI) for this purpose. This CLI requires custom configuration files for each tool, mirroring the interface to the tools themselves. For a workflow system, this offers little benefit over interacting with the tools directly.

7.2.1 Proteomic Analysis Tools

For each stage of this analysis structure, a range of tools has been identified from literature and bioinformatic resource databases. A sample of these tools that are possibly suitable for inclusion within a pipeline is shown in **Table 10**, categorised by the analysis stage.

Analysis Stage / Type	Package
Conversion tools	ms2mz (http://www.bioproximity.com.s3-website-us-east-1.amazonaws.com/utility-for-converting-mass-spectrometer-file-formats), TPP Tools: MzXML2Search (Keller and Shteynberg, 2011), ProteoWizard Tools: msConvert (Kessner et al., 2008)
Peptide Identification	SQID (Li et al., 2011), PeptideProphet (Ma et al., 2012), MSPepSearch (https://chemdata.nist.gov/dokuwiki/doku.php?id=peptidew:mspepsearch)
Protein Identification	ProteinProphet (Nesvizhskii et al., 2003)
Protein & Peptide Identification	X! Tandem (Craig and Beavis, 2004) / X!!Tandem (Bjornson et al., 2008), X! P3 (Craig et al., 2005), X! Hunter (Craig et al., 2006), Sequest (Eng et al., 1994), Mascot (Perkins et al., 1999), OMSSA (Geer et al., 2004), Sherpa (Taylor et al., 1996), RAId (Alves and Yu, 2005), SpectraST (Lam et al., 2007), Skyline (MacLean et al., 2010), ISIS (Kangas et al., 2012),

	MyriMatch (Tabb et al., 2007), InsPecT (Tanner et al., 2005), SIMS (Liu et al., 2008), MassWiz (Yadav et al., 2011)
Quantification	XPRESS (Han et al., 2001), ASAPRatio (Li et al., 2003), Libra (Pedrioli, 2010)

Table 10 - Proteomic Tools: A table listing proteomic tools identified from literature and bioinformatic resource databases categorised by the stage of the proteomic workflow in which they may be used.

The majority of conversion tools listed in **Table 10** convert between two specified file formats, or from a single manufacturer format to a pipeline-specific format. The conversion tool which supports the most file formats is msConvert, part of the ProteoWizard package. ProteoWizard is a toolbox-style application which provides functionality for conversion between several different proteomic data formats. Manufacturers of different instruments have multiple different formats for their output data, often with restrictive licenses. Most manufacturers provide closed source software tools which enable conversion of these formats to less restrictive ones, as detailed in **Table 11**. The tools provided for this, however, are often binary files with limited operating system compatibility. The difficulty of conversion between proteomic data formats is well recognised (Holman et al., 2014). ProteoWizard includes msConvert, a tool which packages together a number of these manufacturer provided converters into a single tool. This allows a single software interface to convert between a large number of different formats, a feature no other software tool currently provides. The unique nature of the ProteoWizard tool has led to it being used in several popular protein analysis workflow tools. Examples of these are OpenMS (Röst et al., 2016), Computational Proteomics Analysis System (CPAS: Rauch et al., 2006) and the Trans-Proteomic Pipeline (Keller and Shteynberg, 2011).

As ProteoWizard still relies on the instrument manufacturer's library to do the underlying conversion, this limits the compatibility of ProteoWizard to those operating systems and hardware platforms supported by the manufacturers. These system limitations can often be based on legacy hardware systems leading to severe limitations on the systems on which ProteoWizard will run. **Table 11** details formats supported by ProteoWizard with the manufacturers and operating system compatibility.

Format	Manufacturer	License	Operating System Compatibility
mzML 1.1 mzML 1.0 mzXML	Proteomics Standards Initiative (Deutsch, 2008)	Open Source	Unix & Microsoft Windows
MGF (Mascot Generic Files)	Matrix Science	None	Unix & Microsoft Windows
MS2/CMS2/BMS2	The Scripps Research Institute (McDonald et al., 2004)	Open Source	Unix & Microsoft Windows
mzIdentML	Proteomics Standards Initiative (Jones et al., 2012)	Open Source	Unix & Microsoft Windows
MassHunter.d	Agilent	Closed	Microsoft Windows
WIFF	Applied Biosystems	Closed	Microsoft Windows
Thermo RAW	Thermo Fisher	Closed	Microsoft Windows
MassLynx RAW	Waters	Closed	Microsoft Windows
Compass.d, FID/YEP/BAF	Bruker	Closed	Microsoft Windows
T2D	ABI/Sciex	Closed	32-bit Microsoft Windows

Table 11 – Manufacturers and Proteomic File Formats Supported by ProteoWizard: A table showing the file formats supported by the ProteoWizard conversion tools. Each file format is listed with its license type, the operating system required for ProteoWizard to be able to read it and the Manufacturer or organisation responsible for its creation and upkeep.

ProteoWizard and msConvert are primarily Microsoft Windows based applications: the procedure for running it in Linux is very complex and requires a high degree of technical knowledge (http://tools.proteomecenter.org/wiki/index.php?title=Msconvert_Wine). In addition to this operating system limitation, for the software to work on Microsoft Windows additional packages are required. As is shown in **Table 12**, all of these requirements are now end of life and not generally supported by Microsoft.

Requirement	End of life (mainstream) date
Microsoft Visual C++ 2008 SP1 Redistributable Package	4/9/2013
Redistributable Packages for Visual Studio 2013	10/11/2016
Microsoft .NET Framework 4	1/12/2016
Microsoft .NET Framework 3.5 SP1	7/12/2011*

Table 12 - The ProteoWizard Software Dependencies and their End Of Life (EOL) Support Dates – A table showing the software dependencies for the Windows installation of the ProteoWizard package. For each dependency, the end of life support date is shown (sourced from https://support.microsoft.com/en-us/lifecycle). ***** Version 3.5 SP1 is not generally supported after the date shown unless shipped with an operating system which is still in support.

These requirements are defined against very specific versions and it can be difficult to correctly install and configure a Windows system to use these tools. Unsupported software no longer receives security updates. It is common for end of life software to have unfixed security flaws and running it exposes the computer to security threats. These security problems can be avoided through isolation of the system from the internet or other networks or virtualisation, but this can be technically complex. The limited operating system support for this tool and the complexity of installing and configuring it substantially increases the time and technical knowledge required to use ProteoWizard and msConvert.

Cumulus has the ability to run a variety of operating systems in a single workflow and enables the researcher to access operating systems they may not have access to locally. A Windows virtual machine has been configured with ProteoWizard and its required dependencies pre-installed. A pre-installed image enables users to start a virtual machine and very rapidly get access to the required tools thus bypassing the complex installation process. In addition, running the software in Cumulus' virtualisation isolates the insecure, end of life ProteoWizard software requirements, removing the associated security problems. This ProteoWizard virtual machine then forms a component which can then be chained together with other virtual machines to build up a pipeline. Each of the tools within these virtual machines may have their own complexities similarly to ProteoWizard but these are isolated from each other and the user.

ProteoWizard demonstrates a number of practical restrictions on software use including support for older software and for commercial closed source formats. In addition, it is difficult to incorporate non-Unix software into academic pipelines that are often run on Unix based systems. Packaging this within Cumulus shows how Cumulus can remove the complexity of requiring different analysis environments to enable researchers to analyse data or reproduce a scientific study.

From the identification category of software, Mascot, X! Tandem and OMSSA have been recognised as strongly performing algorithms (Kapp et al, 2005). Mascot is, however, released under a proprietary licence which is not permitted for inclusion in open source software. Support for OMSSA by the NCBI has since been discontinued making it unsuitable for a stable analysis pipeline. For these reasons, X! Tandem has been used as the protein identification tool.

The protein and peptide lookup stages of this process are carried out using X! Tandem through an R script. In recent years, a number of proteomic tools have been wrapped with R as part of the Bioconductor, a notable example being RforProteomics (Gatto & Christoforou, 2014). X! Tandem has been included in RforProteomics as rTandem (Fournier et al., 2014), a package wrapping the functionality of X! Tandem in R.

rTandem has been used to create an R identification script which is available in **Section 12.2 Appendix B: rTandem Analysis Script**. This script was included as a code block and snippet within CodeLab. The code block was used to process the proteomic data set. The rTandem module runs within Cumulus on a Linux based virtual machine as there is increased complexity with running rTandem on a Windows based machine. Cumulus runs each stage of the workflow on a different virtual machine, so, once the workflow is initially set-up, execution of this multi-operating system set-up is no more complex. Without this type of system, there would be increased complexity of configuring a Windows analysis environment with rTandem or configuring two different operating systems.

7.2.2 Integration of the Workflow with External Systems

Once packaged up in the Cumulus system, analysis workflows and tools can be used repeatedly in reproducible analyses, directly through Cumulus or workflow systems using the Cumulus API. Additionally, the virtual machines and commands used to analyse a study can be downloaded and used separately to the Cumulus system. **Figure 31** shows the newly-developed proteomic workflow tools available in the GeneProf tool palette. Any external workflow system can use the libraries packaged in the Cumulus system through its API in this way. Through this mechanism, proteomic analysis can be made available and integrated together with ChIP-seq, RNA-seq and other types of data within GeneProf.

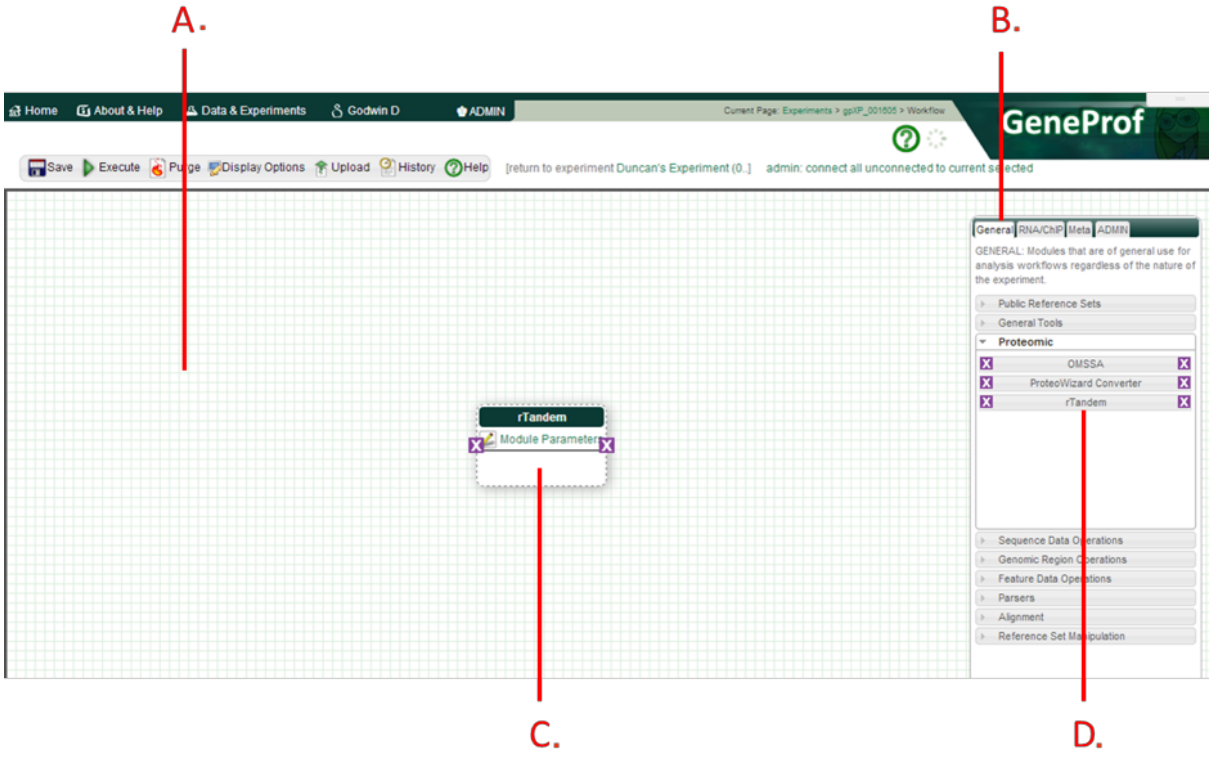

Figure 31 - Proteomics Tools Integrated in to the GeneProf Tool Palette: A screenshot of the GeneProf workflow system showing (**A**) The GeneProf workflow designer canvas. (**B**) The GeneProf workflow tool palette. (**C**) The rTandem tool available as a modular workflow component. (**D**) Additional proteomic tools in the tool palette populated from the Cumulus tool database.

Studies published using this proteomic analysis workflow can then link back to the Cumulus workflow, allowing the environment, its virtual machines and data to be used by the researcher reproducing the study. Sharing the workflow in this way communicates all of the data required to reproduce an analysis and reduces the burden on the reporting researcher identified in **Chapter 1**.

7.3 Proteomic Data Analysis

Section 7.2 of this thesis detailed the development of a new workflow within the Cumulus system. As discussed in **Section 7.1**, the novel pipeline was created to analyse high-throughput proteomic data in order to identify protein level interactions of Nanog in pluripotent cell populations. The following section details the use of this pipeline to investigate the action of dimerized Nanog using a novel proteomic dataset.

7.3.1 Analysis of Dimerized Nanog Binding Partners

In order to investigate the action of Nanog dimerization, a proteomic dataset was provided by Nick Mullin (MRC Centre for Regenerative Medicine, Edinburgh - personal communication) for analysis by the Cumulus proteomic analysis pipeline. Nuclear extracts

were prepared from mouse (*mus musculus*) wildtype ES cells and from mutant cells with a deletion of the tryptophan repeat region. These nuclear extracts were then used for FLAG-affinity purifications as previously described by van den Berg et al., (2010). This affinity purification captures proteins which are interacting with the Nanog protein. Mass spectrometry was then carried out on these purified protein fractions producing a raw data output.

The Cumulus proteomic analysis pipeline discussed in **Section 7.2.1**, was then used to analyse the raw mass spectrometry output. The proteomic pipeline was run from the workflow section of the Cumulus system against the two raw files. Two virtual machines were used in this particular analysis. The analysis report for these two virtual machines, showing installed software and versions are show in the appendices. The disk images for this analysis are stored in the common Cumulus virtual disk library.

7.3.2 Proteomic Analysis Methods

For the proteomic data analysis, the first stage of the Cumulus pipeline used a Windows XP-based virtual machine to convert the raw files using MSConvert from ProteoWizard 3.0.4416 32 bit. MSConvert was run with the default settings to convert the files into Mascot generic format (MGF) files. MGF files are a widely-used file format accepted by an array of identification tools. These MGF data files were saved to the Stembio Drive and passed into the second stage of the workflow.

The second stage of the workflow identified proteins using rTandem 1.16.0. It assigned reads by mapping against a protein database; UniProt 2017_09 *mus musculus,* reviewed mouse proteins (ftp://ftp.uniprot.org/pub/databases/uniprot/previous_releases/release-2017_09/; Wu et al., 2006). This stage of the workflow ran on a Ubuntu Linux 14.04-based virtual machine. The R script used to load and process the data for this stage is shown in **12.2 Appendix B: rTandem Analysis Script**. From the rTandem output, all proteins with an expect value (E-value) of less than 0.05 were regarded as identified and were selected. The E-value is a statistical confidence (Expectation Value) for the individual spectrum-to-sequence assignment (Brosch et al., 2008). The resultant proteins were then annotated using BiomaRt 2.32.1 and the mutant sample results subtracted from the wildtype proteins to produce the interactome. All tools used are listed with their versions and any specified parameters in **Table 13**.

Tool	Version	Specified Parameters
ProteoWizard	3.0.4416 32-bit	
rTandem*	1.16.0	log.expect=-1.3 min.peptides=2
BiomaRt	2.32.1	

Table 13 - Proteomic Analysis Tools and Parameters: A table showing the tools and versions used in the evaluation section. Any parameters specified beyond the defaults are listed in the 'Specified Parameters' column. ***** For additional information see the script in appendices.

7.3.3 Proteomic Analysis Results

Some 4919 and 4842 proteins were identified in the wildtype sample and the tryptophan-mutant sample respectively. The proteins found in the tryptophan mutant were subtracted from the list of those found in the wildtype leaving 1262 proteins which were only found in the wildtype. From this list, those corresponding to transcription factors were selected, leaving a list of 66 proteins. This list of 66 proteins is shown in **Appendix A: A Novel Interactome of Nanog, Table 15**. This protein list shows that Sox2 is one of the proteins which appears in the sample from the wildtype but not in the tryptophan mutant sample.

The interaction of Sox2 with Nanog has previously been shown to be dependent on the presence of the Nanog tryptophan repeat sequences (Gagliardi et al., 2013). In this experiment Sox2 was used as the positive control. The presence of Sox2 on this protein list shows that a protein previously shown to interact with the tryptophan repeat sequences of Nanog can successfully identified. This suggests that the other candidates identified may be novel tryptophan repeat dependent Nanog interacting partners. Earlier studies have identified interactomes of general binding partners of Nanog. However, this is the first time that a proteomic study using the Nanog tryptophan mutant has been used to identify an interactome reliant upon the tryptophan repeat region. The targets generated from this analysis are novel and will thus require further experimental validation.

7.4 Summary of the New Analysis Workflow

The GeneProf system provides analysis workflows to investigate RNA and binding to DNA. In order to investigate protein level interactions such as those identified in **Section 7.1** a new proteomic analysis workflow was required. This chapter has covered the development of this workflow, its integration into the Cumulus and GeneProf systems and its use in a pilot study to analyse a novel proteomic dataset.

The Cumulus proteomic analysis pipeline is a new workflow for the analysis of mass spectrometry data implemented in the Cumulus system. It demonstrates how a multiple stage, reproducible analysis running across two different operating systems can be managed and run by the Cumulus system. Analysis tools which are complex to install and configure can be packaged in a virtual disk and included in the analysis tool library. Each tool may have a different complex chain of dependencies and packaging them in this way isolates this complexity from the other tools and hides it from the user. Whilst in this example two environments have been linked in a workflow, Cumulus allows the linking of any number of virtual machines. This enables combinations of environments which would normally be complex to build and maintain to be assembled relatively simply.

In order to run this workflow without the structure provided by the Cumulus system, a scientist would have to install and configure software on two different operating systems, Windows and Linux. Installing ProteoWizard on Windows requires the user to either run an old, insecure and no longer widely available operating system or to configure a modern operating system with a complex set of out-of-date packages. Neither of these options is simple and Cumulus removes this complexity from the researcher reproducing the study by providing the pre-installed running system with the analysed study. As identified in **Chapter 1**, the kind of complexity in installing and using software demonstrated here is a major hindrance in the reproducibility of a study and in removing this hindrance, the Cumulus system enables reproducible research.

Using this analysis pipeline, we have analysed a proteomic study and identified an interactome apparently dependent upon the tryptophan repeat region of Nanog. As part of this analysis we find Sox2, which is the best-known interaction partner of Nanog (Gagliardi et al., 2013). Sox2 is known to interact with Nanog through the tryptophan repeat. Novel candidate proteins can be identified from the pilot study and could be directly validated in further *in vitro* experiments. This pilot study contains only one replicate and future work will increase the level of replication. Proteomics and protein interaction data can be highly variable (Piehowski et al., 2013) and now that the general approach has been validated, further work is underway to increase the number of replicates in this proteomic data set.

8 System Usage and Availability

The Cumulus system consists of a number of interconnecting components, each of which can be run independently or as part of the larger system. For example, Cumulus can provision virtual machines and connect them to a central Stembio Drive once they have become available. Alternatively, it can connect to a folder on the host operating system as Cumulus can be installed with or without the Stembio Drive. Using a folder on the host operating system simplifies installation, but then means the user cannot manage their files from other applications or a web interface. Installing the system as a whole provides greater functionality but is not a requirement. Each component is distributed as a Web Application Resource (WAR) file with a configuration file.

The WAR files provided can be run in any Java servlet container, they have been tested with Apache Tomcat 7. All of the applications require an SQL based database supported by the Hibernate framework. They have been tested with MySQL version 5.7 and MariaDB version 10.1. The authentication and security components support connection to a range of Single Sign On (SSO) frameworks through Spring Security. It has been tested with both the central authentication service protocol and individual users specified in a configuration file.

The components of the system are available under the Apache Software License 2 or the Affero General Public License as indicated in **Table 14**. Binaries and source code will be available at http://github.com/stembio after the publication of this thesis.

Component	License	Requirements
Cumulus	Creative Commons Attribution-Non-Commercial 3.0	VirtualBox, Java, SQL database
Stembio Drive	Affero General Public License (AGPL)	Java, SQL database
CodeLab	Apache Software License (ASL) v2	Cumulus, Java, SQL database
Stembio Visualisation	Apache Software License (ASL) v2	Java, SQL database

Table 14 – System Components, Licenses and Requirements: A table showing the components of the software developed in this thesis, the software license under which they are distributed and their application requirements.

9 Discussion

Reproducibility of scientific studies is a central tenet of science and yet in practice published genomic studies often cannot be reproduced. The field has been described as being in crisis (Baker, 2016). The inclusion of a computational analysis within a study can further increase the complexity of reproducing that study. Even when provided with the source data and a list of the tools used, it can be difficult for a researcher to reproduce the same results as described in the original paper. It is not enough for a publication to simply name the tools used in an analysis (Duck et al., 2015). Differences between different tools, versions, configurations, dependencies, operating systems and hardware all potentially impact on the ability to reproduce the analysis results from a study.

A published study provides a snapshot of the analysis fixed in time and using a specific set of tools and techniques which represent the state of the art at the time of publication. When published, even if the analysis tools used in a study represent the best current analysis approach, they can rapidly be superseded by advancements in the field. Once this happens, it can lead to the version of the software used for the publication no longer being actively developed or supported. Software tools which are no longer developed or supported are regarded as legacy and are often brittle and highly susceptible to breaking changes in software or hardware on which they depend (Bisbal et al., 1999). As has been shown in **Section 6.4**, it is common to try to reproduce an analysis sometime after a paper is published and to find that the tools used are unavailable or broken. There are currently no systems which enable the complete reproduction of a historical analysis despite extensive community efforts. The initial question posed as a result of these problem is, how can we use and extend existing tools and systems in order to incorporate the ability to fully record and share an analysis long into the future?

The work presented here builds on the state of the art, high throughput workflow and virtualisation tools to produce a software system which addresses the issues of reproducibility. **Sections 2 and 3** of this thesis detail the design, development and implementation of this software. The resultant analysis system, Cumulus, captures the complex configuration of an analysis environment in a set of virtual machines. As shown in **Section 7.2.1** each analysis tool that forms part of a workflow can be run in a different virtual machine enabling incompatible components to be run together in the same workflow. Different operating systems and configurations which would otherwise be very complex for a researcher to set up and configure can be assembled with relative ease. These virtual machines can then be shared in collaboration with the data already available from the workflow system to allow the study to be re-run.

The Cumulus disk images use a standard format which is compatible with a wide range of other tools outside of the Cumulus system. The ability to access and use the analysis images outside of the Cumulus system is a major advantage in maintaining reproducibility. It allows a scientist attempting to reproduce the study to use the study materials without the need for any part of the Cumulus system, entirely avoiding virtualisation or indeed having any knowledge of the platform. Over time, new systems are developed and adopted replacing existing ones. All software has a lifespan and if the Cumulus platform becomes redundant or unavailable in the future, the data and studies it contains will still be accessible. This may not be as easy for other formats of virtual disk image which are dependent upon a specific software tool to read them.

In order to evaluate the reproducibility of a study captured by the Cumulus system against the state of the art, a legacy RNA-seq study has been recorded in a virtual machine and re-analysed by the Cumulus system. **Section 6.7** demonstrates that the Cumulus system enables the reproduction of this study where installation of the legacy tool set or reproduction with modern tools is, by comparison, technically complex or not practically achievable. During the reproduction of this RNA-seq data analysis study, it was identified that the original published result was likely to be affected by multiple mapping of the RNA-seq reads to pseudogenes. This multiple mapping was an unknown problem at the time of the original publication and likely to cause some discrepancy between the original study and the re-analysis using modern aligners. **Section 6.8** shows how Cumulus enables the study to be re-run with different settings, removing multiple mapping to pseudogenes, bringing the results closer to those of more modern aligners.

The Cumulus system maintains a database of analysis tools and configured virtual disk images which contain these tools. **Section 7.2** demonstrates the Cumulus system being used to build a novel automated workflow for the analysis of high-throughput proteomic data. One of the components of this novel workflow, ProteoWizard, is an example of a legacy tool which is complex to install and maintain. The Cumulus system wraps this component in a virtualised environment, hiding this complexity from the researcher and allowing it to be used in an automated, reproducible workflow. This demonstrates that using a tool which has been captured by the Cumulus system can reduce the complexity of using that tool.

A Cumulus workflow is defined as a selection of wrapped tools, each within its own virtual disk image. This allows each tool within a workflow to have a unique operating system and configuration entirely isolated from the other components. In order to run a workflow, Cumulus assembles and starts virtual machines for each disk image in the correct order for the workflow. Cumulus automatically selects and launches the correct virtual disk image for

a specific tool. Without a system such as Cumulus, the complexity of installing and configuring a large number of incompatible tools within a workflow can be arduous. The automation of the process of assembling the workflow combined with the reduced complexity of each individual tool allows different workflows to be rapidly built and tested. Widely-used workflow tools can be used multiple times to construct multiple different pipelines.

In addition to providing automation whilst assembling a new workflow, Cumulus allows the researcher to avoid manual processes involved in running an existing workflow. Provisioning virtual machines, copying the analysis data and running the tool is carried out automatically. This allows the researcher to start large numbers of analysis tools simultaneously and run the analysis at scale over many datasets.

The novel workflow detailed in **Section 7.2** has been used to analyse a pilot study of proteomic data. The results of this analysis define a novel interactome, apparently dependent upon a specific structural region of the Nanog protein.

9.1 Research Outputs

The initial research question of this work was answered with the examples presented in **Section 6**, which illustrate that the Cumulus system has the functionality to capture and reduce the complexity of reproducing bioinformatic studies. In this way, Cumulus incorporates the ability to fully record and share an analysis in a time-stable format.

The body of research presented in this thesis was successful in producing a number of novel research outputs. Firstly, the design and implementation of Cumulus; a novel software platform which enables the reproduction of scientific data analyses (see **Section 3**). Secondly, the design and construction of a bespoke proteomics pipeline, optimised specifically for the novel analysis of a newly generated proteomic data set (see **Section 7.2**). Thirdly, the use of this pipeline to discover a novel Nanog interactome, apparently dependent on its tryptophan repeat region (see **Section 7.3**). Finally, the extension of the Cumulus system with the design and implementation of a data visualisation framework, allowing the output of new interactive visualisation tools built for specific data sets to be shared to other researchers online (see **Section 5**). This visualisation framework has the ability to facilitate data exchange between research groups with an initial example published in the Journal of Experimental Medicine (McGarvey et al., 2017).

9.2 Developments During the Course of This Work

Since the completion of the Cumulus system, there have been further developments in a number of pertinent areas. The following section details some of these developments and their relevance to the work covered by this thesis.

Docker is a Unix based virtualisation technology discussed in **Section 1.7.2**. It was considered as a potential technology on which to base Cumulus at the start of the project but at that time was rejected. It has significantly matured since the creation of the Cumulus system and has been suggested as a suitable tool to aid reproducibility (Boettiger, 2015). Docker, and its ecosystem of tools is now more developed, robust and widely used (Fjukstad and Bongo, 2017). Several new, orchestrated, multiple-container environments have been developed for Containers such as Docker Swarm (Nguyen and Bein, 2017) and Kubernetes (Burns et al., 2016). These multiple-container environments could be used to enable parallelisation of virtualised reproducible workflows.

The growth in the use of Docker-based tools in bioinformatics has primarily been to assist with the distribution, deployment and execution of analysis tools (Fjukstad and Bongo, 2017). Projects such as BioContainers (da Veiga Leprevost et al., 2017) or BIDS apps (Gorgolewski et al., 2017) provide libraries of pre-configured Docker containers which can be downloaded and used by researchers. Others such as Dugong (Menegidio et al., 2018) use Docker as a platform on which to use various existing package management systems. The Reproducible Bioinformatics Project (Kulkarni et al., 2017) uses Docker combined with a standard schema to allow people to build their own containers for use in standardised workflows. These projects utilise Docker as a replacement for the various package management tools on different Unix distributions. Providing a package in the Docker container format allows the same release to be used across most Unix operating systems, rather than requiring multiple releases. For instance, in order to release software through package management tools which would work on most Linux operating systems, a release in both the RPM and deb format package would be required. A single Docker image release, however, could work on most Linux operating systems.

One of the main benefits of using a virtualisation system within Cumulus is that it provides a homogenised computational platform which will run an analysis in an identical manner on a range of different operating systems and hardware. Whilst containerisation can enable the more efficient distribution of scientific software than package managers, it does not provide the homogenised computational platform of virtualisation. Matelsky et al. (2018) noted that observed differences between results from the Pre-FreeSurfer pipeline (Gronenschild et al., 2012) caused by differences between operating systems, is not avoided through the use of

containerised environments. In addition to this, Docker does not support non-Unix based operating systems, either as the host operating system or within the containerised environment. Pipelines such as the one detailed in **Section 7**, which incorporate both Windows and Linux operating systems within the virtualised workflow, could not be supported. This limitation is also applicable to the platforms on which a researcher reproducing a study might want to use. Unless virtualisation is used, container-based workflows require a Unix-based computer to run them (Turnbull, 2014). Virtualisation systems, on the other hand, can be operating system agnostic.

Virtualisation support in newer computer hardware and recent improvements in virtualisation software have both made virtualisation a relatively lightweight addition to bare metal or containerisation systems (Barik et al., 2016; Morabito et al., 2015). Containerisation, however, retains the economic advantage of being able to sub-divide a single computer more efficiently than running multiple virtual machines (Kumar and Kurhekar, 2016). Whilst these improvements in efficiency are not a concern for reproducibility, they are important for the adoption of these technologies for large scale systems.

Kubernetes (Burns et al., 2016) is a software orchestration system based on container virtualisation systems, such as Docker, which automates the deployment and management of containerised applications. The design of Kubernetes is heavily influenced by Google's Borg system (Verma et al., 2015) and is designed to provide mechanisms for deploying, maintaining, and scaling applications in a robust manner. Kubernetes achieves this robustness through the use of a distributed structure and employing concepts such as auto scalers and replica sets. Auto scalers increase the available hardware resource when an application requires it, and a replica set replaces part of the system on failure. These concepts allow Kubernetes to maintain a running system despite failures (Poulton, 2017).

Kubernetes also has the concept of a Job. This is a piece of processing work which needs to be carried out. Multiples of these jobs can be connected to form a workflow which can then be orchestrated by the Kubernetes system. The Cumulus system, by comparison, has a much more simplistic orchestration structure. Virtual machines can be automatically run but user intervention is required in the case of failure. A future area of investigation could be to use an orchestration system such as Kubernetes in order to improve the robustness and computational throughput of Cumulus. The cost of using a containerisation platform such as Kubernetes would be the loss of cross-platform support provided by virtualisation.

Section 2.4 details the design of the Virtual Container API. Since the Virtual Container API has been written, several standard interfaces for connecting to a wide range of virtual machines have become available. These alternatives could now be exchanged with, or

interfaced by, the Virtual Container API. One example of such an interface is Vagrant (https://www.vagrantup.com, Hashimoto, 2013), another is Apache jclouds (https://jclouds.apache.org), both have the ability to interact with numerous clouds and virtualisation environments such as VirtualBox or XEN. These technologies could be used in future to broaden the different types of virtualisation supported by the Cumulus system.

At the start of this project, private cloud systems were very early in their development and with limited functionality. Since the creation of the Cumulus system, private cloud systems such as the FOSS OpenStack have significantly developed. OpenStack has become a robust private cloud system supporting a wide variety of uses. It is widely adopted and has become the most commonly used private cloud virtualisation framework (Elia et al., 2017). Even now, the installation and configuration of private clouds such as OpenStack can be highly challenging, requiring the installation of several complex enterprise components (Pepple, 2011). A lot of this complexity has however been reduced by automation tools such as Ansible (https://www.ansible.com), Chef (http://www.chef.io) and Puppet (http://www.puppet.com).

OpenStack provides Mistral, a basic workflow service which enables the specification and execution of jobs on virtual machines within OpenStack. Cumulus maintains a database of bioinformatic analysis tools and their dependencies, linked to their virtual disk images. This kind of information is not captured by Mistral and, in future work, the systems could be combined. The Cumulus system could also adopt OpenStack as the virtualisation and orchestration engine for its workflows to provide far greater scalability to the system. The Cumulus system is an investigation into producing a reproducible system rather than a large enterprise system. Should Cumulus need to support a high volume of users in the future, swapping the virtualisation component to OpenStack would be a good way to support this.

9.3 Additional Further Work

The initial further work from this thesis will be a full-scale analysis of the pilot study of Nanog interactomes found in **Section 7.3**. As more replicate data becomes available this will be incorporated to strengthen the analysis outcomes. In addition, the proteins in the identified interactome will be further validated individually.

One area which is not currently directly addressed by the Cumulus system is that of random seeds (discussed in **Section 1.5.5**). Analysis methods often contain heuristics that use a pseudorandom seed element to generate an initial state. Running an analysis with different seed values will generate different results, so to generate exactly the same result requires the same seed. The results generated using different seeds from the same pipeline, whilst

different, are equally valid. Currently the Cumulus system expects a researcher to specify and communicate when random seeds are used by a software tool. A potential method to standardise this process could be to override the operating system's random source inside the virtual machine to a known value. This would allow the process of capturing and communicating the random seed to be automated.

10 References

Achinstein, P. (2004). Science rules: a historical introduction to scientific methods (Johns Hopkins University Press). ISBN: 978-0-8018-7944-9

Adair, R.J. (1966). A Virtual Machine System for the 360/40 (International Business Machines Corporation, Cambridge Scientific Center).

Afgan, E., Baker, D., van den Beek, M., Blankenberg, D., Bouvier, D., Čech, M., Chilton, J., Clements, D., Coraor, N., Eberhard, C., et al. (2016). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. Nucleic Acids Res. *44*, W3–W10. https://doi.org/10.1093/nar/gkw343

Alves, G., and Yu, Y.-K. (2005). Robust accurate identification of peptides (RAId): deciphering MS2 data using a structured library search with de novo based statistics. Bioinformatics *21*, 3726–3732. https://doi.org/10.1093/bioinformatics/bti620

Anderson, N.L., and Anderson, N.G. (1998). Proteome and proteomics: new technologies, new concepts, and new words. Electrophoresis *19*, 1853–1861. https://doi.org/10.1002/elps.1150191103

Andrei, A. (2017). Why Oracle Java 7 And 6 Installers No Longer Work. http://www.webupd8.org/2017/06/why-oracle-java-7-and-6-installers-no.html (Accessed August 2018)

Bacon, F. (1878). Novum organum, translated by Jazzybee Verlag, 2010. (Createspace). ISBN: 9781440042508

Baggerly, K.A., and Coombes, K.R. (2009). Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology. The Annals of Applied Statistics *3*, 1309–1334. https://doi.org/10.1214/09-AOAS291

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. Nature *533*, 452–454. https://doi.org/10.1038/533452a

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. ACM SIGOPS operating systems review. Vol. 37, No. 5, p. 164. https://doi.org/10.1145/945445.945462

Barik, R.K., Lenka, R.K., Rao, K.R., and Ghose, D. (2016). Performance analysis of virtual machines and containers in cloud computing. Computing, Communication and Automation

(ICCCA), 2016 International Conference on, IEEE, pp. 1204–1210. https://doi.org/10.1109/CCAA.2016.7813925

Barrett, D.J., and Silverman, R.E. (2001). SSH, the secure shell: the definitive guide (O'Reilly Media). ISBN: 978-0-596-00011-0

Begley, C.G., and Ellis, L.M. (2012). Drug development: Raise standards for preclinical cancer research. Nature *483*, 531–533. https://doi.org/10.1038/483531a

Bellard, F. (2005). QEMU, a fast and portable dynamic translator. In USENIX Annual Technical Conference, FREENIX Track, pp. 41–46.

Belmann, P., Dröge, J., Bremges, A., McHardy, A.C., Sczyrba, A., and Barton, M.D. (2015). Bioboxes: standardised containers for interchangeable bioinformatics software. Gigascience *4*, 47. https://doi.org/10.1186/s13742-015-0087-0

Ben-Yehuda, M., Day, M.D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., and Yassour, B.-A. (2010). The Turtles Project: Design and Implementation of Nested Virtualization. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, (USENIX Association), pp. 423–436.

van den Berg, D.L.C., Snoek, T., Mullin, N.P., Yates, A., Bezstarosti, K., Demmers, J., Chambers, I., and Poot, R.A. (2010). An Oct4-centered protein interaction network in embryonic stem cells. Cell Stem Cell *6*, 369–381. https://doi.org/10.1016/j.stem.2010.02.014

Berger, B., Peng, J., and Singh, M. (2013). Computational solutions for omics data. Nat. Rev. Genet. *14*, 333–346. https://doi.org/10.1038/nrg3433

Beserra, D., Moreno, E.D., Endo, P.T., Barreto, J., Sadok, D., and Fernandes, S. (2015). Performance Analysis of LXC for HPC Environments. Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference, IEEE, pp. 358–363. https://doi.org/10.1109/CISIS.2015.53

Bisbal, J., Lawless, D., Bing Wu, and Grimson, J. (1999). Legacy information systems: issues and directions. IEEE Software *16*, 103–111. https://doi.org/10.1109/52.795108

Bjornson, R.D., Carriero, N.J., Colangelo, C., Shifman, M., Cheung, K.-H., Miller, P.L., and Williams, K. (2008). X!!Tandem, an improved method for running X!tandem in parallel on collections of commodity computers. J. Proteome Res. 7, 293–299. https://doi.org/10.1021/pr0701198

Blackman, D. (2000). Debian Package Management, Part 1: A User's Guide. Linux J. *2000*.

Blackstock, W.P., and Weir, M.P. (1999). Proteomics: quantitative and physical mapping of cellular proteins. Trends Biotechnol. *17*, 121–127. https://doi.org/10.1016/S0167-7799(98)01245-1

Blankenberg, D., Von Kuster, G., Coraor, N., Ananda, G., Lazarus, R., Mangan, M., Nekrutenko, A., and Taylor, J. (2010). Galaxy: a web-based genome analysis tool for experimentalists. Curr Protoc Mol Biol *Chapter 19*, Unit 19.10.1-21. https://doi.org/10.1002/0471142727.mb1910s89

Blankenberg, D., Von Kuster, G., Bouvier, E., Baker, D., Afgan, E., Stoler, N., Galaxy Team, Taylor, J., and Nekrutenko, A. (2014). Dissemination of scientific software with Galaxy ToolShed. Genome Biol. *15*, 403. https://doi.org/10.1186/gb4161

Boettiger, C. (2015). An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review *49*, 71–79. https://doi.org/10.1145/2723872.2723882

Bolger, A.M., Lohse, M., and Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics *30*, 2114–2120. https://doi.org/10.1093/bioinformatics/btu170

Bossuyt, P.M., Reitsma, J.B., Bruns, D.E., Gatsonis, C.A., Glasziou, P.P., Irwig, L.M., Moher, D., Rennie, D., de Vet, H.C.W., Lijmer, J.G., et al. (2003). The STARD statement for reporting studies of diagnostic accuracy: explanation and elaboration. Ann. Intern. Med. *138*, W1-12. https://doi.org/10.7326/0003-4819-138-1-200301070-00012-w1

Brazma, A., Hingamp, P., Quackenbush, J., Sherlock, G., Spellman, P., Stoeckert, C., Aach, J., Ansorge, W., Ball, C.A., Causton, H.C., et al. (2001). Minimum information about a microarray experiment (MIAME)-toward standards for microarray data. Nat. Genet. *29*, 365–371. https://doi.org/10.1038/ng1201-365

Brosch, M., Swamy, S., Hubbard, T., and Choudhary, J. (2008). Comparison of Mascot and X!Tandem performance for low and high accuracy mass spectrometry and the development of an adjusted Mascot threshold. Mol. Cell Proteomics *7*, 962–970. https://doi.org/10.1074/mcp.M700293-MCP200

Brusic, V. (2007). The growth of bioinformatics. Brief. Bioinformatics *8*, 69–70. https://doi.org/10.1093/bib/bbm008

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue *14*, 70–93. https://doi.org/10.1145/2898442.2898444

Calvo, A. (2015). Looking Forward: Microsoft Support for Secure Shell (SSH). [Web log post]. Retrieved from https://blogs.msdn.microsoft.com/powershell/2015/06/03/looking-forward-microsoft-support-for-secure-shell-ssh (Accessed August 2017).

Carr, C.S. (1969). RFC 15: Network subsystem for time sharing hosts (1969). https://www.ietf.org/rfc/rfc15.txt (Accessed August 2017)

Celebi, M.E., Kingravi, H.A., and Vela, P.A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert Systems with Applications *40*, 200–210. https://doi.org/10.1016/j.eswa.2012.07.021

Chamberlin, D.D., and Boyce, R.F. (1974). SEQUEL: A Structured English Query Language. In Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control, ACM, pp. 249–264. https://doi.org/10.1145/800296.811515

Chambers, I., Colby, D., Robertson, M., Nichols, J., Lee, S., Tweedie, S., and Smith, A. (2003). Functional expression cloning of Nanog, a pluripotency sustaining factor in embryonic stem cells. Cell *113*, 643–655. https://doi.org/10.1016/S0092-8674(03)00392-1

Chen, X., Xu, H., Yuan, P., Fang, F., Huss, M., Vega, V.B., Wong, E., Orlov, Y.L., Zhang, W., Jiang, J., et al. (2008). Integration of external signaling pathways with the core transcriptional network in embryonic stem cells. Cell *133*, 1106–1117. https://doi.org/10.1016/j.cell.2008.04.043

Christiansen, T., Foy, B.D. and Wall, L. (2012). Programming Perl (O'Reilly Media). ISBN: 978-0-596-00492-7

Code share. (2014). Nature *514*, 536. https://doi.org/10.1038/514536a

Cohen-Boulakia, S., Belhajjame, K., Collin, O., Chopard, J., Froidevaux, C., Gaignard, A., Hinsen, K., Larmande, P., Bras, Y.L., Lemoine, F., et al. (2017). Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. Future Generation Computer Systems *75*, 284–298. https://doi.org/10.1016/j.future.2017.01.012

Combs, C.A. (2010). Fluorescence microscopy: a concise guide to current imaging methods. Curr Protoc Neurosci *Chapter 2*, Unit2.1. https://doi.org/10.1002/0471142301.ns0201s50

Conesa, A., Madrigal, P., Tarazona, S., Gomez-Cabrero, D., Cervera, A., McPherson, A., Szcześniak, M.W., Gaffney, D.J., Elo, L.L., Zhang, X., et al. (2016). A survey of best practices for RNA-seq data analysis. Genome Biol. *17*, 13. https://doi.org/10.1186/s13059-016-0881-8

Corden, M.J., and Kreitzer, D. (2012). Consistency of Floating-Point Results using the Intel® Compiler. Technical report, Intel Corporation, Software Solutions Group. https://software.intel.com/en-us/articles/consistency-of-floating-point-results-using-the-intel-compiler (Accessed August 2017).

Craig, R., and Beavis, R.C. (2004). TANDEM: matching proteins with tandem mass spectra. Bioinformatics *20*, 1466–1467. https://doi.org/10.1093/bioinformatics/bth092

Craig, R., Cortens, J.P., and Beavis, R.C. (2005). The use of proteotypic peptide libraries for protein identification. Rapid Commun. Mass Spectrom. 19, 1844–1850. https://doi.org/10.1002/rcm.1992

Craig, R., Cortens, J.C., Fenyo, D., and Beavis, R.C. (2006). Using annotated peptide mass spectrum libraries for protein identification. J. Proteome Res. 5, 1843–1849. https://doi.org/10.1021/pr0602085

Crane, D., and McCarthy, P. (2008). Comet and Reverse Ajax: the next-generation Ajax 2.0 (Apress). ISBN: 978-1-59059-998-3

Dash, P., and Blaminsky, J. (2013). Getting Started with Oracle VM VirtualBox. (Packt Publishing). ISBN: 978-1-78217-783-8

Davidson, R.L., Weber, R.J.M., Liu, H., Sharma-Oates, A., and Viant, M.R. (2016). Galaxy-M: a Galaxy workflow for processing and analyzing direct infusion and liquid chromatography mass spectrometry-based metabolomics data. Gigascience *5*, 10. https://doi.org/10.1186/s13742-016-0115-8

Denton, J. (2014). Learning OpenStack Networking (Neutron). (Packt Publishing). ISBN: 978-1-322-19802-6

Deutsch, E. (2008). mzML: a single, unifying data format for mass spectrometer output. Proteomics *8*, 2776–2777. https://doi.org/10.1002/pmic.200890049

Dobin, A., Davis, C.A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., and Gingeras, T.R. (2013). STAR: ultrafast universal RNA-seq aligner. Bioinformatics *29*, 15–21. https://doi.org/10.1093/bioinformatics/bts635

Dodge, Y. (2006). The Oxford dictionary of statistical terms (Oxford Univ. Press). ISBN: 978-0-19-920613-1

Droms, R. (2008). RFC 2131: Dynamic host configuration protocol, 1997. https://www.ietf.org/rfc/rfc2131.txt (Accessed August 2017)

Duck, G., Kovacevic, A., Robertson, D.L., Stevens, R., and Nenadic, G. (2015). Ambiguity and variability of database and software names in bioinformatics. J Biomed Semantics *6*, 29. https://doi.org/10.1186/s13326-015-0026-0

Duckett, J., Ruppert, G., and Moore, J. (2014). JavaScript & jQuery: interactive front-end web development (Wiley). ISBN: 978-1-118-87165-2

Durinck, S., Moreau, Y., Kasprzyk, A., Davis, S., De Moor, B., Brazma, A., and Huber, W. (2005). BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis. Bioinformatics *21*, 3439–3440. https://doi.org/10.1093/bioinformatics/bti525

Dusseault, L. (2004). WebDav: next generation collaborative Web authoring (Prentice Hall). ISBN: 978-0-13-065208-9

Edgar, R., Domrachev, M., and Lash, A.E. (2002). Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. Nucl. Acids Res. *30*, 207–210. https://doi.org/10.1093/nar/30.1.207

Elia, I.A., Antunes, N., Laranjeiro, N., and Vieira, M. (2017). An Analysis of OpenStack Vulnerabilities. 2017 13th European Dependable Computing Conference (EDCC). IEEE, pp. 129–134. https://doi.org/10.1109/EDCC.2017.29

Eng, J.K., McCormack, A.L., and Yates, J.R. (1994). An approach to correlate tandem mass spectral data of peptides with amino acid sequences in a protein database. J. Am. Soc. Mass Spectrom. *5*, 976–989. https://doi.org/10.1016/1044-0305(94)80016-2

Festuccia, N., Osorno, R., Halbritter, F., Karwacki-Neisius, V., Navarro, P., Colby, D., Wong, F., Yates, A., Tomlinson, S.R., and Chambers, I. (2012). Esrrb is a direct Nanog target gene that can substitute for Nanog function in pluripotent cells. Cell Stem Cell *11*, 477–490. https://doi.org/10.1016/j.stem.2012.08.002

Fette, I., and Melnikov, A. (2011). RFC 6455: The WebSocket Protocol. IETF, Network Working Group, 2011. https://www.ietf.org/rfc/rfc6455.txt (Accessed August 2017).

Fielding, R.T., and Taylor, R.N. (2000). Principled Design of the Modern Web Architecture. In Proceedings of the 22nd International Conference on Software Engineering, ACM, pp. 407–416. https://doi.org/10.1145/337180.337228

Fisher, R. (1974). The design of experiments, Second edition (Oliver and Boyd). ISBN: 978-0-02-844690-5

Fisher, P.T., and Murphy, B.D. (2010). Spring persistence with Hibernate (Apress). ISBN: 978-1-4302-2633-8

Fjukstad, B., and Bongo, L.A. (2017). A Review of Scalable Bioinformatics Pipelines. Data Science and Engineering *2*, 245–251. https://doi.org/10.1007/s41019-017-0047-z

Flanagan, D., and Matsumoto, Y. (2008). The Ruby programming language (O'Reilly Media). ISNB: 978-0-596-51617-8

Foster-Johnson, E. (2003). Red Hat RPM Guide (Red Hat). ISBN: 978-0764549656

Fournier, F., Joly Beauparlant, C., Paradis, R., and Droit, A. (2014). rTANDEM, an R/Bioconductor package for MS/MS protein identification. Bioinformatics *30*, 2233–2234. https://doi.org/10.1093/bioinformatics/btu178

Freet, D., Agrawal, R., Walker, J.J., and Badr, Y. (2016). Open source cloud management platforms and hypervisor technologies: A review and comparison. SoutheastCon, 2016 IEEE, pp. 1–8. https://doi.org/10.1109/SECON.2016.7506698

Furht, B., and Escalante, A. (2010). Handbook of cloud computing (Springer). ISBN: 978-1-4419-6523-3

Gagliardi, A., Mullin, N.P., Ying Tan, Z., Colby, D., Kousa, A.I., Halbritter, F., Weiss, J.T., Felker, A., Bezstarosti, K., Favaro, R., et al. (2013). A direct physical interaction between Nanog and Sox2 regulates embryonic stem cell self-renewal. EMBO J. *32*, 2231–2247. https://doi.org/10.1038/emboj.2013.161

Gamma, E. (1995). Design patterns: elements of reusable object-oriented software (Addison-Wesley). ISBN: 978-0-201-63361-0

Garijo, D., Kinnings, S., Xie, L., Xie, L., Zhang, Y., Bourne, P.E., and Gil, Y. (2013). Quantifying reproducibility in computational biology: the case of the tuberculosis drugome. PLoS ONE *8*, e80278. https://doi.org/10.1371/journal.pone.0080278

Gatto, L., and Christoforou, A. (2014). Using R and Bioconductor for proteomics data analysis. Biochimica et Biophysica Acta (BBA) - Proteins and Proteomics *1844*, 42–51. https://doi.org/10.1016/j.bbapap.2013.04.032

Gauch, H.G. (2003). Scientific method in practice (Cambridge University Press). ISBN: 978-0-521-81689-2

Geer, L.Y., Markey, S.P., Kowalak, J.A., Wagner, L., Xu, M., Maynard, D.M., Yang, X., Shi, W., and Bryant, S.H. (2004). Open mass spectrometry search algorithm. J. Proteome Res. *3*, 958–964. https://doi.org/10.1021/pr0499491

Gentleman, R.C., Carey, V.J., Bates, D.M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., et al. (2004). Bioconductor: open software development for computational biology and bioinformatics. Genome Biology *5*, R80. https://doi.org/10.1186/gb-2004-5-10-r80

Gillespie, C., and Lovelace, R. (2016). Efficient R programming: a practical guide to smarter programming. (O'Reilly Media) ISBN: 978-1-4919-5078-4

Gorgolewski, K.J., Alfaro-Almagro, F., Auer, T., Bellec, P., Capotă, M., Chakravarty, M.M., Churchill, N.W., Cohen, A.L., Craddock, R.C., Devenyi, G.A., et al. (2017). BIDS apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods. PLoS Comput. Biol. *13*, e1005209. https://doi.org/10.1371/journal.pcbi.1005209

Gould, J. (2015). Data sharing: Fewer experiments, more knowledge. [Web log post]. Retrieved from http://blogs.nature.com/naturejobs/2015/10/21/data-sharing-fewer-experiments-more-knowledge (Accessed August 2017)

Greenberg, J., White, H.C., Carrier, S., and Scherle, R. (2009). A Metadata Best Practice for a Scientific Data Repository. Journal of Library Metadata *9*, 194–212. https://doi.org/10.1080/19386380903405090

Gronenschild, E.H.B.M., Habets, P., Jacobs, H.I.L., Mengelers, R., Rozendaal, N., van Os, J., and Marcelis, M. (2012). The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements. PLoS ONE *7*, e38234. https://doi.org/10.1371/journal.pone.0038234

Guo, Q. (2016). Publishing: Journals, agree on manuscript format. Nature *540*, 525–525. https://doi.org/10.1038/540525d

Guttman, M., Garber, M., Levin, J.Z., Donaghey, J., Robinson, J., Adiconis, X., Fan, L., Koziol, M.J., Gnirke, A., Nusbaum, C., et al. (2010). Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. Nat. Biotechnol. *28*, 503–510. https://doi.org/10.1038/nbt.1633

Haidich, A.B. (2010). Meta-analysis in medical research. Hippokratia *14*, 29–37.

Halbritter, F., Vaidya, H.J., and Tomlinson, S.R. (2012). GeneProf: analysis of high-throughput sequencing experiments. Nat. Methods *9*, 7–8. https://doi.org/10.1038/nmeth.1809

Halbritter, F., Kousa, A.I., and Tomlinson, S.R. (2014). GeneProf data: a resource of curated, integrated and reusable high-throughput genomics experiments. Nucleic Acids Res. *42*, D851-858. https://doi.org/10.1093/nar/gkt966

Han, D.K., Eng, J., Zhou, H., and Aebersold, R. (2001). Quantitative profiling of differentiation-induced microsomal proteins using isotope-coded affinity tags and mass spectrometry. Nat. Biotechnol. *19*, 946–951. https://doi.org/10.1038/nbt1001-946

Hashimoto, M. (2013). Vagrant: up and running (O'Reilly Media). ISBN: 978-1-4493-3583-0

Helmke, M., Joseph, E.K., Rey, J.A., and Hill, B.M. (2017). The official Ubuntu book (Prentice Hall). ISBN: 978-0-13-451342-3

Holman, J.D., Tabb, D.L., and Mallick, P. (2014). Employing ProteoWizard to Convert Raw Mass Spectrometry Data. Curr Protoc Bioinformatics *46*, 13.24.1-9. https://doi.org/10.1002/0471250953.bi1324s46

Huang, S., Chaudhary, K., and Garmire, L.X. (2017). More Is Better: Recent Progress in Multi-Omics Data Integration Methods. Front Genet *8*, 84. https://doi.org/10.3389/fgene.2017.00084

Huerta, M., Downing, G., Haseltine, F., Seto, B., and Liu, Y. (2000). NIH working definition of bioinformatics and computational biology. US National Institute of Health. http://www.bisti.nih.gov/docs/CompuBioDef.pdf (Accessed August 2017).

Ihaka, R., and Gentleman, R. (1996). R: A Language for Data Analysis and Graphics. Journal of Computational and Graphical Statistics *5*, 299–314. https://doi.org/10.1080/10618600.1996.10474713

Ince, D.C., Hatton, L., and Graham-Cumming, J. (2012). The case for open computer programs. Nature *482*, 485–488. https://doi.org/10.1038/nature10836

Ioannidis, J.P.A., Allison, D.B., Ball, C.A., Coulibaly, I., Cui, X., Culhane, A.C., Falchi, M., Furlanello, C., Game, L., Jurman, G., et al. (2009). Repeatability of published microarray gene expression analyses. Nat. Genet. *41*, 149–155. https://doi.org/10.1038/ng.295

Jang, M.H. (2006). Linux annoyances for geeks (O'Reilly Media). ISBN: 978-0596008017

Jones, A.R., Eisenacher, M., Mayer, G., Kohlbacher, O., Siepen, J., Hubbard, S.J., Selley, J.N., Searle, B.C., Shofstahl, J., Seymour, S.L., et al. (2012). The mzIdentML data standard for mass spectrometry-based proteomics results. Mol. Cell Proteomics *11*, M111.014381. https://doi.org/10.1074/mcp.M111.014381

Jones, P., Côté, R.G., Martens, L., Quinn, A.F., Taylor, C.F., Derache, W., Hermjakob, H., and Apweiler, R. (2006). PRIDE: a public repository of protein and peptide identifications for the proteomics community. Nucleic Acids Res. *34*, D659-663. https://doi.org/10.1093/nar/gkj138

Kanehisa, M., and Bork, P. (2003). Bioinformatics in the post-sequence era. Nat. Genet. *33 Suppl*, 305–310. https://doi.org/10.1038/ng1109

Kangas, L.J., Metz, T.O., Isaac, G., Schrom, B.T., Ginovska-Pangovska, B., Wang, L., Tan, L., Lewis, R.R., and Miller, J.H. (2012). In silico identification software (ISIS): a machine learning approach to tandem mass spectral identification of lipids. Bioinformatics *28*, 1705–1713. https://doi.org/10.1093/bioinformatics/bts194

Kapp, E.A., Schütz, F., Connolly, L.M., Chakel, J.A., Meza, J.E., Miller, C.A., Fenyo, D., Eng, J.K., Adkins, J.N., Omenn, G.S., et al. (2005). An evaluation, comparison, and accurate benchmarking of several publicly available MS/MS search algorithms: sensitivity and specificity analysis. Proteomics *5*, 3475–3490. https://doi.org/10.1002/pmic.200500126

Keller, A., and Shteynberg, D. (2011). Software pipeline and data analysis for MS/MS proteomics: the trans-proteomic pipeline. Methods Mol. Biol. *694*, 169–189. https://doi.org/10.1007/978-1-60761-977-2_12

Kessner, D., Chambers, M., Burke, R., Agus, D., and Mallick, P. (2008). ProteoWizard: open source software for rapid proteomics tools development. Bioinformatics *24*, 2534–2536. https://doi.org/10.1093/bioinformatics/btn323

Kim, D., Pertea, G., Trapnell, C., Pimentel, H., Kelley, R., and Salzberg, S.L. (2013). TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. Genome Biol. *14*, R36. https://doi.org/10.1186/gb-2013-14-4-r36

Kim, D., Langmead, B., and Salzberg, S.L. (2015). HISAT: a fast spliced aligner with low memory requirements. Nat. Methods *12*, 357–360. https://doi.org/10.1038/nmeth.3317

King, G. (2007). An Introduction to the Dataverse Network as an Infrastructure for Data Sharing. Sociological Methods & Research *36*, 173–199. https://doi.org/10.1177/0049124107306660

Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A. (2007). kvm: the Linux virtual machine monitor. In Proceedings of the Linux Symposium, pp. 225–230.

de Koning, A.P.J., Gu, W., Castoe, T.A., Batzer, M.A., and Pollock, D.D. (2011). Repetitive elements may comprise over two-thirds of the human genome. PLoS Genet. *7*, e1002384. https://doi.org/10.1371/journal.pgen.1002384

Kulkarni, N., Alessandri, L., Panero, R., Arigoni, M., Olivero, M., Cordero, F., Beccuti, M., and Calogero, R.A. (2017). Reproducible Bioinformatics Project: A community for reproducible bioinformatics analysis pipelines. BMC Bioinformatics, 19(10), p.211. https://doi.org/10.1101/239947

Kumar, K., and Kurhekar, M. (2016). Economically Efficient Virtualization over Cloud Using Docker Containers. Cloud Computing in Emerging Markets (CCEM), 2016 IEEE International Conference on. IEEE, pp. 95–100. https://doi.org/10.1109/CCEM.2016.025

Lam, H., Deutsch, E.W., Eddes, J.S., Eng, J.K., King, N., Stein, S.E., and Aebersold, R. (2007). Development and validation of a spectral library searching method for peptide identification from MS/MS. Proteomics 7, 655–667. https://doi.org/10.1002/pmic.200600625

Langmead, B., and Salzberg, S.L. (2012). Fast gapped-read alignment with Bowtie 2. Nat Meth *9*, 357–359. https://doi.org/10.1038/nmeth.1923

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S.L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biology *10*, R25. https://doi.org/10.1186/gb-2009-10-3-r25

Lecarpentier, D., Wittenburg, P., Elbers, W., Michelini, A., Kanso, R., Coveney, P., and Baxter, R. (2013). EUDAT: A New Cross-Disciplinary Data Infrastructure for Science. International Journal of Digital Curation *8*, 279–287. https://doi.org/10.2218/ijdc.v8i1.260

Leek, J.T., and Peng, R.D. (2015). Opinion: Reproducible research can still be wrong: adopting a prevention approach. Proc. Natl. Acad. Sci. U.S.A. *112*, 1645–1646. https://doi.org/10.1073/pnas.1421412111

Leinonen, R., Sugawara, H., Shumway, M., and International Nucleotide Sequence Database Collaboration (2011). The sequence read archive. Nucleic Acids Res. *39*, D19-21. https://doi.org/10.1093/nar/gkq1019

Lesluyes, T., Johnson, J., Machanick, P., and Bailey, T.L. (2014). Differential motif enrichment analysis of paired ChIP-seq experiments. BMC Genomics *15*, 752. https://doi.org/10.1186/1471-2164-15-752

Liu, J., Erassov, A., Halina, P., Canete, M., Nguyen, D.V., Chung, C., Cagney, G., Ignatchenko, A., Fong, V., and Emili, A. (2008). Sequential interval motif search: unrestricted

database surveys of global MS/MS data sets for detection of putative post-translational modifications. Anal. Chem. 80, 7846–7854. https://doi.org/10.1021/ac8009017

Li, H., and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics *25*, 1754–1760. https://doi.org/10.1093/bioinformatics/btp324

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics *25*, 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

Li, W., Ji, L., Goya, J., Tan, G., and Wysocki, V.H. (2011). SQID: an intensity-incorporated protein identification algorithm for tandem mass spectrometry. J. Proteome Res. *10*, 1593–1602. https://doi.org/10.1021/pr100959y

Li, X.-J., Zhang, H., Ranish, J.A., and Aebersold, R. (2003). Automated statistical analysis of protein abundance ratios from data generated by stable-isotope dilution and tandem mass spectrometry. Anal. Chem. *75*, 6648–6657. https://doi.org/10.1021/ac034633i

Liao, Y., Smyth, G.K., and Shi, W. (2014). featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. Bioinformatics *30*, 923–930. https://doi.org/10.1093/bioinformatics/btt656

Lithgow, G.J., Driscoll, M., and Phillips, P. (2017). A long journey to reproducible results. Nature *548*, 387–388. https://doi.org/10.1038/548387a

Lloyd, S. (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory *28*, 129–137. https://doi.org/10.1109/TIT.1982.1056489

Lowagie, B. (2011). iText in action (Manning Publications). ISBN: 978-1-935182-61-0

MacLean, B., Tomazela, D.M., Shulman, N., Chambers, M., Finney, G.L., Frewen, B., Kern, R., Tabb, D.L., Liebler, D.C., and MacCoss, M.J. (2010). Skyline: an open source document editor for creating and analyzing targeted proteomics experiments. Bioinformatics *26*, 966–968. https://doi.org/10.1093/bioinformatics/btq054

Ma, K., Vitek, O., and Nesvizhskii, A.I. (2012). A statistical model-building perspective to identification of MS/MS spectra with PeptideProphet. BMC Bioinformatics 13 Suppl 16, S1. https://doi.org/10.1186/1471-2105-13-S16-S1

Matelsky, J., Kiar, G., Johnson, E., Rivera, C., Toma, M., and Gray-Roncal, W. (2018). Container-Based Clinical Solutions for Portable and Reproducible Image Analysis. J Digit Imaging. https://doi.org/10.1007/s10278-018-0089-4

McDonald, W.H., Tabb, D.L., Sadygov, R.G., MacCoss, M.J., Venable, J., Graumann, J., Johnson, J.R., Cociorva, D., and Yates, J.R. (2004). MS1, MS2, and SQT-three unified, compact, and easily parsed file formats for the storage of shotgun proteomic spectra and identifications. Rapid Commun. Mass Spectrom. *18*, 2162–2168. https://doi.org/10.1002/rcm.1603

McGarvey, A.C., Rybtsov, S., Souilhol, C., Tamagno, S., Rice, R., Hills, D., Godwin, D., Rice, D., Tomlinson, S.R., and Medvinsky, A. (2017). A molecular roadmap of the AGM region reveals BMPER as a novel regulator of HSC maturation. J. Exp. Med. *214*, 3731–3751. https://doi.org/10.1084/jem.20162012

Menegidio, F.B., Jabes, D.L., Costa de Oliveira, R., and Nunes, L.R. (2018). Dugong: a Docker image, based on Ubuntu Linux, focused on reproducibility and replicability for bioinformatics analyses. Bioinformatics *34*, 514–515. https://doi.org/10.1093/bioinformatics/btx554

Mitchell, R. (1990). Managing complexity in software engineering (Peregrinus on behalf of the Institution of Electrical Engineers). ISBN: 978-0-86341-171-7

Molloy, J.C. (2011). The Open Knowledge Foundation: Open Data Means Better Science. PLoS Biology *9*, e1001195. https://doi.org/10.1371/journal.pbio.1001195

Monroe, D. (2015). When data is not enough. Communications of the ACM *58*, 12–14. https://doi.org/10.1145/2833138

Morabito, R., Kjallman, J., and Komu, M. (2015). Hypervisors vs. Lightweight Virtualization: A Performance Comparison. Cloud Engineering (IC2E), 2015 IEEE International Conference on. pp. 386–393. https://doi.org/10.1109/IC2E.2015.74

Mouat, A. (2015). Using Docker (O'Reilly Media). ISBN: 978-1-4919-1576-9

Mularien, P. (2010). Spring security 3 (Packt Publising). ISBN: 978-1-84719-975-1

Mullin, N.P., Yates, A., Rowe, A.J., Nijmeijer, B., Colby, D., Barlow, P.N., Walkinshaw, M.D., and Chambers, I. (2008). The pluripotency rheostat Nanog functions as a dimer. Biochem. J. *411*, 227–231. https://doi.org/10.1042/BJ20080134

Narlikar, L., and Jothi, R. (2012). ChIP-Seq data analysis: identification of protein-DNA binding sites with SISSRs peak-finder. Methods Mol. Biol. *802*, 305–322. https://doi.org/10.1007/978-1-61779-400-1_20

National Research Council (US) Committee on Responsibilities of Authorship in the Biological Sciences (2003). Sharing Publication-Related Data and Materials: Responsibilities of Authorship in the Life Sciences (National Academies Press (US)). ISBN: 0-309-08859-3

Nesvizhskii, A.I., Keller, A., Kolker, E., and Aebersold, R. (2003). A Statistical Model for Identifying Proteins by Tandem Mass Spectrometry. Analytical Chemistry 75, 4646–4658. https://doi.org/10.1021/ac0341261

Network Working Group. (2007). RFC 4918: HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). https://www.ietf.org/rfc/rfc4918.txt (Accessed August 2017)

Nguyen, N., and Bein, D. (2017). Distributed MPI cluster with Docker Swarm mode. Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual, pp. 1–7. https://doi.org/10.1109/CCWC.2017.7868429

Nie, J. (2013). A Study on the Application Cost of Server Virtualisation. Computational Intelligence and Security (CIS), 2013 9th International Conference on. IEEE, pp. 807–811. https://doi.org/10.1109/CIS.2013.176

Nookaew, I., Papini, M., Pornputtapong, N., Scalcinati, G., Fagerberg, L., Uhlén, M., and Nielsen, J. (2012). A comprehensive comparison of RNA-Seq-based transcriptome analysis from reads to differential gene expression and cross-comparison with microarrays: a case study in Saccharomyces cerevisiae. Nucleic Acids Res. *40*, 10084–10097. https://doi.org/10.1093/nar/gks804

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., et al. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics *20*, 3045–3054. https://doi.org/10.1093/bioinformatics/bth361

Pancake, C.M. (1995). The emperor has no clothes: what HPC users need to say and HPC vendors need to hear. In Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference, (IEEE), pp. 1–2. https://doi.org/10.1109/SUPERC.1995.242449

Park, S.K., and Miller, K.W. (1988). Random number generators: good ones are hard to find. Communications of the ACM *31*, 1192–1201. https://doi.org/10.1145/63039.63042

Pasquetto, I.V., Randles, B.M., and Borgman, C.L. (2017). On the Reuse of Scientific Data. Data Science Journal *16*. p. 8. https://doi.org/10.5334/dsj-2017-008

Pedrioli, P.G.A. (2010). Trans-proteomic pipeline: a pipeline for proteomic analysis. Methods Mol. Biol. 604, 213–238. https://doi.org/10.1007/978-1-60761-444-9_15

Pepple, K. (2011). Deploying OpenStack (O'Reilly Media). ISBN: 978-1-4493-1105-6

Perez, F., Granger, B.E., and Hunter, J.D. (2011). Python: An Ecosystem for Scientific Computing. Computing in Science & Engineering *13*, 13–21. https://doi.org/10.1109/MCSE.2010.119

Perkins, D.N., Pappin, D.J., Creasy, D.M., and Cottrell, J.S. (1999). Probability-based protein identification by searching sequence databases using mass spectrometry data. Electrophoresis *20*, 3551–3567. https://doi.org/10.1002/(SICI)1522-2683(19991201)20:18%3C3551::AID-ELPS3551%3E3.0.CO;2-2

Piehowski, P.D., Petyuk, V.A., Orton, D.J., Xie, F., Moore, R.J., Ramirez-Restrepo, M., Engel, A., Lieberman, A.P., Albin, R.L., Camp, D.G., et al. (2013). Sources of technical variability in quantitative LC-MS proteomics: human brain tissue sample analysis. J. Proteome Res. *12*, 2128–2137. https://doi.org/10.1021/pr301146m

Popper, K.R. (1959). The Logic of scientific discovery, Second edition (Routledge). ISBN: 978-0-415-27844-7

Postel, J., and Reynolds, J. (1983). RFC 854: Telnet Protocol. Information Sciences Institute. https://www.ietf.org/rfc/rfc854.txt (Accessed August 2017)

Poulton, N. (2017). The Kubernetes Book (Independently published). ISBN: 978-1-5218-2363-7

Prinz, F., Schlange, T., and Asadullah, K. (2011). Believe it or not: how much can we rely on published data on potential drug targets? Nat Rev Drug Discov *10*, 712. https://doi.org/10.1038/nrd3439-c1

Quackenbush, J. (2002). Microarray data normalization and transformation. Nature Genetics *32*, 496–501. https://doi.org/10.1038/ng1032

Quinlan, A.R., and Hall, I.M. (2010). BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics *26*, 841–842. https://doi.org/10.1093/bioinformatics/btq033

Racine, J.S. (2012). RStudio: A Platform-Independent IDE for R and Sweave. J. Appl. Econ. *27*, 167–172. https://doi.org/10.1002/jae.1278

Ramey, C., and Fox, B. (2002). Bash reference manual: reference documentation for Bash (Network Theory). ISBN: 978-0-9541617-7-4

Rapier, C., and Bennett, B. (2008). High speed bulk data transfer using the SSH protocol. Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications,

tools, infrastructures, interoperability, and the incremental adoption of key capabilities, p. 11. https://doi.org/10.1145/1341811.1341824

Rauch, A., Bellew, M., Eng, J., Fitzgibbon, M., Holzman, T., Hussey, P., Igra, M., Maclean, B., Lin, C.W., Detter, A., et al. (2006). Computational Proteomics Analysis System (CPAS): an extensible, open-source analytic system for evaluating and publishing proteomic data and high throughput biological experiments. J. Proteome Res. *5*, 112–121. https://doi.org/10.1021/pr0503533

Reitz, K., and Schlusser, T. (2016). The hitchhiker's guide to Python: best practices for development (O'Reilly Media). ISBN: 978-1491933176

Ritchie, D.M., and Thompson, K. (1978). The UNIX Time-Sharing System. Bell System Technical Journal *57*, 1905–1929. https://doi.org/10.1002/j.1538-7305.1978.tb02136.x

Robertson, G., Hirst, M., Bainbridge, M., Bilenky, M., Zhao, Y., Zeng, T., Euskirchen, G., Bernier, B., Varhol, R., Delaney, A., et al. (2007). Genome-wide profiles of STAT1 DNA association using chromatin immunoprecipitation and massively parallel sequencing. Nat. Methods 4, 651–657. https://doi.org/10.1038/nmeth1068

Robey, R.W., Robey, J.M., and Aulwes, R. (2011). In search of numerical consistency in parallel programming. Parallel Computing *37*, 217–229. https://doi.org/10.1016/j.parco.2011.02.009

Röst, H.L., Sachsenberg, T., Aiche, S., Bielow, C., Weisser, H., Aicheler, F., Andreotti, S., Ehrlich, H.-C., Gutenbrunner, P., Kenar, E., et al. (2016). OpenMS: a flexible open-source software platform for mass spectrometry data analysis. Nat. Methods *13*, 741–748. https://doi.org/10.1038/nmeth.3959

Rung, J., and Brazma, A. (2013). Reuse of public genome-wide gene expression data. Nat. Rev. Genet. *14*, 89–99. https://doi.org/10.1038/nrg3394

Sabharwal, N., and Ravi Shankar (2103). Apache CloudStack Cloud Computing. First edition (Packt Publishing). ISBN: 978-1-78216-011-3

Sansone, S.-A., Rocca-Serra, P., Field, D., Maguire, E., Taylor, C., Hofmann, O., Fang, H., Neumann, S., Tong, W., Amaral-Zettler, L., et al. (2012). Toward interoperable bioscience data. Nat. Genet. *44*, 121–126. https://doi.org/10.1038/ng.1054

Schadt, E.E., Linderman, M.D., Sorenson, J., Lee, L., and Nolan, G.P. (2010). Computational solutions to large-scale data management and analysis. Nat. Rev. Genet. *11*, 647–657. https://doi.org/10.1038/nrg2857

Scheifler, R. (1987). RFC 1013: X Window System Protocol (Version). https://www.ietf.org/rfc/rfc1013.txt (Accessed August 2017)

Schildt, H. (2017). Java. The complete reference (McGraw-Hill Education). ISBN: 978-1-259-58933-1

Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012). OpenStack: toward an open-source solution for cloud computing. International Journal of Computer Applications *55*. https://doi.org/10.5120/8738-2991

Shapin, S., Schaffer, S., and Hobbes, T. (1985). Leviathan and the air-pump: Hobbes, Boyle, and the experimental life: including a translation of Thomas Hobbes, Dialogus physicus de natura aeris by Simon Schaffer (Princeton University Press). ISBN: 978-0-691-08393-3

Sharing images. (2017). Nature Methods *14*, 753–753. https://doi.org/10.1038/nmeth.4389

Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D. (2003). RFC 3530: Network File System (NFS) version 4 Protocol. IETF, April. https://www.ietf.org/rfc/rfc3530.txt (Accessed August 2017)

Sheynkman, G.M., Johnson, J.E., Jagtap, P.D., Shortreed, M.R., Onsongo, G., Frey, B.L., Griffin, T.J., and Smith, L.M. (2014). Using Galaxy-P to leverage RNA-Seq for the discovery of novel protein variations. BMC Genomics *15*, 703. https://doi.org/10.1186/1471-2164-15-703

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The Hadoop Distributed File System. Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on, pp. 1–10. https://doi.org/10.1109/MSST.2010.5496972

Sivaiah, B., Murthy, T.S.N., and Babu, T.V. (2014). Boot multiple Operating Systems from ISO images using USB Disk. Electronics and Communication Systems (ICECS), 2014 International Conference on (IEEE), pp. 1–5. https://doi.org/10.1109/ECS.2014.6892602

Smalley, S., and Loscocco, P. (2001). Integrating flexible support for security policies into the Linux operating system. In Proc. of the USENIX Annual Technical Conference, pp 29—42.

Smith, J.E., and Ravi Nair (2005). The architecture of virtual machines. Computer *38*, 32–38. https://doi.org/10.1109/MC.2005.173

Sollins, K. (1992). RFC 1350: The tftp protocol (revision 2). Network Working Group, MIT. https://www.ietf.org/rfc/rfc1350.txt (Accessed August 2017)

Spinellis, D. (2006). Choosing a programming language. IEEE Software *23*, 62–63. https://doi.org/10.1109/MS.2006.97

Spjuth, O., Bongcam-Rudloff, E., Hernández, G.C., Forer, L., Giovacchini, M., Guimera, R.V., Kallio, A., Korpelainen, E., Kańduła, M.M., Krachunov, M., et al. (2015). Experiences with workflows for automating data-intensive bioinformatics. Biol. Direct *10*, 43. https://doi.org/10.1186/s13062-015-0071-8

St. Laurent, A.M. (2004). Understanding open source and free software licensing (O'Reilly Media). ISBN: 978-0-596-00581-8

Stark, C. (2006). BioGRID: a general repository for interaction datasets. Nucleic Acids Research *34*, D535–D539. https://doi.org/10.1093/nar/gkj109

Tabb, D.L., Fernando, C.G., and Chambers, M.C. (2007). MyriMatch: highly accurate tandem mass spectral peptide identification by multivariate hypergeometric analysis. J. Proteome Res. *6*, 654–661. https://doi.org/10.1021/pr0604054

Tanner, S., Shu, H., Frank, A., Wang, L.-C., Zandi, E., Mumby, M., Pevzner, P.A., and Bafna, V. (2005). InsPecT: identification of posttranslationally modified peptides from tandem mass spectra. Anal. Chem. *77*, 4626–4639. https://doi.org/ 10.1021/ac050102d

Taylor, C.F., Paton, N.W., Lilley, K.S., Binz, P.-A., Julian, R.K., Jones, A.R., Zhu, W., Apweiler, R., Aebersold, R., Deutsch, E.W., et al. (2007). The minimum information about a proteomics experiment (MIAPE). Nature Biotechnology *25*, 887–893. https://doi.org/10.1038/nbt1329

Taylor, C.F., Field, D., Sansone, S.-A., Aerts, J., Apweiler, R., Ashburner, M., Ball, C.A., Binz, P.-A., Bogue, M., Booth, T., et al. (2008). Promoting coherent minimum reporting guidelines for biological and biomedical investigations: the MIBBI project. Nat. Biotechnol. *26*, 889–896. https://doi.org/10.1038/nbt.1411

Taylor, J.A., Walsh, K.A., and Johnson, R.S. (1996). Sherpa: a Macintosh-based expert system for the interpretation of electrospray ionization LC/MS and MS/MS data from protein digests. Rapid Commun. Mass Spectrom. *10*, 679–687. https://doi.org/10.1002/(SICI)1097-0231(199604)10:6<679::AID-RCM528>3.0.CO;2-Q

Tiwari, A., and Sekhar, A.K.T. (2007). Workflow based framework for life science informatics. Computational Biology and Chemistry *31*, 305–319. https://doi.org/10.1016/j.compbiolchem.2007.08.009

Trapnell, C., Pachter, L., and Salzberg, S.L. (2009). TopHat: discovering splice junctions with RNA-Seq. Bioinformatics *25*, 1105–1111. https://doi.org/10.1093/bioinformatics/btp120

Turnbull, J. (2014). The Docker Book: Containerization is the new virtualization (James Turnbull).

Uys, D. (2011). A nice picture of (dependency) hell. [Web log post]. Retrieved from http://disfunksioneel.blogspot.com/2011/04/linux-software-dependencies.html (Accessed August 2017)

Van Heusden, P. (2015). How Galaxy resolves dependencies (or not). [Web log post]. Retrieved from http://pvh.wp.sanbi.ac.za/2015/10/09/how-galaxy-resolves-dependencies-or-not (Accessed August 2017)

Vanin, E.F. (1985). Processed pseudogenes: characteristics and evolution. Annu. Rev. Genet. *19*, 253–272. https://doi.org/10.1146/annurev.ge.19.120185.001345

Vaudel, M., Barsnes, H., Berven, F.S., Sickmann, A., and Martens, L. (2011). SearchGUI: An open-source graphical user interface for simultaneous OMSSA and X!Tandem searches. Proteomics *11*, 996–999. https://doi.org/10.1002/pmic.201000595

da Veiga Leprevost, F., Grüning, B.A., Alves Aflitos, S., Röst, H.L., Uszkoreit, J., Barsnes, H., Vaudel, M., Moreno, P., Gatto, L., Weber, J., et al. (2017). BioContainers: an open-source and community-driven framework for software standardization. Bioinformatics *33*, 2580–2582. https://doi.org/10.1093/bioinformatics/btx192

Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. (2015). Large-scale cluster management at Google with Borg. Proceedings of the European Conference on Computer Systems (EuroSys), pp. 1–17. https://doi.org/10.1145/2741948.2741964

Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D., and Maltzahn, C. (2006). Ceph: A scalable, high-performance distributed file system. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, (USENIX Association), pp. 307–320.

Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L.B., Bourne, P.E., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. Sci Data *3*, 160018. https://doi.org/10.1038/sdata.2016.18

Wu, C.H., Apweiler, R., Bairoch, A., Natale, D.A., Barker, W.C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., et al. (2006). The Universal Protein Resource (UniProt):

an expanding universe of protein information. Nucleic Acids Res. *34*, D187-191. https://doi.org/10.1093/nar/gkj161

Xia, C., Yu, G., and Tang, M. (2009). Efficient implement of orm (object/relational mapping) use in j2ee framework: Hibernate. In Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference On, (IEEE), pp. 1–3. https://doi.org/10.1109/CISE.2009.5365905

Xu, H., Handoko, L., Wei, X., Ye, C., Sheng, J., Wei, C.-L., Lin, F., and Sung, W.-K. (2010). A signal-noise model for significance analysis of ChIP-seq with negative control. Bioinformatics *26*, 1199–1204. https://doi.org/10.1093/bioinformatics/btq128

Yadav, A.K., Kumar, D., and Dash, D. (2011). MassWiz: A Novel Scoring Algorithm with Target-Decoy Based Analysis Pipeline for Tandem Mass Spectrometry. Journal of Proteome Research 10, 2154–2160. https://doi.org/10.1021/pr200031z

Ylonen, T., and Lonvick, C. (2012a). RFC 4251: The secure shell (ssh) protocol architecture. https://www.ietf.org/rfc/rfc4251.txt (Accessed August 2017)

Ylonen, T., and Lonvick, C. (2012b). RFC 4252: The Secure Shell (SSH) Authentication Protocol. https://www.ietf.org/rfc/rfc4252.txt (Accessed August 2017)

Yourdon, E., and Constantine, L.L. (1979). Structured design: fundamentals of a discipline of computer program and systems design (Prentice Hall). ISBN: 978-0-13-854471-3

Zhang, Y., Liu, T., Meyer, C.A., Eeckhoute, J., Johnson, D.S., Bernstein, B.E., Nusbaum, C., Myers, R.M., Brown, M., Li, W., et al. (2008). Model-based analysis of ChIP-Seq (MACS). Genome Biol. *9*, R137. https://doi.org/10.1186/gb-2008-9-9-r137

10.1 Internet References

http://agmniche.stembio.org – Gene expression in the developing HSC niche. (Accessed August 2017)

https://alpinelinux.org – Alpine Linux, a security-oriented, lightweight Linux distribution based on musl libc and busybox. (Accessed August 2017)

https://www.ansible.com – Ansible, open source software that automates software provisioning, configuration management, and application deployment - Red Hat Inc. (Accessed August 2017)

http://ant.apache.org – Apache Ant, a software tool for automating software build processes - Apache Foundation. (Accessed August 2017)

https://apereo.github.io/cas – CAS Enterprise Single Sign-On. (Accessed August 2017)

https://aws.amazon.com – Amazon Web Services, an on-demand cloud computing platform - Amazon Inc. (Accessed August 2017)

https://azure.microsoft.com – Microsoft Azure, a cloud computing service created by Microsoft. (Accessed August 2017)

http://bioboxes.org – Bioboxes, a standard for creating interchangeable bioinformatics software containers. (Accessed August 2017)

http://www.bioproximity.com.s3-website-us-east-1.amazonaws.com/utility-for-converting-mass-spectrometer-file-formats – Ms2mz, Utility for Converting Mass Spectrometer File Formats – Bioproximity (Accessed August 2017)

https://www.biostars.org/p/90390 – Question: Bwa Mem Have Different Alignment Result When Using Different Threads - Biostars. (Accessed August 2017)

https://brew.sh – Homebrew, The missing package manager for macOS.

https://www.centos.org – CentOS, a Linux distribution that provides a free, enterprise-class, community-supported computing platform. (Accessed August 2017)

http://ceph.com – Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability - Red Hat Inc. (Accessed August 2017)

http://www.chef.io – Chef, a continuous automation platform for delivering infrastructure, compliance, applications, and whatever comes next. (Accessed August 2017)

https://chemdata.nist.gov/dokuwiki/doku.php?id=peptidew:mspepsearch – MS PepSearch, a program for the identification of tandem mass spectra - National Institute of Standards and Technology, Mass Spectrometry Data Center. (Accessed August 2017)

https://chocolatey.org – Chocolatey, the package manager for Windows. (Accessed August 2017)

https://cloud.google.com/compute – Google Compute Engine Scalable, High-Performance Virtual Machines - Google Inc. (Accessed August 2017)

https://cloudstack.apache.org – Apache CloudStack is open source software designed to deploy and manage large networks of virtual machines - Apache Foundation. (Accessed August 2017)

https://code.google.com/archive – The Google Code Archive contains the data found on the Google Code Project Hosting Service, which was turned down in early 2016 - Google Inc. (Accessed August 2017)

https://community.alfresco.com – Alfresco community edition, open source information management - Alfresco Software Inc. (Accessed August 2017)

http://www.coreos.com – CoreOs, an open-source lightweight operating system based on the Linux kernel. (Accessed August 2017)

https://coreos.com/os/docs/latest/booting-with-pxe.html – Booting CoreOS Container Linux via PXE - CoreOS. (Accessed August 2017)

https://www.cpan.org – The Comprehensive Perl Archive Network (CPAN) (Accessed August 2017)

https://cran.r-project.org – R precompiled binary distributions of the base system and contributed packages - The Comprehensive R Archive Network. (Accessed August 2017)

http://cxf.apache.org – Apache CXF, an open source services framework which helps you build and develop services using frontend programming APIs - Apache Foundation. (Accessed August 2017)

http://www.dans.knaw.nl – DANS, the Dutch institute for permanent access to digital research data. (Accessed August 2017)

https://data.mendeley.com – The Mendeley data publishing platform - Mendeley. (Accessed August 2017)

http://datahub.io – Find, Share and Publish Quality Data - Dathub. (Accessed August 2017)

http://www.dest-unreach.org/socat – socat - Multipurpose relay (Accessed August 2017)

https://www.dhcp4java.com – dhcp4java, the free open-source DHCP API for Java. (Accessed August 2017)

http://www.docker.com – Docker, a computer program that performs operating-system-level virtualization also known as containerization. (Accessed August 2017)

http://www.ebi.ac.uk/Tools/rcloud – R Cloud Workbench, Remote access to R/Bioconductor on EBI's 64-bit Linux Cluster - EMBL-EBI. (Accessed August 2017)

https://eclipse.org – Eclipse, an open source integrated development environment - Eclipse Foundation. (Accessed August 2017)

http://www.ecma-international.org/publications/standards/Ecma-119.htm – Standard ECMA-119: Volume and File Structure of CDROM for Information Interchange - ECMA International. (Accessed August 2017)

ftp://ftp.ensembl.org/pub/release-58/fasta/mus_musculus – The *Mus musculus* reference genome, Ensembl, assembly NCBIM37, annotation version 58. (Accessed October 2018)

http://figshare.com – An online digital repository where researchers can preserve and share their research outputs. (Accessed August 2017)

http://www.geneprof.org/GeneProf/help_advancedtopics.jsp#chapter:AdvancedTopics – The GeneProf Manual (Accessed August 2017)

http://www.geneprof.org/GeneProf/show?id=gpXP_000385 – Esrrb Is a Direct Nanog Target Gene that Can Substitute for Nanog Function in Pluripotent Cells - GeneProf Data open web resource. (Accessed August 2017)

http://github.com – GitHub, a web-based hosting service for version control using Git - Github Inc. (Accessed August 2017)

https://github.com/c960657/ftpproxy – Java FTP proxy server - Christian Schmidt. (Accessed August 2017)

https://github.com/gvalkov/tailon – Tailon, a web application for looking at and searching through log files - Georgi Valkov. (Accessed August 2017)

https://github.com/kamilion/customizer – Customizer, an advanced Live CD customization and remastering tool. (Accessed August 2017)

https://github.com/novnc/noVNC – NoVNC, a VNC client using HTML5. (Accessed August 2017)

https://github.com/pyenv/pyenv – Simple Python Version Management: pyenv (Accessed August 2017)

https://github.com/shellinabox/shellinabox – Shell In A Box, a web server that can export arbitrary command line tools to a web based terminal emulator. (Accessed August 2017)

https://github.com/stembio/cumulus – Cumulus GitHub, the source code repository for the Cumulus project. (Accessed August 2017)

https://www.gluster.org – Gluster is a free and open source software scalable network filesystem - Red Hat Inc. (Accessed August 2017)

http://hadoop.apache.org – The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing - Apache Foundation. (Accessed August 2017)

http://hibernate.org – Hibernate, an object-relational mapping tool for the Java programming language - Red Hat Inc. (Accessed August 2017)

https://hub.docker.com – Docker Hub, the default registry for Docker images - Docker Inc. (Accessed August 2017)

https://itextpdf.com – iText, easy PDF generation and manipulation for Java and .NET developers - iText Group NV. (Accessed August 2017)

https://jclouds.apache.org – Apache JClouds, the Java Multi-Cloud Toolkit - Apache Foundation. (Accessed August 2017)

http://www.jcraft.com/weirdx – WeirdX is an X Window System server in pure Java under GNU GPL - JCraft Inc. (Accessed August 2017)

https://jquery.com – jQuery, a fast, small, and feature-rich JavaScript library - The jQuery Foundation. (Accessed August 2017)

http://www.khelekore.org – RabbIT proxy for a faster web - Robert Olofsson. (Accessed August 2017)

https://launchpad.net/ubuntu-mini-remix – Ubuntu Mini Remix, a fully working Ubuntu livecd containing only the minimal set of software to make the system work. (Accessed August 2017)

https://linuxcontainers.org/lxc – LXC, a userspace interface for the Linux kernel containment features. (Accessed August 2017)

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html – Mersenne Twister with improved initialization - Makoto Matsumoto. (Accessed August 2017)

https://maven.apache.org – Apache Maven, a software project management and comprehension tool - The Apache Foundation. (Accessed August 2017)

https://www.mysql.com – MySQL, an open-source relational database management system (RDBMS) - Oracle Inc. (Accessed August 2017)

http://nucleotid.es – Continuous, objective and reproducible evaluation of genome assemblers using docker containers. (Accessed August 2017)

http://openjdk.java.net – OpenJDK (Open Java Development Kit), a free and open source implementation of the Java Platform, Standard Edition. (Accessed August 2017)

https://opensource.org – The Open Source Initiative (OSI), to promote and protect open source software, projects and communities. (Accessed August 2017)

https://opensource.org/licenses – Open source licenses & standards - The Open Source Initiative. (Accessed August 2017)

http://www.openstack.org – OpenStack, a free and open-source software platform for cloud computing - OpenStack Foundation. (Accessed August 2017)

https://owncloud.org – Open source client–server software for creating and using file hosting services. (Accessed August 2017)

http://www.packet.net – Packet, a cloud & edge computing infrastructure provider (Accessed August 2017)

https://www.perl.org – Perl, a high-level, general-purpose, interpreted, dynamic programming language. (Accessed August 2017)

https://www.psc.edu/hpn-ssh – High Performance SSH/SCP (HPN-SSH) - Rapier et al. (Accessed August 2017)

http://www.puppet.com – Puppet, an open-source software configuration management tool. (Accessed August 2017)

https://pyd.io – Pydio, On Premise File Sharing. (Accessed August 2017)

https://pypi.python.org – PyPi, find, install and publish Python packages with the Python Package Index. (Accessed August 2017)

https://www.python.org – Python, an interpreted high-level programming language for general-purpose programming - Python Software Foundation. (Accessed August 2017)

https://www.qemu.org – QEMU, a generic and open source machine emulator and virtualizer. (Accessed August 2017)

https://www.ruby-lang.org – Ruby, a dynamic, open source programming language with a focus on simplicity and productivity. (Accessed August 2017)

https://www.seafile.com – Seafile, Enterprise file sync and share platform with high reliability and performance - Seafile Ltd. (Accessed August 2017)

http://www.softlayer.com, (2017) SoftLayer, a dedicated server, managed hosting, and cloud computing provider. (Accessed August 2017)

http://sparkleshare.org – SparkleShare, an open-source cloud storage and file synchronization client app. (Accessed August 2017)

https://spring.io – The Spring Framework is an application framework and inversion of control container for the Java platform - Pivotal Software Inc. (Accessed August 2017)

https://spring.io/projects/spring-security – Spring Security is a powerful and highly customizable authentication and access-control framework - Pivotal Software Inc. (Accessed August 2017)

https://support.microsoft.com/en-us/lifecycle – Microsoft Lifecycle Policy: Consistent and predictable guidelines for the availability of support throughout the life of a product. (Accessed August 2017)

http://www.tinycorelinux.net – The Core Project is a highly modular based system with community build extensions. (Accessed August 2017)

http://tools.proteomecenter.org/wiki/index.php?title=Msconvert_Wine – Msconvert Wine installation and setup wiki - Seattle Proteome Center (Accessed August 2017)

http://tools.proteomecenter.org/wiki/index.php?title=Software:TPP – Trans Proteomic Pipeline, a collection of integrated tools for MS/MS proteomics, developed at the SPC.

http://uck.sf.net – Ubuntu Customization Kit, a tool that helps you customizing official Ubuntu Live CDs. (Accessed August 2017)

http://uk.mathworks.com/products/matlab – MATLAB, combines a desktop environment and a programming language that expresses matrix and array mathematics directly - MathWorks Inc. (Accessed August 2017)

https://www.ukri.org/funding/information-for-award-holders/data-policy/common-principles-on-data-policy. Common principles on data policy - UK Research and Innovation. (Accessed August 2018)

ftp://ftp.uniprot.org/pub/databases/uniprot/previous_releases/release-2017_09/ UniProt previous releases 2017_09. (Accessed September 2018)

https://www.vagrantup.com – Vagrant, building and maintaining portable virtual software development environments - HashiCorp. (Accessed August 2017)

https://www.virtualbox.org – VirtualBox, a powerful x86 and AMD64/Intel64 virtualization product - Oracle Inc. (Accessed August 2017)

http://virtualboxes.org – Virtual Boxes, ready-to-use virtual machines for open-source operating systems. (Accessed August 2017)

http://www.vmware.com – Vmware, cloud computing and platform virtualization software and services - Dell Inc. (Accessed August 2017)

https://wiki.centos.org/HowTos/Virtualization/VirtualBox – The CentOS Wiki: Installing and using VirtualBox on CentOS. (Accessed August 2017)

https://wiki.debian.org/Teams/Dpkg – Dpkg package management team page - Debian. (Accessed August 2017)

http://www.xenproject.org – Xen, a hypervisor using a microkernel design - The Linux Foundation. (Accessed August 2017)

http://yum.baseurl.org – Yum package manager, an automatic updater and package installer/remover for rpm systems. (Accessed August 2017)

http://zenodo.org – The Zenodo research data repository. (Accessed August 2017)

11 Abbreviations

AJAX – Asynchronous JavaScript And XML

API – Application Programming Interface

AWS – Amazon Web Services

BWA – Burrows-Wheeler Aligner

CAS – Central Authentication Service

ChIP-Seq – Chromatin Immunoprecipitation Sequencing

CIFS - Common Internet File System

CLI – Command Line Interface

CPAN – Comprehensive Perl Archive Network

CPU – Central Processing Unit

CSS – Cascading Style Sheet

DHCP - Dynamic Host Configuration Protocol

DNA - Deoxyribonucleic acid

DOS – Denial Of Service

DWR – Direct Web Remoting

EBI – European Bioinformatics Institute

ES – Embryonic Stem

FOSS – Free and Open Source Software

FPS - Frames Per Second

FTP – File Transfer Protocol

GEO – Gene Expression Omnibus

GlusterFS – Gluster File System

HDFS – Hadoop Distributed File System

HPC – High Performance Computation

HPN SSH – High PerformaNce SSH

HTML – Hyper Text Mark-up Language

HTTP – Hyper Text Transfer Protocol

IDE – Integrated Development Environment

IP – Internet Protocol

JSP – Java Server Page

KNIME - Konstanz Information Miner

KVM – Kernel based Virtual Machine

LXC – Linux Containers

MGF – Mascot Generic Format

MIAME - Minimum Information About a Microarray Experiment

MIAPE - Minimum Information About a Proteomic Experiment

MySQL – My Structured Query Language

NAS – Network Attached Storage

NAT – Network Address Translation

NCBI - National Center for Biotechnology Information

NFS – Network File System

ORM – Object-Relational Mapping

OSI – Open Source Initiative

PDF – Portable Document File

PRIDE - PRoteomics IDEntifications

PXE – Pre-execution Environment

QA – Quality Assurance

QEMU – Quick Emulator

RAM – Random Access Memory

REST - Representational State Transfer

RNA – RiboNucleic Acid

RNA-Seq – RiboNucleic Acid Sequencing

S3 – Simple Storage Service

SCP – Secure CoPy

SELinux - Security-Enhanced Linux

SFTP – Secure File Transfer Protocol

SMB – Server Message Block

SOAP - Simple Object Access Protocol

SQL – Structured Query Language

SRA – Sequence Read Archive

SSH – Secure Shell

SSO – Single Sign On

STARD - Standards for the Reporting of Diagnostic

TCP – Transmission Control Protocol

TFTP – Trivial File Transfer Protocol

UDP – User Datagram Protocol

UI – User Interface

URL – Uniform Resource Locator

USB – Universal Serial Bus

VDI – Virtual Disk Image

VHD – Virtual Hard Disk

VMDK – Virtual Machine DisK

WAR – Web Application Resource

WebDAV – Web Distributed Authoring and Versioning

XML – Extensible Mark-up Language

12 Appendices

12.1 Appendix A: A Novel Interactome of Nanog

Expect Value log10	UniProt Identifier	Gene Name
-55	B7ZND6	Rere
-55	Q80TZ9	Rere
-54.8	Q61286	Tcf12
-54.5	Q920S3	Gatad1
-48.8	Q52KB8	Six3
-48.8	Q62233	Six3
-36.4	Q0VGT2	Gli2
-36.2	P70459	Erf
-33.4	Q3UI98	Gtf3c5
-33.4	Q8R2T8	Gtf3c5
-30.8	Q8BY02	Nkrf
-28.3	Q60821	Zbtb17
-22.9	Q6PHC1	Eno1
-22.4	B1AS11	Rere
-20.4	Q9Z248	Aebp2
-17.2	**P48432**	**Sox2**
-17.2	**Q60I23**	**Sox2**
-16.7	B1AY10	Nfx1
-16.2	Q6P8W7	Zfp655
-15.9	Q00417	Tcf7
-15.1	P09065	En1
-15	B2RPW6	Smad7
-15	O35253	Smad7
-14.7	A2ARW8	Gm14399

-14.2	A2ARR7	Gm14412
-13.8	P42225	Stat1
-13	B2KFW1	Zscan20
-12.7	A2BGG7	Ybx1
-11.9	Q80TR0	Nfat5
-11.8	B2ZAC8	Tcf25
-11.8	Q8R3L2	Tcf25
-11.7	D3YVM1	Zscan10
-11.6	E9Q6A9	Ctnnb1
-11.4	P42128	Foxk1
-11.1	P53569	Cebpz
-11	Q8K5C0	Grhl2
-11	Q9DCN4	Grhl2
-10.9	Q62441	Tle4
-10.5	Q08639	Tfdp1
-10.5	Q3V3X3	Tfdp1
-10.5	Q9CYZ7	Tfdp1
-10.4	P18911	Rarg
-10.4	Q80X44	Zbtb24
-10.1	E9QNT5	Taf12
-10.1	Q8VE65	Taf12
-9.9	Q80V18	Trps1
-9.1	Q04888	Sox10
-9	Q3URY5	Zfp941
-9	Q6NV92	Zfp941
-9	Q99LS8	Pbx1
-8.9	A5AA27	Gli2
-8.8	D3Z6V3	Esr1
-8.8	E7FJU2	Esr1

-8.8	P19785	Esr1
-8.8	P45481	Crebbp
-8.7	Q64249	Nr6a1
-8.7	Q8BNQ4	Sp2
-8.7	Q8C5J0	Sp2
-8.7	Q9D2H6	Sp2
-8.6	Q0GE24	Glis3
-8.5	P81269	Atf1
-8.5	Q8BSS2	Foxn2
-5	A2BFU4	Gm14403
-2.4	E9Q6S4	Gm17067
-2.4	E9Q981	Zfp976
-2.4	Q6NVD6	Zfp975

Table 15 - A Novel Interactome of Nanog Dependent on its Tryptophan Repeat Region: A table showing 66 proteins identified from the proteomic analysis detailed in **Section 7.3**. Proteins are listed with the log10 of their expect value, UniProt ID and gene name ordered by the log10 of their expect value. Sox2 is highlighted.

12.2 Appendix B: rTandem Analysis Script

```
# load the global libs
library(rTANDEM)
library(biomaRt)

# set up the spectra files to be looked up
files = c(%%inputFiles%%)

message(paste("Loading", length(files), "files", sep=" "))
taxonomy <- rTTaxo(
          taxon="mouse",
          format="peptide",
          URL="mouse_uprot.fasta.pro"
          )
taxonomy

# set up the global rTandem params
param <- rTParam()
param <- setParamValue(param, 'protein', 'taxon', value="mouse")
param <- setParamValue(param, 'list path', 'taxonomy information', taxonomy)
param <- setParamValue(param, 'list path', 'default parameters',
value=system.file("extdata/default_input.xml", package="rTANDEM"))
param <- setParamValue(param, 'output', 'xsl path', value=system.file("extdata/tandem-input-
style.xsl", package="rTANDEM"))
param <- setParamValue(param, 'output', 'path', value=paste(getwd(), "output.xml", sep="/"))

# make a new dataframe to hold them all
concat <- data.frame()
protconcat <- data.frame()

# loop through the input files
for (file in files) {
  message(paste("Loading", file, sep=" "))
  # load in each file
  param <- setParamValue(param, 'spectrum', 'path', value=file)
  # do a database lookup with rTandem
  result.path <- tandem(param)
  result.R <- GetResultsFromXML(result.path)
  proteins <- GetProteins(result.R, log.expect=-1.3, min.peptides=2)
  list <- proteins[['label']]
  list <- gsub("sp\\|", "", list, perl = T, )
  list <- gsub("\\|", "", list, perl = T, )
  proteins[['label']] <- list

  # annotate
  ensembl.mart<- useMart(biomart="ensembl", dataset="mmusculus_gene_ensembl")
  annotation <- getBM(attributes=c("uniprotswissprot", "external_gene_name"),
filters="uniprotswissprot", values=list, mart=ensembl.mart)

  # concat all the runs
  message(paste("Adding", length(annotation[,'external_gene_name']), "proteins", sep=" "))
  concat <- rbind(concat, annotation)
  protconcat <- rbind(concat, proteins)
  }

# write to a csv
message(paste("Identified", length(concat[,'external_gene_name']), "proteins total", sep=" "))
write.csv(concat, file = "%%outputFile1%%")
write.csv(protconcat, file = "%%outputFile2%%")
```

12.3 Appendix C: Cumulus Analysis Report Windows ProteoWizard

This is the Cumulus analysis execution report for one stage of the workflow with a report of the disk image used. A workflow report consists of one of these reports for each step of the analysis.

Execution Request 513

This is a summary report of execution request 513. It details the execution, the commands and all the software used.

drigodwin Batch: ProteoWizard

Requested By: **drigodwin**
Request status: **COMPLETED**, Exit Status: **0**
Command: **C:\Program Files\ProteoWizard\ProteoWizard 3.0.4416\msconvert.exe --mgf c: /_vBoxExternal/*.raw**
Cores: **2**
RAM: **2 GB**
Internet Enabled HTTP: **YES** FTP: **YES**
Stembio Drive Enabled: **YES**
User Login Enabled: **YES**

Disk Image 52: Cumulus-ProteoWizard

Platform: **X86**

Image Permalink:
http://cumulus.stembio.org/cumulus/download?id=52&typeVirtualContainer

Cumulus Packages

ProteoWizard 3.0.4416 32-bit: 3.0.4416

System Packages

MySQL Server: 5.55.5.13
Microsoft .NET Framework 3.0 Service Pack 2: 3.2.30729
MSXML 4.0 SP2 (KB936181): 4.20.9848.0
Microsoft Visual C++ 2010 x86 Redistributable - 10.0.40219: 10.0.40219
Microsoft .NET Framework 3.5 SP1: 3.5.30729
Microsoft Visual Studio 2010 Tools for Office Runtime (x86): 10.0.50908
Java(TM) SE Development Kit 6 Update 37: 1.6.0.370
Java SE Development Kit 7 Update 65: 1.7.0.650
MSXML 4.0 SP2 (KB927978): 4.20.9841.0
Microsoft .NET Framework 4 Client Profile: 4.0.30319
Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.6161: 9.0.30729.6161
MSXML 4.0 SP2 (KB973688): 4.20.9876.0
Microsoft Visual C++ 2008 Redistributable - x86 9.0.21022: 9.0.21022
7-Zip 9.20: 9.20.00.0
Microsoft Visual C++ 2008 Redistributable - x86 9.0.30729.4148: 9.0.30729.4148
Microsoft Silverlight: 5.1.30514.0
Microsoft .NET Framework 2.0 Service Pack 2: 2.2.30729
MSXML 4.0 SP2 (KB954430): 4.20.9870.0
Microsoft .NET Framework 1.1: 1.1.4322

JavaFX 2.1.1: 2.1.1
Java Auto Updater: 2.1.9.0
Microsoft .NET Framework 4 Extended: 4.0.30319

12.4 Appendix D: Cumulus Analysis Report rTandem

This is the Cumulus analysis execution report for one stage of the workflow with a report of the disk image used. A workflow report consists of one of these reports for each step of the analysis.

Execution Request 514

This is a summary report of execution request 514. It details the execution, the commands and all the software used.

drigodwin Batch: rTandem identification

Requested By: **drigodwin**
Request status: **COMPLETED**, Exit Status: **0**
Command: **/scripts/rTandem/rTandem.sh c: /_vBoxExternal/proteomics/wt.mgf c: /_vBoxExternal/proteomics/output1.csv c: /_vBoxExternal/proteomics/output2.csv**

Cores: **2**
RAM: **2 GB**
Internet Enabled HTTP: **YES** FTP: **YES**
Stembio Drive Enabled: **YES**
User Login Enabled: **YES**

Disk Image 45: Cumulus-rTandem

Platform: **X86**

Image Permalink:
http://cumulus.stembio.org/cumulus/download?id=45&typeVirtualContainer

Cumulus Packages

rTandem (4059): 1.16.0
biomaRt (3442): 2.32.1

System Packages

xserver-xorg-input-vmmouse (1032): 1:13.0.0-1build1 (1033)
tcpd (1034): 7.6.q-25 (1035)
python-urllib3 (1035): 1.7.1-1build1 (1036)
libtext-wrapi18n-perl (1): 0.06-7 (690)
libustr-1.0-1 (690): 1.0.4-3ubuntu2 (1037)
libhttp-date-perl (1036): 6.02-1 (1038)
debconf (691): 1.5.51ubuntu2 (1039)
xserver-common (1037): 2:1.15.1-0ubuntu2.1 (1040)
libasprintf0c2 (692): 0.18.3.1-1ubuntu3 (1041)
libelf1 (693): 0.158-0ubuntu5.1 (1042)
dash (695): 0.5.7-4ubuntu1 (1043)
libsocket6-perl (1038): 0.25-1 (1044)
libsystemd-login0 (861): 204-5ubuntu20.9 (1416)
g++ (1275): 4:4.8.2-1ubuntu6 (1417)
e2fslibs (862): 1.42.9-3ubuntu1 (1418)
python3.4 (1276): 3.4.0-2ubuntu1 (1419)
libgnutls-openssl27 (865): 2.12.23-12ubuntu2.1 (1420)
libharfbuzz0b (1277): 0.9.27-1ubuntu1 (1421)
r-cran-rpart (1279): 4.1-8-1trusty0 (1423)
xfonts-scalable (1280): 1:1.0.3-1 (1424)
libio-html-perl (1281): 1.00-1 (1425)
libkrb5support0 (866): 1.12+dfsg-2ubuntu4.2 (1426)
libfile-fcntllock-perl (1282): 0.14-2build1 (1427)
samba-common (867): 2:4.1.6+dfsg-

libprotobuf-lite8 (1039): 2.5.0-9ubuntu1 (1045)
python-six (1040): 1.5.2-1 (1046)
libpam-cap (1041): 1:2.24-0ubuntu2 (1047)
libpolkit-gobject-1-0 (696): 0.105-4ubuntu2 (1048)
iproute2 (697): 3.12.0-2 (1049)
shared-mime-info (698): 1.2-0ubuntu3 (1050)
r-cran-foreign (1042): 0.8.61-1trusty0 (1051)
libxdmcp6 (699): 1:1.1.1-1 (700)
libgl1-mesa-dri (1043): 10.1.3-0ubuntu0.2 (1052)
libnl-3-200 (700): 3.2.21-1 (1053)
coreutils (701): 8.21-1ubuntu5 (1054)
libopts25 (1044): 1:5.18-2ubuntu2 (1055)
mircommon-dev (1045): 0.1.8+14.04.20140411-0ubuntu1 (1056)
sudo (702): 1.8.9p5-1ubuntu1 (1057)
libfreetype6 (703): 2.5.2-1ubuntu2.2 (1058)
debianutils (704): 4.4 (705)
libxcb-shm0 (1046): 1.10-2ubuntu1 (1059)
libxml-parser-perl (1047): 2.41-1build3 (1060)
libjpeg8-dev (1048): 8c-2ubuntu8 (1061)
libxmuu1 (705): 2:1.1.1-1 (706)
libprotobuf8 (1049): 2.5.0-9ubuntu1 (1062)
x11proto-input-dev (1050): 2.3-1 (1063)
initramfs-tools (706): 0.103ubuntu4.2 (1064)
makedev (707): 2.3.1-93ubuntu1 (708)
libxml2 (708): 2.9.1+dfsg1-3ubuntu4.4 (1065)
linux-headers-3.13.0-24 (1051): 3.13.0-24.47 (1066)
x11-xkb-utils (1052): 7.7+1 (1067)
telnet (709): 0.17-36build2 (710)
login (710): 1:4.1.5.1-1ubuntu9 (1068)
gpgv (711): 1.4.16-1ubuntu2.1 (1069)
python3-dbus (712): 1.2.0-2build2 (1070)
libssl1.0.0 (713): 1.0.1f-1ubuntu2.7 (1071)
libdbus-1-3 (714): 1.6.18-0ubuntu4.3 (1072)
gettext-base (715): 0.18.3.1-1ubuntu3 (1073)
module-init-tools (716): 15-0ubuntu6 (1074)
libffi6 (717): 3.1~rc1+r3.0.13-12 (1075)
man-db (718): 2.6.7.1-1ubuntu1 (1076)
libxcb-shape0-dev (1053): 1.10-2ubuntu1 (1077)
localechooser-data (719): 2.49ubuntu5 (1078)
libcairo2 (1054): 1.13.0~20140204-0ubuntu1 (1079)
libhtml-format-perl (1055): 2.11-1 (1080)
libgdbm3 (720): 1.8.3-12build1 (721)
libgcrypt11 (721): 1.5.3-2ubuntu4.1 (1081)
libpciaccess-dev (1056): 0.13.2-1 (1082)
libxshmfence1 (1057): 1.1-2 (1083)
libgnutlsxx27 (1059): 2.12.23-12ubuntu2.1 (1085)
xserver-xorg-video-sis (1060): 1:0.10.7-0ubuntu6 (1086)
libnet-http-perl (1061): 6.06-1 (1087)
libmpdec2 (1062): 2.4.0-6 (1088)
isc-dhcp-common (722): 4.2.4-7ubuntu12 (1089)
libxinerama1 (1063): 2:1.1.3-1 (1090)
lzma (723): 9.22-2ubuntu2 (724)
libatomic1 (1064): 4.8.2-19ubuntu1 (1091)
libuuid1 (724): 2.20.1-5.1ubuntu20.3 (1092)
libtasn1-6 (1065): 3.4-3ubuntu0.1 (1093)
lsb-base (726): 4.1+Debian11ubuntu6 (1094)
libcurl3-gnutls (727): 7.35.0-1ubuntu2.2 (1095)
r-cran-codetools (1066): 0.2-9-1trusty0 (1096)
fontconfig (1067): 2.11.0-0ubuntu4.1 (1097)
libparted0debian1 (728): 2.3-19ubuntu1 (1098)
cpp-4.8 (1068): 4.8.2-19ubuntu1 (1099)
1ubuntu2.14.04.3 (1428)
wget (868): 1.15-1ubuntu1.14.04.1 (1429)
libgl1-mesa-glx (1283): 10.1.3-0ubuntu0.2 (1430)
libarchive-extract-perl (1284): 0.70-1 (1431)
libxcb-present0 (1285): 1.10-2ubuntu1 (1432)
x11-utils (1286): 7.7+1 (1433)
base-passwd (869): 3.5.33 (1434)
uuid-runtime (870): 2.20.1-5.1ubuntu20.3 (1435)
bind9-host (871): 1:9.9.5.dfsg-3 (1436)
python3-commandnotfound (872): 0.3ubuntu12 (1437)
libalgorithm-diff-xs-perl (1287): 0.04-2build4 (1438)
x11proto-xinerama-dev (1288): 1.2.1-2 (1439)
libfakeroot (1289): 1.20-3ubuntu2 (1440)
libpaper-utils (1290): 1.1.24+nmu2ubuntu3 (1441)
libcomerr2 (873): 1.42.9-3ubuntu1 (1442)
libterm-ui-perl (1291): 0.42-1 (1443)
libfile-listing-perl (1292): 6.04-1 (1444)
mawk (874): 1.3.3-17ubuntu2 (1445)
python3-chardet (1489): 2.0.1-1 (1767)
linux-headers-3.13.0-40-generic (1293): 3.13.0-40.69 (1446)
libtasn1-6-dev (1294): 3.4-3ubuntu0.1 (1447)
plymouth-theme-ubuntu-text (875): 0.8.8-0ubuntu17 (1448)
groff-base (876): 1.22.2-5 (1449)
ntp (1295): 1:4.2.6.p5+dfsg-3ubuntu2 (1768)
gnupg (877): 1.4.16-1ubuntu2.1 (1451)
libhttp-daemon-perl (1296): 6.01-1 (1452)
xserver-xorg-video-cirrus (1297): 1:1.5.2-1build1 (1453)
libnewt0.52 (878): 0.52.15-2ubuntu5 (1454)
ltrace (879): 0.7.3-4ubuntu5.1 (1455)
libalgorithm-diff-perl (1298): 1.19.02-3 (1456)
libisccc90 (880): 1:9.9.5.dfsg-3 (1457)
libtevent0 (881): 0.9.19-1 (1458)
x11-apps (1299): 7.7+2 (1459)
x11-common (1300): 1:7.7+1ubuntu8 (1460)
libxdamage-dev (1301): 1:1.1.4-1ubuntu1 (1461)
net-tools (882): 1.60-25ubuntu2.1 (1462)
wireless-regdb (883): 2013.02.13-1ubuntu1 (884)
libxfixes3 (1302): 1:5.0.1-1ubuntu1 (1463)
xserver-xorg-video-r128 (1303): 6.9.2-1build1 (1464)
xdg-utils (1304): 1.1.0~rc1-2ubuntu7.1 (1465)
unattended-upgrades (1305): 0.82.1ubuntu2 (1466)
libaccountsservice0 (884): 0.6.35-0ubuntu7.1 (1467)
r-base-dev (1306): 3.1.2-1trusty0 (1468)
libfs6 (1307): 2:1.0.5-1 (1469)
libtcl8.6 (1308): 8.6.1-4ubuntu1 (1470)
debconf-i18n (885): 1.5.51ubuntu2 (1471)
ubuntu-release-upgrader-core (886): 1:0.220.5 (1472)
libkadm5srv-mit9 (1309): 1.12+dfsg-2ubuntu4.2 (1473)
libxmu6 (1310): 2:1.1.1-1 (1474)
fuse (887): 2.9.2-4ubuntu4 (1475)
grep (888): 2.16-1 (1476)
libtirpc1 (1311): 0.2.2-5ubuntu2 (1477)
libnl-genl-3-200 (889): 3.2.21-1 (1478)
busybox-static (890): 1:1.21.0-1ubuntu1 (1479)
gcc-4.8-base (891): 4.8.2-19ubuntu1 (1480)
libxcb-sync1 (1312): 1.10-2ubuntu1 (1481)
plymouth (892): 0.8.8-0ubuntu17 (1482)
libdrm-nouveau2 (1313): 2.4.52-1 (1483)
libxtables10 (893): 1.4.21-1ubuntu1 (1484)

user-setup (729): 1.48ubuntu2 (1100)
procps (730): 1:3.3.9-1ubuntu2 (1101)
g++-4.8 (1069): 4.8.2-19ubuntu1 (1102)
libgeoip1 (731): 1.6.0-1 (1103)
dnsutils (732): 1:9.9.5.dfsg-3 (1104)
python3-pycurl (1070): 7.19.3-0ubuntu3 (1105)
r-cran-class (1071): 7.3-11-1trusty0 (1106)
libllvm3.4 (1072): 1:3.4-1ubuntu3 (1107)
libboost-system1.54.0 (1073): 1.54.0-4ubuntu3.1 (1108)
libgssapi-krb5-2 (733): 1.12+dfsg-2ubuntu4.2 (1109)
libxcb-util0 (1074): 0.3.8-2ubuntu1 (1110)
python-minimal (1075): 2.7.5-5ubuntu3 (1111)
libfuse2 (734): 2.9.2-4ubuntu4 (1112)
libpython3.4-minimal (1076): 3.4.0-2ubuntu1 (1113)
libmirprotobuf0 (1077): 0.1.8+14.04.20140411-0ubuntu1 (1114)
r-cran-nlme (1078): 3.1.118-1trusty0 (1115)
ncurses-base (735): 5.9+20140118-1ubuntu1 (1116)
libgraphite2-3 (1079): 1.2.4-1ubuntu1 (1117)
libauthen-sasl-perl (1081): 2.1500-1 (1119)
libreadline-dev (1082): 6.3-4ubuntu2 (1120)
libpopt0 (736): 1.16-8ubuntu1 (1121)
ubuntu-defaults-umr (1083): 0.1 (1122)
libpam0g (737): 1.1.8-1ubuntu2 (1123)
libxcb-randr0 (1084): 1.10-2ubuntu1 (1124)
libpci3 (738): 1:3.2.1-1ubuntu5 (1125)
autotools-dev (1085): 20130810.1 (1126)
cdbs (1086): 0.4.122ubuntu2 (1127)
libxfixes-dev (1087): 1:5.0.1-1ubuntu1 (1128)
libusb-0.1-4 (740): 2:0.1.12-23.3ubuntu1 (1129)
libpython2.7 (1088): 2.7.6-8 (1130)
libgpg-error0 (741): 1.12-0.2ubuntu1 (1131)
psmisc (742): 22.20-1ubuntu2 (743)
ucf (743): 3.0027+nmu1 (744)
libgssrpc4 (1089): 1.12+dfsg-2ubuntu4.2 (1132)
libcurl4-openssl-dev (1090): 7.35.0-1ubuntu2.2 (1133)
perl-modules (1091): 5.18.2-2ubuntu1 (1134)
bsdutils (744): 1:2.20.1-5.1ubuntu20.3 (1135)
libgl1-mesa-dev (1092): 10.1.3-0ubuntu0.2 (1136)
python3-apt (745): 0.9.3.5 (1137)
sensible-utils (746): 0.0.9 (747)
iw (747): 3.4-1 (748)
libwayland-server0 (1093): 1.4.0-1ubuntu1 (1138)
libwbclient0 (748): 2:4.1.6+dfsg-1ubuntu2.14.04.3 (1140)
x11proto-gl-dev (1095): 1.4.17-1 (1141)
install-info (749): 5.2.0.dfsg.1-2 (1142)
libmirprotobuf-dev (1096): 0.1.8+14.04.20140411-0ubuntu1 (1143)
libxss1 (1097): 1:1.2.2-1 (1144)
libfile-basedir-perl (1098): 0.03-1fakesync1 (1145)
libxext-dev (1099): 2:1.3.2-1 (1146)
dh-apparmor (1100): 2.8.95~2430-0ubuntu5.1 (1147)
liblwres90 (750): 1:9.9.5.dfsg-3 (1148)
libxkbfile1 (1101): 1:1.0.8-1 (1149)
lsb-release (751): 4.1+Debian11ubuntu6 (1150)
debhelper (1102): 9.20131227ubuntu1 (1151)
krb5-multidev (1103): 1.12+dfsg-2ubuntu4.2 (1152)
libx11-xcb-dev (1104): 2:1.6.2-1ubuntu2 (1153)
dpkg-dev (1105): 1.17.5ubuntu5.3 (1154)
dkms (1106): 2.2.0.3-1.1ubuntu5 (1155)
init-system-helpers (1107): 1.14 (1156)
python3-gdbm (894): 3.4.0-0ubuntu1 (1485)
kbd (895): 1.15.5-1ubuntu1 (896)
linux-image-generic (896): 3.13.0.24.29 (1486)
libasan0 (1314): 4.8.2-19ubuntu1 (1487)
libcurl3 (1315): 7.35.0-1ubuntu2.2 (1488)
liblapack-dev (1316): 3.5.0-2ubuntu1 (1489)
autoconf (1317): 2.69-6 (1490)
libxdamage1 (1318): 1:1.1.4-1ubuntu1 (1491)
x11-xserver-utils (1319): 7.7+2ubuntu1 (1492)
libxaw7 (1320): 2:1.0.12-1 (1493)
keyboard-configuration (897): 1.70ubuntu8 (1494)
libxcb-render0-dev (1321): 1.10-2ubuntu1 (1495)
upstart (898): 1.12.1-0ubuntu4.2 (1496)
xserver-xorg-video-neomagic (1322): 1:1.2.8-1build1 (1497)
mlocate (899): 0.26-1ubuntu1 (900)
ureadahead (900): 0.100.0-16 (901)
libgfortran3 (1323): 4.8.2-19ubuntu1 (1498)
command-not-found-data (901): 0.3ubuntu12 (1499)
ftp (902): 0.17-28 (903)
libpipeline1 (903): 1.3.0-1 (1500)
python-tdb (1324): 1.2.12-1 (1501)
bash-completion (904): 1:2.1-4 (1502)
libsasl2-modules (906): 2.1.25.dfsg1-17build1 (1503)
libsasl2-2 (907): 2.1.25.dfsg1-17build1 (1504)
passwd (908): 1:4.1.5.1-1ubuntu9 (1505)
bsdmainutils (909): 9.0.5ubuntu1 (910)
hostname (910): 3.15ubuntu1 (1506)
r-base-html (1325): 3.1.2-1trusty0 (1507)
xkb-data (911): 2.10.1-1ubuntu1 (1508)
fakeroot (1326): 1.20-3ubuntu2 (1509)
libacl1 (912): 2.2.52-1 (913)
libpthread-stubs0-dev (1327): 0.3-4 (1510)
screen (1328): 4.1.0~20120320gitdb59704-9 (1511)
libtxc-dxtn-s2tc0 (1329): 0~git20131104-1.1 (1512)
libpam-runtime (913): 1.1.8-1ubuntu2 (1513)
geoip-database (914): 20140313-1 (1514)
libjpeg8 (1330): 8c-2ubuntu8 (1515)
xserver-xorg-video-modesetting (1331): 0.8.1-1build1 (1516)
xauth (915): 1:1.0.7-1ubuntu1 (916)
libjpeg-turbo8-dev (1332): 1.3.0-0ubuntu2 (1517)
ufw (916): 0.34~rc-0ubuntu2 (1519)
libslang2 (917): 2.2.4-15ubuntu1 (918)
x11proto-xf86bigfont-dev (1334): 1.2.0-3 (1520)
libtinfo-dev (1335): 5.9+20140118-1ubuntu1 (1521)
initscripts (918): 2.88dsf-41ubuntu6 (1522)
libblkid1 (919): 2.20.1-5.1ubuntu20.3 (1523)
systemd-shim (920): 6-2bzr1 (1524)
libdatrie1 (1336): 0.2.8-1 (1525)
gfortran (1337): 4:4.8.2-1ubuntu6 (1526)
xinput (1338): 1.6.1-1 (1527)
python3-minimal (921): 3.4.0-0ubuntu2 (1529)
libss2 (922): 1.42.9-3ubuntu1 (1530)
libsemanage-common (923): 2.2-1 (1531)
liblwp-protocol-https-perl (1340): 6.04-2ubuntu0.1 (1532)
libgck-1-0 (924): 3.10.1-1 (1533)
libpam-modules-bin (925): 1.1.8-1ubuntu2 (1534)
command-not-found (926): 0.3ubuntu12 (1535)
libavahi-common-data (1341): 0.6.31-4ubuntu1 (1536)
findutils (927): 4.4.2-7 (1537)

libxcb-dri2-0-dev (1108): 1.10-2ubuntu1 (1157)
libc6-dev (1109): 2.19-0ubuntu6.4 (1158)
libssl0.9.8 (1486): 0.9.8o-7ubuntu3.2.14.04.1 (1764)
nfs-common (1110): 1:1.2.8-6ubuntu1.1 (1159)
bzip2 (752): 1.0.6-5 (1160)
libstdc++-4.8-dev (1111): 4.8.2-19ubuntu1 (1161)
vim-common (753): 2:7.4.052-1ubuntu3 (1162)
libhttp-negotiate-perl (1112): 6.00-2 (1163)
python-requests (1113): 2.2.1-1ubuntu0.1 (1164)
x11proto-scrnsaver-dev (1114): 1.2.2-1 (1165)
librtmp-dev (1115): 2.4+20121230.gitdf6c518-1 (1166)
openssl (754): 1.0.1f-1ubuntu2.7 (1167)
libglib2.0-data (755): 2.40.2-0ubuntu1 (1168)
libcloog-isl4 (1116): 0.18.2-1 (1169)
python-software-properties (1117): 0.92.37.2 (1170)
libgcrypt11-dev (1118): 1.5.3-2ubuntu4.1 (1171)
libquadmath0 (1119): 4.8.2-19ubuntu1 (1172)
xorg (1120): 1:7.7+1ubuntu8 (1173)
ncurses-bin (756): 5.9+20140118-1ubuntu1 (1174)
perl-base (757): 5.18.2-2ubuntu1 (1175)
sysv-rc (758): 2.88dsf-41ubuntu6 (1176)
libutempter0 (1121): 1.1.5-4build1 (1177)
libpython3-stdlib (759): 3.4.0-0ubuntu2 (1178)
libtimedate-perl (1122): 2.3000-1 (1179)
apt (760): 1.0.1ubuntu2.6 (1180)
python-ntdb (1123): 1.0-2ubuntu1 (1181)
sgml-base (761): 1.26+nmu4ubuntu1 (762)
liblapack3 (1124): 3.5.0-2ubuntu1 (1182)
libxdmcp-dev (1125): 1:1.1.1-1 (1183)
pciutils (762): 1:3.2.1-1ubuntu5 (1184)
libprocps3 (1126): 1:3.3.9-1ubuntu2 (1185)
libunistring0 (1127): 0.9.3-5ubuntu3 (1186)
libdbus-glib-1-2 (764): 0.100.2-1 (765)
manpages-dev (1128): 3.54-1ubuntu1 (1187)
ubuntu-standard (765): 1.325 (1188)
grub-common (766): 2.02~beta2-9ubuntu1 (1189)
libjson-c2 (767): 0.11-3ubuntu1.2 (1190)
accountsservice (768): 0.6.35-0ubuntu7.1 (1191)
libgssglue1 (1129): 0.4-2ubuntu1 (1192)
mtr-tiny (769): 0.85-2 (1193)
xserver-xorg-core (1130): 2:1.15.1-0ubuntu2.1 (1194)
libxcb1-dev (1131): 1.10-2ubuntu1 (1195)
libk5crypto3 (770): 1.12+dfsg-2ubuntu4.2 (1196)
libestr0 (1133): 0.1.9-0ubuntu2 (1198)
netbase (771): 5.2 (1199)
libmagic1 (772): 1:5.14-2ubuntu3.2 (1200)
libdrm2 (773): 2.4.52-1 (1201)
libgcr-3-common (774): 3.10.1-1 (1202)
libpython-stdlib (1134): 2.7.5-5ubuntu3 (1203)
libspice-server1 (1135): 0.12.4-0nocelt2 (1204)
whiptail (775): 0.52.15-2ubuntu5 (1205)
xserver-xorg-video-openchrome (1136): 1:0.3.3-1build1 (1206)
ntfs-3g (776): 1:2013.1.13AR.1-2ubuntu2 (1207)
libsepol1 (777): 2.2-1 (1208)
dmidecode (778): 2.12-2 (779)
apt-transport-https (779): 1.0.1ubuntu2.6 (1209)
libpam-modules (780): 1.1.8-1ubuntu2 (1210)
xserver-xorg-video-vmware (1137): 1:13.0.2-2ubuntu1 (1211)
tzdata (781): 2014i-0ubuntu0.14.04 (1212)
libtsan0 (1138): 4.8.2-19ubuntu1 (1213)
libbsd0 (928): 0.6.0-2ubuntu1 (1538)
libopenvg1-mesa (1342): 10.1.3-0ubuntu0.2 (1539)
libnfsidmap2 (1343): 0.25-5 (1540)
libxft2 (1344): 2.3.1-2 (1541)
libldb1 (1345): 1:1.1.16-1 (1542)
xz-utils (1346): 5.1.1alpha+20120614-2ubuntu2 (1543)
xserver-xorg-video-nouveau (1347): 1:1.0.10-1ubuntu2 (1544)
dosfstools (929): 3.0.26-1 (1545)
linux-image-3.13.0-40-generic (1348): 3.13.0-40.69 (1546)
r-recommended (1349): 3.1.2-1trusty0 (1547)
libxau-dev (1350): 1:1.0.8-1 (1548)
xfonts-encodings (1351): 1:1.0.4-1ubuntu1 (1549)
libx11-6 (930): 2:1.6.2-1ubuntu2 (1550)
libmirclient-dev (1352): 0.1.8+14.04.20140411-0ubuntu1 (1551)
grub-gfxpayload-lists (931): 0.6 (932)
ncurses-term (1353): 5.9+20140118-1ubuntu1 (1552)
e2fsprogs (932): 1.42.9-3ubuntu1 (1553)
libpaper1 (1354): 1.1.24+nmu2ubuntu3 (1554)
libmodule-pluggable-perl (1355): 5.1-1 (1555)
openssh-server (1356): 1:6.6p1-5hpn14v5~wrouesnel~trusty2 (1556)
liblzma5 (933): 5.1.1alpha+20120614-2ubuntu2 (1558)
rstudio-server (1490): 0.98.1091 (1769)
libnih1 (934): 1.0.3-4ubuntu25 (1559)
libxcb-dri3-0 (1358): 1.10-2ubuntu1 (1560)
libpng12-dev (1359): 1.2.50-1ubuntu2 (1561)
libkadm5clnt-mit9 (1360): 1.12+dfsg-2ubuntu4.2 (1562)
libcroco3 (1361): 0.6.8-2ubuntu1 (1563)
libavahi-common3 (1362): 0.6.31-4ubuntu1 (1564)
cpp (1363): 4:4.8.2-1ubuntu6 (1565)
libfile-desktopentry-perl (1364): 0.07-1 (1566)
libgnutls26 (935): 2.12.23-12ubuntu2.1 (1568)
libaudit1 (936): 1:2.3.2-2ubuntu1 (1569)
xserver-xorg-video-mga (1366): 1:1.6.3-1build1 (1570)
libdb5.3 (1368): 5.3.28-3ubuntu3 (1572)
python3-distupgrade (938): 1:0.220.5 (1573)
insserv (939): 1.14.0-5ubuntu2 (940)
gzip (940): 1.6-3ubuntu1 (1574)
libwww-perl (1369): 6.05-2 (1575)
dpkg (941): 1.17.5ubuntu5.3 (1576)
grub-pc (942): 2.02~beta2-9ubuntu1 (1577)
python-chardet (1370): 2.0.1-2build2 (1578)
libnet-ssleay-perl (1371): 1.58-1 (1579)
libpcre3 (943): 1:8.31-2ubuntu2 (1580)
libsasl2-modules-db (944): 2.1.25.dfsg1-17build1 (1581)
pkg-config (1372): 0.26-1ubuntu4 (1582)
iso-codes (945): 3.52-1 (1583)
libncursesw5 (946): 5.9+20140118-1ubuntu1 (1584)
libxml2-dev (1373): 2.9.1+dfsg1-3ubuntu4.4 (1585)
liblocale-gettext-perl (947): 1.05-7build3 (1586)
libssl-doc (1374): 1.0.1f-1ubuntu2.7 (1587)
ubuntu-keyring (948): 2012.05.19 (949)
x11proto-dri2-dev (1375): 2.8-2 (1588)
r-cran-lattice (1376): 0.20-29-1trusty0 (1589)
libglamor0 (1377): 0.6.0-0ubuntu4 (1590)
libwayland-client0 (1378): 1.4.0-1ubuntu1 (1591)

libudev1 (782): 204-5ubuntu20.9 (1214)
python (1139): 2.7.5-5ubuntu3 (1215)
r-cran-cluster (1140): 1.15.3-1trusty0 (1216)
libbz2-dev (1141): 1.0.6-5 (1217)
gettext (1142): 0.18.3.1-1ubuntu3 (1218)
libfontconfig1 (1143): 2.11.0-0ubuntu4.1 (1219)
xserver-xorg-input-wacom (1144): 1:0.23.0-0ubuntu2 (1220)
ifupdown (783): 0.7.47.2ubuntu4.1 (1221)
libsemanage1 (784): 2.2-1 (1222)
systemd-services (785): 204-5ubuntu20.9 (1223)
grub-pc-bin (786): 2.02~beta2-9ubuntu1 (1224)
libxxf86dga1 (1145): 2:1.1.4-1 (1225)
libxext6 (787): 2:1.3.2-1 (788)
libusb-1.0-0 (788): 2:1.0.17-1ubuntu2 (1226)
xterm (1146): 297-1ubuntu1 (1227)
xserver-xorg-video-radeon (1147): 1:7.3.0-1ubuntu3.1 (1228)
libdpkg-perl (1148): 1.17.5ubuntu5.3 (1229)
x11-session-utils (1149): 7.7+1 (1230)
lupin-casper (790): 0.55 (1231)
libxrender1 (1150): 1:0.9.8-1 (1232)
mountall (791): 2.53 (1233)
kmod (793): 15-0ubuntu6 (1234)
libc-dev-bin (1151): 2.19-0ubuntu6.4 (1235)
libtk8.6 (1152): 8.6.1-3ubuntu2 (1236)
dbus (794): 1.6.18-0ubuntu4.3 (1237)
xserver-xorg-video-qxl (1153): 0.1.1-0ubuntu3 (1238)
xserver-xorg-video-savage (1154): 1:2.3.7-2ubuntu2 (1239)
libapt-pkg4.12 (795): 1.0.1ubuntu2.6 (1240)
xserver-xorg-video-fbdev (1155): 1:0.4.4-1build1 (1241)
libxfont1 (1156): 1:1.4.7-1ubuntu0.1 (1242)
libthai0 (1157): 0.1.20-3 (1243)
diffutils (796): 1:3.3-1 (1244)
libxvmc1 (1158): 2:1.0.8-1ubuntu1 (1245)
libwww-robotrules-perl (1159): 6.01-1 (1246)
libasprintf-dev (1160): 0.18.3.1-1ubuntu3 (1247)
xserver-xorg-input-evdev (1161): 1:2.8.2-1ubuntu2 (1248)
libfile-mimeinfo-perl (1162): 0.22-1 (1249)
xserver-xorg-video-glamoregl (1163): 0.6.0-0ubuntu4 (1250)
linux-libc-dev (1164): 3.13.0-40.69 (1251)
x11proto-xext-dev (1165): 7.3.0-1 (1252)
resolvconf (797): 1.69ubuntu1.1 (1253)
libthai-data (1166): 0.1.20-3 (1254)
python3-gi (798): 3.12.0-1ubuntu1 (1255)
libpango-1.0-0 (1167): 1.36.3-1ubuntu1.1 (1256)
rpcbind (1168): 0.2.1-2ubuntu2.1 (1257)
libjpeg-dev (1169): 8c-2ubuntu8 (1258)
libxpm4 (1170): 1:3.5.10-1 (1259)
libkmod2 (799): 15-0ubuntu6 (1260)
libtdb1 (800): 1.2.12-1 (1261)
tar (801): 1.27.1-1 (1262)
libgettextpo0 (1171): 0.18.3.1-1ubuntu3 (1263)
file (802): 1:5.14-2ubuntu3.2 (1264)
linux-headers-3.13.0-24-generic (1172): 3.13.0-24.47 (1265)
gdebi-core (1487): 0.9.5.3ubuntu2 (1765)
libsigsegv2 (1173): 2.10-2 (1266)
gcc-4.8 (1174): 4.8.2-19ubuntu1 (1267)
liburi-perl (1175): 1.60-1 (1268)
xbitmaps (1379): 1.1.1-2 (1592)
build-essential (1380): 11.6ubuntu6 (1593)
libencode-locale-perl (1381): 1.03-1 (1594)
software-properties-common (1382): 0.92.37.2 (1595)
libxxf86vm-dev (1383): 1:1.1.3-1 (1596)
info (949): 5.2.0.dfsg.1-2 (1597)
liblzma-dev (1384): 5.1.1alpha+20120614-2ubuntu2 (1598)
libheimntlm0-heimdal (950): 1.6~git20131207+dfsg-1ubuntu1 (1599)
x11proto-core-dev (1385): 7.0.24-1 (1600)
crda (951): 1.1.2-1ubuntu2 (952)
x11proto-resource-dev (1386): 1.2.0-3 (1601)
dmsetup (952): 2:1.02.77-6ubuntu2 (1602)
samba-common-bin (953): 2:4.1.6+dfsg-1ubuntu2.14.04.3 (1603)
eject (954): 2.1.5+deb1+cvs20081104-13.1 (1604)
libcap-ng0 (955): 0.7.3-1ubuntu2 (1605)
libssl-dev (1387): 1.0.1f-1ubuntu2.7 (1606)
libmtdev1 (1388): 1.1.4-1ubuntu1 (1607)
irqbalance (956): 1.0.6-2ubuntu0.14.04.1 (1608)
gir1.2-glib-2.0 (957): 1.40.0-1ubuntu0.2 (1609)
gcc (1389): 4:4.8.2-1ubuntu6 (1610)
busybox-initramfs (958): 1:1.21.0-1ubuntu1 (1611)
libio-socket-ssl-perl (1390): 1.965-1ubuntu1 (1612)
libwayland-egl1-mesa (1391): 10.1.3-0ubuntu0.2 (1613)
dh-python (959): 1.20140128-1ubuntu8 (1614)
python-crypto (1392): 2.6.1-4build1 (1615)
python3 (960): 3.4.0-0ubuntu2 (1616)
libreadline6-dev (1393): 6.3-4ubuntu2 (1617)
xserver-xorg-video-s3 (1394): 1:0.6.5-0ubuntu4 (1618)
libx11-dev (1395): 2:1.6.2-1ubuntu2 (1619)
iputils-ping (961): 3:20121221-4ubuntu1.1 (1620)
r-cran-nnet (1396): 7.3-8-1trusty0 (1621)
libheimbase1-heimdal (962): 1.6~git20131207+dfsg-1ubuntu1 (1622)
xserver-xorg-video-trident (1397): 1:1.3.6-0ubuntu5 (1623)
iputils-tracepath (963): 3:20121221-4ubuntu1.1 (1624)
xtrans-dev (1398): 1.3.2-1 (1625)
libgettextpo-dev (1399): 0.18.3.1-1ubuntu3 (1626)
libgssapi3-heimdal (964): 1.6~git20131207+dfsg-1ubuntu1 (1627)
libasn1-8-heimdal (965): 1.6~git20131207+dfsg-1ubuntu1 (1628)
libbz2-1.0 (966): 1.0.6-5 (1629)
libxkbcommon0 (1400): 0.4.1-0ubuntu1 (1630)
libbind9-90 (967): 1:9.9.5.dfsg-3 (1631)
xserver-xorg-video-intel (1401): 2:2.99.910-0ubuntu1.3 (1632)
libdns100 (1402): 1:9.9.5.dfsg-3 (1633)
perl (1403): 5.18.2-2ubuntu1 (1634)
libpangoft2-1.0-0 (1404): 1.36.3-1ubuntu1.1 (1635)
automake (1405): 1:1.14.1-2ubuntu1 (1636)
r-base-core (1406): 3.1.2-1trusty0 (1637)
xserver-xorg-video-mach64 (1408): 6.9.4-1build1 (1639)
x11proto-damage-dev (1409): 1:1.2.1-2 (1640)
friendly-recovery (968): 0.2.25 (969)
libhtml-parser-perl (1410): 3.71-1build1 (1641)
libdrm-dev (1411): 2.4.52-1 (1642)

libcap2-bin (1176): 1:2.24-0ubuntu2 (1269)
libdrm-radeon1 (1177): 2.4.52-1 (1270)
libfont-afm-perl (1178): 1.20-1 (1271)
libhx509-5-heimdal (803): 1.6~git20131207+dfsg-1ubuntu1 (1272)
libxcb-sync-dev (1179): 1.10-2ubuntu1 (1273)
libmount1 (804): 2.20.1-5.1ubuntu20.3 (1274)
libhttp-cookies-perl (1180): 6.00-2 (1275)
libio-socket-inet6-perl (1181): 2.71-1 (1276)
libxcb-xfixes0-dev (1182): 1.10-2ubuntu1 (1277)
netcat-openbsd (806): 1.105-7ubuntu1 (807)
libfribidi0 (807): 0.19.6-1 (1278)
freeglut3 (1183): 2.8.1-1 (1279)
zlib1g (808): 1:1.2.8.dfsg-1ubuntu1 (809)
libroken18-heimdal (809): 1.6~git20131207+dfsg-1ubuntu1 (1280)
xserver-xorg (1184): 1:7.7+1ubuntu8 (1281)
libgfortran-4.8-dev (1185): 4.8.2-19ubuntu1 (1282)
libmailtools-perl (1186): 2.12-1 (1283)
libxrandr2 (1187): 2:1.4.2-1 (1284)
os-prober (810): 1.63ubuntu1 (1285)
libglu1-mesa (1188): 9.0.0-2 (1286)
libapt-inst1.5 (811): 1.0.1ubuntu2.6 (1287)
adduser (812): 3.113+nmu3ubuntu3 (1288)
libxcb1 (813): 1.10-2ubuntu1 (1289)
linux-image-extra-3.13.0-24-generic (1189): 3.13.0-24.47 (1290)
python3-debian (1488): 0.1.21+nmu2ubuntu2 (1766)
liblwp-mediatypes-perl (1190): 6.02-1 (1291)
xserver-xorg-video-all (1191): 1:7.7+1ubuntu8 (1292)
libgcc-4.8-dev (1192): 4.8.2-19ubuntu1 (1293)
locales (814): 2.13+git20120306-12.1 (1294)
libisl10 (1193): 0.12.2-1 (1295)
update-manager-core (815): 1:0.196.12 (1296)
python2.7 (1194): 2.7.6-8 (1297)
manpages (817): 3.54-1ubuntu1 (818)
ed (816): 1.9-2 (817)
python3-update-manager (818): 1:0.196.12 (1298)
dh-translations (1195): 121 (1299)
libsm-dev (1196): 2:1.2.1-2 (1300)
liblog-message-simple-perl (1197): 0.10-1 (1301)
pppconfig (820): 2.3.19ubuntu1 (821)
libgcc1 (821): 1:4.9.1-0ubuntu1 (1302)
libidn11-dev (1198): 1.28-1ubuntu2 (1303)
libck-connector0 (1199): 0.4.5-3.1ubuntu2 (1304)
libdebconfclient0 (1200): 0.187ubuntu1 (1305)
libxcomposite1 (1201): 1:0.4.4-1 (1306)
libice-dev (1202): 2:1.0.8-2 (1307)
libsys-hostname-long-perl (1203): 1.4-3 (1308)
x11proto-fixes-dev (1204): 1:5.0-2ubuntu2 (1309)
libklibc (822): 2.0.3-0ubuntu1 (1310)
libxcb-glx0-dev (1206): 1.10-2ubuntu1 (1312)
libhcrypto4-heimdal (823): 1.6~git20131207+dfsg-1ubuntu1 (1313)
x11proto-fonts-dev (1207): 2.1.2-1 (1314)
libxv1 (1208): 2:1.0.10-1 (1315)
libncurses5-dev (1209): 5.9+20140118-1ubuntu1 (1316)
libmirclient7 (1210): 0.1.8+14.04.20140411-0ubuntu1 (1317)
libgirepository-1.0-1 (824): 1.40.0-1ubuntu0.2 (1318)
libcgmanager0 (1211): 0.24-0ubuntu7 (1319)
mesa-common-dev (1212): 10.1.3-0ubuntu0.2 (1320)
python-talloc (1213): 2.1.0-1 (1321)
python3-six (1491): 1.5.2-1 (1770)
libxshmfence-dev (1412): 1.1-2 (1643)
comerr-dev (1413): 2.1-1.42.9-3ubuntu1 (1644)
libdrm-intel1 (1414): 2.4.52-1 (1645)
libtinfo5 (969): 5.9+20140118-1ubuntu1 (1646)
python-apt-common (970): 0.9.3.5 (1647)
xserver-xorg-dev (1415): 2:1.15.1-0ubuntu2.1 (1648)
libx11-xcb1 (1416): 2:1.6.2-1ubuntu2 (1649)
unzip (1417): 6.0-9ubuntu1 (1650)
zlib1g-dev (1418): 1:1.2.8.dfsg-1ubuntu1 (1651)
libpciaccess0 (1419): 0.13.2-1 (1652)
libpython3.4-stdlib (1420): 3.4.0-2ubuntu1 (1653)
libavahi-client3 (1421): 0.6.31-4ubuntu1 (1654)
libc-bin (971): 2.19-0ubuntu6.4 (1655)
linux-firmware (972): 1.127.10 (1656)
mime-support (973): 3.54ubuntu1 (974)
cifs-utils (974): 2:6.0-1ubuntu2 (975)
libldap-2.4-2 (975): 2.4.31-1+nmu2ubuntu8 (1657)
libxcb-xfixes0 (1422): 1.10-2ubuntu1 (1658)
r-base (1423): 3.1.2-1trusty0 (1659)
libkrb5-26-heimdal (976): 1.6~git20131207+dfsg-1ubuntu1 (1660)
xml-core (977): 0.13+nmu2 (978)
sysvinit-utils (978): 2.88dsf-41ubuntu6 (1661)
cron (979): 3.0pl1-124ubuntu2 (980)
xserver-xorg-input-all (1424): 1:7.7+1ubuntu8 (1662)
libkdb5-7 (1425): 1.12+dfsg-2ubuntu4.2 (1663)
libmail-sendmail-perl (1426): 0.79.16-1 (1664)
multiarch-support (980): 2.19-0ubuntu6.4 (1665)
fontconfig-config (1427): 2.11.0-0ubuntu4.1 (1666)
libedit2 (981): 3.1-20130712-2 (1667)
libxtst6 (1428): 2:1.2.2-1 (1668)
libxcb-dri3-dev (1429): 1.10-2ubuntu1 (1669)
libegl1-mesa (1430): 10.1.3-0ubuntu0.2 (1670)
nano (982): 2.2.6-1ubuntu1 (983)
r-cran-mass (1431): 7.3-35-1trusty0 (1671)
linux-generic (983): 3.13.0.24.29 (1672)
xserver-xorg-input-mouse (1432): 1:1.9.0-1build1 (1673)
libalgorithm-merge-perl (1433): 0.08-2 (1674)
libice6 (1434): 2:1.0.8-2 (1675)
pppoeconf (984): 1.20ubuntu1 (985)
libjson0 (985): 0.11-3ubuntu1.2 (1676)
python-apt (1435): 0.9.3.5 (1677)
libp11-kit0 (986): 0.20.2-2ubuntu2 (1678)
libpcrecpp0 (1436): 1:8.31-2ubuntu2 (1679)
libkrb5-dev (1437): 1.12+dfsg-2ubuntu4.2 (1680)
libxcb-shape0 (1438): 1.10-2ubuntu1 (1681)
libnih-dbus1 (987): 1.0.3-4ubuntu25 (1682)
libx11-data (988): 2:1.6.2-1ubuntu2 (1683)
time (989): 1.7-24 (990)
linux-headers-generic (990): 3.13.0.24.29 (1684)
libselinux1 (991): 2.2.2-1ubuntu0.1 (1685)
python3-software-properties (1439): 0.92.37.2 (1686)
libpython2.7-stdlib (1440): 2.7.6-8 (1687)
openssh-client (992): 1:6.6p1-5hpn14v5~wrouesnel~trusty2 (1688)
libaudit-common (993): 1:2.3.2-2ubuntu1 (1689)
libevent-2.0-5 (1441): 2.0.21-stable-1ubuntu1 (1690)
r-cran-matrix (1442): 1.1-4-1trusty0 (1691)
ntpdate (994): 1:4.2.6.p5+dfsg-3ubuntu2 (995)
xorg-sgml-doctools (1444): 1:1.11-1 (1693)
libtalloc2 (995): 2.1.0-1 (1694)

libsystemd-daemon0 (825): 204-5ubuntu20.9 (1322)
base-files (826): 7.2ubuntu5.1 (1323)
mount (827): 2.20.1-5.1ubuntu20.3 (1324)
libhtml-form-perl (1214): 6.03-1 (1325)
x11proto-video-dev (1215): 2.3.2-1 (1326)
libsqlite3-0 (828): 3.8.2-1ubuntu2 (1327)
libtext-iconv-perl (829): 1.7-5build2 (1328)
libpixman-1-dev (1216): 0.30.2-2ubuntu1 (1329)
libpam-systemd (830): 204-5ubuntu20.9 (1330)
r-cran-survival (1217): 2.37-7-1 (1331)
libpangocairo-1.0-0 (1218): 1.36.3-1ubuntu1.1 (1332)
libcups2 (1219): 1.7.2-0ubuntu1.2 (1333)
samba-libs (1220): 2:4.1.6+dfsg-1ubuntu2.14.04.3 (1334)
console-setup (831): 1.70ubuntu8 (1335)
python-pycurl (1221): 7.19.3-0ubuntu3 (1336)
xserver-xorg-input-synaptics (1222): 1.7.4-0ubuntu1 (1337)
xserver-xorg-video-vesa (1223): 1:2.3.3-1build1 (1338)
libreadline6 (832): 6.3-4ubuntu2 (1339)
libncurses5 (833): 5.9+20140118-1ubuntu1 (1340)
libplymouth2 (834): 0.8.8-0ubuntu17 (1341)
fonts-dejavu-core (1225): 2.34-1ubuntu1 (1343)
libprotobuf-dev (1226): 2.5.0-9ubuntu1 (1344)
rsync (835): 3.1.0-2ubuntu0.1 (1345)
libxcb-randr0-dev (1227): 1.10-2ubuntu1 (1346)
x11proto-xf86vidmode-dev (1228): 2.3.1-2 (1347)
libxcb-dri2-0 (1229): 1.10-2ubuntu1 (1348)
casper (836): 1.340 (1349)
libwind0-heimdal (837): 1.6~git20131207+dfsg-1ubuntu1 (1350)
gcc-4.9-base (1230): 4.9.1-0ubuntu1 (1351)
python3.4-minimal (1231): 3.4.0-2ubuntu1 (1352)
libldap2-dev (1232): 2.4.31-1+nmu2ubuntu8 (1353)
libp11-kit-dev (1233): 0.20.2-2ubuntu2 (1354)
lockfile-progs (838): 0.1.17 (839)
libegl1-mesa-drivers (1234): 10.1.3-0ubuntu0.2 (1355)
ppp (839): 2.4.5-5.1ubuntu2.1 (1356)
x11proto-render-dev (1235): 2:0.11.1-2 (1357)
libjpeg-turbo8 (1236): 1.3.0-0ubuntu2 (1358)
libkeyutils1 (840): 1.5.6-1 (1359)
libhttp-message-perl (1237): 6.06-1 (1360)
parted (841): 2.3-19ubuntu1 (1361)
logrotate (842): 3.8.7-1ubuntu1 (1362)
libexpat1 (843): 2.1.0-4ubuntu1 (1363)
python-ldb (1238): 1:1.1.16-1 (1364)
x11proto-xf86dri-dev (1239): 2.1.1-2 (1365)
libxt-dev (1240): 1:1.1.4-1 (1366)
libmirclientplatform-mesa (1241): 0.1.8+14.04.20140411-0ubuntu1 (1367)
libglu1-mesa-dev (1242): 9.0.0-2 (1368)
libntdb1 (1243): 1.0-2ubuntu1 (1369)
openssh-sftp-server (1244): 1:6.6p1-5hpn14v5~wrouesnel~trusty2 (1370)
python-scour (1245): 0.26-3build1 (1371)
python-samba (1246): 2:4.1.6+dfsg-1ubuntu2.14.04.3 (1372)
patch (1247): 2.7.1-4ubuntu1 (1373)
initramfs-tools-bin (844): 0.103ubuntu4.2 (1374)
linux-headers-3.13.0-40 (1248): 3.13.0-40.69 (1375)
intltool-debian (1249): 0.35.0+20060710.1 (1376)
libjbig0 (1250): 2.0-2ubuntu4.1 (1377)
libc6 (996): 2.19-0ubuntu6.4 (1695)
libsm6 (1445): 2:1.2.1-2 (1696)
x11proto-present-dev (1446): 1.0-1 (1697)
language-selector-common (997): 0.129.3 (1698)
grub2-common (998): 2.02~beta2-9ubuntu1 (1699)
libxxf86vm1 (1447): 1:1.1.3-1 (1700)
libpng12-0 (999): 1.2.50-1ubuntu2 (1701)
apt-utils (1000): 1.0.1ubuntu2.6 (1702)
udev (1001): 204-5ubuntu20.9 (1703)
libgnutls-dev (1448): 2.12.23-12ubuntu2.1 (1704)
libpixman-1-0 (1449): 0.30.2-2ubuntu1 (1705)
libnet-smtp-ssl-perl (1450): 1.01-3 (1706)
libnuma1 (1002): 2.0.9~rc5-1ubuntu2 (1707)
r-cran-kernsmooth (1451): 2.23-13-1trusty0 (1708)
libisc95 (1003): 1:9.9.5.dfsg-3 (1710)
iptables (1004): 1.4.21-1ubuntu1 (1711)
libpod-latex-perl (1453): 0.61-1 (1712)
bash (1005): 4.3-7ubuntu1.5 (1713)
linux-image-3.13.0-24-generic (1454): 3.13.0-24.47 (1714)
libpcre3-dev (1455): 1:8.31-2ubuntu2 (1715)
make (1456): 3.81-8.2ubuntu3 (1716)
libapparmor-perl (1006): 2.8.95~2430-0ubuntu5.1 (1717)
util-linux (1007): 2.20.1-5.1ubuntu20.3 (1718)
libgmp10 (1457): 2:5.1.3+dfsg-1ubuntu1 (1719)
libxcursor1 (1458): 1:1.1.14-1 (1720)
xfonts-utils (1459): 1:7.7+1 (1721)
libgbm1 (1460): 10.1.3-0ubuntu0.2 (1722)
po-debconf (1461): 1.0.16+nmu2ubuntu1 (1723)
libisccfg90 (1008): 1:9.9.5.dfsg-3 (1724)
ubuntu-minimal (1009): 1.325 (1725)
popularity-contest (1010): 1.57ubuntu1 (1011)
apparmor (1011): 2.8.95~2430-0ubuntu5.1 (1726)
m4 (1462): 1.4.17-2ubuntu1 (1727)
libfontenc1 (1463): 1:1.1.2-1 (1728)
rsyslog (1012): 7.4.4-1ubuntu2.3 (1729)
cpio (1013): 2.11+dfsg-1ubuntu1 (1014)
powermgmt-base (1014): 1.31build1 (1015)
libstdc++6 (1015): 4.8.2-19ubuntu1 (1730)
x11-xfs-utils (1464): 7.7+1 (1731)
zip (1465): 3.0-8 (1732)
r-cran-mgcv (1466): 1.8-3-1trusty0 (1733)
libmpfr4 (1467): 3.1.2-1 (1734)
libhtml-tagset-perl (1468): 3.20-2 (1735)
isc-dhcp-client (1016): 4.2.4-7ubuntu12 (1736)
ca-certificates (1017): 20130906ubuntu2 (1737)
strace (1018): 4.8-1ubuntu5 (1738)
r-doc-html (1469): 3.1.2-1trusty0 (1739)
vim-tiny (1019): 2:7.4.052-1ubuntu3 (1740)
xorg-docs-core (1470): 1:1.7-1 (1741)
xserver-xorg-video-ati (1471): 1:7.3.0-1ubuntu3.1 (1742)
xserver-xorg-video-siliconmotion (1472): 1:1.7.7-2build1 (1743)
binutils (1473): 2.24-5ubuntu3 (1744)
libblas3 (1474): 1.2.20110419-7 (1745)
libblas-dev (1475): 1.2.20110419-7 (1746)
ssh-import-id (1476): 3.21-0ubuntu1 (1747)
libx11-doc (1477): 2:1.6.2-1ubuntu2 (1748)
less (1020): 458-2 (1021)
libapparmor1 (1021): 2.8.95~2430-0ubuntu5.1 (1749)
libglib2.0-0 (1022): 2.40.2-0ubuntu1 (1750)

usbutils (845): 1:007-2ubuntu1 (1378)
libdevmapper1.02.1 (846): 2:1.02.77-6ubuntu2 (1379)
libxkbfile-dev (1251): 1:1.0.8-1 (1380)
libxi6 (1252): 2:1.7.1.901-1ubuntu1 (1381)
libxatracker2 (1253): 10.1.3-0ubuntu0.2 (1382)
xinit (1254): 1.3.2-1 (1383)
libattr1 (847): 1:2.4.47-1ubuntu1 (1384)
libgcr-base-3-1 (848): 3.10.1-1 (1385)
xfonts-base (1255): 1:1.0.3 (1386)
xserver-xorg-video-sisusb (1256): 1:0.9.6-2build1 (1387)
krb5-locales (849): 1.12+dfsg-2ubuntu4.2 (1388)
freeglut3-dev (1257): 2.8.1-1 (1389)
libmpc3 (1258): 1.0.1-1ubuntu1 (1390)
libpcap0.8 (850): 1.5.3-2 (1391)
libhtml-tree-perl (1259): 5.03-1 (1392)
sed (851): 4.2.2-4ubuntu1 (1393)
gfortran-4.8 (1260): 4.8.2-19ubuntu1 (1394)
libidn11 (852): 1.28-1ubuntu2 (1395)
hdparm (853): 9.43-1ubuntu3 (1396)
r-cran-boot (1261): 1.3-13-1trusty0 (1397)
python2.7-minimal (1262): 2.7.6-8 (1398)
tcpdump (854): 4.5.1-2ubuntu1.1 (1399)
libxcb-render0 (1263): 1.10-2ubuntu1 (1400)
x11proto-randr-dev (1264): 1.4.0+git20120101.is.really.1.4.0-0ubuntu1 (1401)
x11proto-kb-dev (1265): 1.0.6-2 (1402)
klibc-utils (859): 2.0.3-0ubuntu1 (1414)
keyutils (860): 1.5.6-1 (1415)
intltool (1478): 0.50.2-2 (1751)
libgomp1 (1479): 4.8.2-19ubuntu1 (1752)
libtext-charwidth-perl (1023): 0.04-7build3 (1753)
libxt6 (1480): 1:1.1.4-1 (1754)
readline-common (1024): 6.3-4ubuntu2 (1755)
libcap2 (1025): 1:2.24-0ubuntu2 (1756)
libitm1 (1481): 4.8.2-19ubuntu1 (1757)
x11proto-dri3-dev (1482): 1.0-1 (1758)
libnfnetlink0 (1026): 1.0.1-2 (1027)
libwrap0 (1483): 7.6.q-25 (1759)
libkrb5-3 (1027): 1.12+dfsg-2ubuntu4.2 (1760)
lsof (1028): 4.86+dfsg-1ubuntu2 (1029)
libpython2.7-minimal (1484): 2.7.6-8 (1761)
liblockfile1 (1029): 1.09-6ubuntu1 (1762)
libxau6 (1030): 1:1.0.8-1 (1031)
r-cran-spatial (1266): 7.3-7-1trusty0 (1403)
liblockfile-bin (855): 1.09-6ubuntu1 (1404)
xserver-xorg-video-tdfx (1267): 1:1.4.5-1build1 (1405)
libxcb-present-dev (1268): 1.10-2ubuntu1 (1406)
libxcb-glx0 (1269): 1.10-2ubuntu1 (1407)
libgpg-error-dev (1270): 1.12-0.2ubuntu1 (1408)
lshw (856): 02.16-2ubuntu1.2 (1409)
libglapi-mesa (1271): 10.1.3-0ubuntu0.2 (1410)
libtext-soundex-perl (1272): 3.4-1build1 (1411)
libtiff5 (1273): 4.0.3-7ubuntu0.1 (1412)
librtmp0 (857): 2.4+20121230.gitdf6c518-1 (858)

12.5 Appendix E: Installing and Running Cumulus

The following section provides instructions for compiling, installing and running Cumulus. This documentation details the process for a CentOS 7 Linux-based system; these should be replaced with equivalent commands for other operating systems. The instructions here are recreated from the instructions in the Cumulus repository. For the latest versions, please refer to the instructions in the Cumulus repository at https://github.com/stembio/cumulus.

12.5.1 Installing Cumulus

The latest version of Cumulus, CodeLab or StembioDrive can be downloaded from the releases folder of the relevant repositories for example, see https://github.com/stembio/cumulus/releases for the latest Cumulus release. To install Cumulus the following steps must be carried out:

1) Install Java 8, Apache Tomcat 7, Squashfs (version 4.2 tested) and MySQL (version 5.7 tested) or the open source equivalent MariaDB (version 10.1 tested)

```
sudo yum install java-1.8.0-openjdk tomcat mariadb-server squashfs-tools
```

2) Install VirtualBox as detailed on the CentOS wiki - https://wiki.centos.org/HowTos/Virtualization/VirtualBox

3) Start the Tomcat and MySQL or equivalent service

```
sudo service tomcat7 start
sudo systemctl start mariadb
```

4) Harden the database and create users and passwords as required

```
sudo mysql_secure_installation
```

5) Open the WAR file and configure the ApplicationContext.xml. Minimally, set some usernames and passwords for the User and Admin Role as shown in the following example taken from the Spring security documentation:

```
<authentication-manager>
    <authentication-provider>
      <user-service>
        <user name="jimi" password="jimispassword" authorities="ROLE_USER, ROLE_ADMIN" />
        <user name="bob" password="bobspassword" authorities="ROLE_USER" />
      </user-service>
    </authentication-provider>
</authentication-manager>
```

6) Create a Cumulus directory in the root of your filesystem. Edit and copy in the cumulus.conf file from the root of the project and edit it. The configuration options for this are extensive and detailed in the project repository. Sensible defaults are included. Minimally fill in database_user and database_password created in step 3.

7) Copy the Cumulus WAR into the Tomcat applications folder. Launch a web browser and navigate to the URL http://localhost:8080/cumulus. Log in with the username and password specified in step 5.

12.5.2 Compilation from source

1) Install the Java Development Kit (JDK), Apache Maven (version 3.3.3 tested) and a client for the source code management system git (version 2.4 tested).

```
sudo yum install java-1.8.0-openjdk-devel apache-maven git
```

2) Checkout and build each repository in order. This will install the code into your Apache Maven local cache.

```
git clone https://github.com/stembio/<repository name>
cd <repository name>
mvn clean install
cd                                                                    -
```

Where <repository name> is stembio-core, useradmin-client, x11applet, stembio-drive cumulus and codelab.

Within the stembio-drive, cumulus and codelab repositories, a WAR file will be created in the target folder. This is the compiled version of each application.