The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) ________________________________

Nimbus Cloud Architecture for Digital Histopathology
Bhupinder Singh Soorma
Computing
2009 / 2010
Summary

This project investigates the use of Distributed Systems specifically using a Grid and Cloud architecture. The basis of this project is a re-engineering development of a histopathology application and how such a system can be modified sufficiently to utilise the native environment it is re-deployed onto, all be it on a Cloud Virtual Infrastructure Manager called Nimbus.

Primarily this project is focuses on taking an existing histopathology application, deployed as simple client server composition, then using a more virtual dynamically scalable infrastructure, namely a Cloud, to utilise and re-deploy it resource demanding configuration. A series of throw away prototyping is the main fundamental bridging new solutions, ideas and tying together the software development life cycle adopted for the entire multifaceted processes incorporated within this project.
Acknowledgements

I would like to thank Karim Djemame for his support guidance throughout the duration of this project and his priceless knowledge and experience in the weekly scheduled meetings.

I would like to thank Derek Magee for his time to read and mark this project and providing valuable feedback on the mid-term project report and subsequent progress meeting.

I would like to thank Dr. Darren Treanor a pathologist at St James University Hospital and user of the histopathology application for proving critical information on the various techniques and explaining the application issues and potential of the application.

I would like to thank Django Armstrong for his helpful technical knowledge at the supervision meetings.
# Table of Contents

**SUMMARY** ........................................................................................................... I

**ACKNOWLEDGEMENTS** ....................................................................................... II

**TABLE OF CONTENTS** ........................................................................................ III

**CHAPTER 1: INTRODUCTION** .............................................................................. 1

1.1 AIM ..................................................................................................................... 1
1.2 OBJECTIVES ..................................................................................................... 1
1.3 MINIMUM REQUIREMENTS ............................................................................. 1
1.4 ENHANCEMENTS ..................................................................................... 1
1.5 SCHEDULE .................................................................................................... 2
1.6 SUMMARY .................................................................................................... 2

**CHAPTER 2: BACKGROUND READING & RESEARCH** ........................................ 3

2.1 INTRODUCTION ............................................................................................. 3
2.2 VIRTUALIZATION AND VISUALISATION .............................................. 4
2.3 GRIDS ........................................................................................................... 5
   2.3.1 Architecture ....................................................................................... 6
   2.3.2 Application ....................................................................................... 7
   2.3.3 Infrastructure .................................................................................. 9
   2.3.4 Middleware .................................................................................... 11
2.4 CLOUDS ......................................................................................................... 13
   2.4.1 Architecture .................................................................................. 13
   2.4.2 Application .................................................................................. 15
   2.4.3 Infrastructure ............................................................................. 17
   2.4.4 Middleware ................................................................................ 18
2.5 HISTOPATHOLOGY APPLICATION ......................................................... 19
2.6 CURRENT APPLICATION STATE ............................................................. 20
2.7 BENEFITS OF MOVING TO A CLOUD VIRTUAL INFRASTRUCTURE MANAGER – NIMBUS .................................................................................................................................................................................. 22
   2.7.1 Why Choose Nimbus ..................................................................... 22
2.8 METHODOLOGIES ..................................................................................... 23
   2.8.1 Software Development ................................................................. 23
   2.8.2 Cloud TestBed ............................................................................. 24
2.9 SUMMARY .................................................................................................... 25

**CHAPTER 3: DESIGN** .......................................................................................... 26

3.1 INTRODUCTION ............................................................................................. 26
3.2 METHODOLOGY ............................................................................................ 26
3.3 PROJECT MANAGEMENT ........................................................................... 26
   3.3.1 Original Schedule ......................................................................... 26
   3.3.2 Revised Schedule .......................................................................... 27
3.4 APPLICATION DESIGN ............................................................................... 28
3.5 APPLICATION IMPLEMENTATION ......................................................... 29
3.6 APPLICATION TESTING ............................................................................ 29
3.7 CURRENT PATHOLOGY APPLICATION DESIGN AND SETUP .......... 30
   3.7.1 Technical overview – Invocation procedure of the current application .................................................................................................................. 30
3.8 PROPOSED SYSTEM RE-DEPLOYMENT ONTO NIMBUS MIDDLEWARE .................................................................................................................................................................................. 31
3.9 DESIGNING THE PROTOTYPE ................................................................. 31
   3.9.1 Java Web Service ........................................................................... 32
   3.9.2 Java Web Service Using GRAM .................................................. 34
      3.9.2.1 Globus Resource Allocation Manager (GRAM) .................. 34
      3.9.2.2 Designing using GRAM ................................................... 34
3.10 DEPLOYMENT PLAN OF APPLICATION ON CLOUD VIRTUAL IMAGE/MACHINE .................................................................................................................................................................................. 37
3.11 SUMMARY .................................................................................................. 38
APPENDIX H: DEVELOPMENT SOURCE CODE...........................................................................88

PATHOCALL.JAVA..................................................................................................................88
PATHOLOGY.JAVA....................................................................................................................89
CLIENT.JAVA............................................................................................................................90
NUCDETSERVER.RSL FILE .....................................................................................................91
SAMPLE-WORKSPACE.XML SOURCE CODE ............................................................................92
MATHSERVICE.JAVA (PATHOLOGYSERVICE.JAVA) SOURCE CODE ...........................................93
Chapter 1: Introduction

1.1 Aim
To re-deploy a histopathology application which analyses and views digital tissue samples currently deployed on a client server model, via a cloud computing virtual infrastructure manager known as Nimbus. The cloud architecture should simplify and improve reliability, usability and efficiency using distributed computing resource techniques to aid in a transparent deployment for the end user, focusing on providing functionality at the Cloud infrastructure level.

1.2 Objectives
The main objective of the project is to investigate and re-deployment of a data intensive pathology application suitably utilising its distributed architectural computing environment. Creating a proof of concept for the application able to re-deploy onto a cloud computing architecture is the main basis to the project. Necessary steps to achieve the aim of the project include:

I. Investigate and understand Grid and Cloud frameworks covering architectural, infrastructure, application and middleware as apart of the literature review process.
II. Comprehend and practically train on a Grid Globus GT4 middleware and Cloud middleware Nimbus.
III. Formulate an appropriate development process to re-deploy application.
IV. Develop the solution and deploy the virtual image onto a cloud test bed.
V. Evaluate the effectiveness of cloud technology within integrated pathology application in relation to the aims of the project.

1.3 Minimum Requirements
The minimum requirements are:

I. Design suitable pathology application for implementation on a cloud.
II. Develop the product on Nimbus middleware.

1.4 Enhancements
Possible extensions are:

I. Data management
Create seamless connection across different administrative domains (University/NHS Hospital users) in the cloud. With application predominantly looking at sensitive real patient data, it will be necessary to integrate policy constraints (e.g. confidential compliance with patient data records). Other more formal factors of data management will have to accurately ensure optimisation in the case of Integrity/synchronization is met.
II. **Parallelization**

Performance issues of computation time taken to reconstruct 3D volumes from 2D images, which require significant CPU power. This could be addressed through the virtual machines present in the cloud architecture distributing and able to perform execution of tasks at the concurrently.

1.5 **Schedule**

Appendix D illustrates the initial plan of the project. Each section of the Gantt chart indicates a milestone, with a further breakdown of critical sections. The milestones and projected completion dates are:

I. Preliminary investigation (18/01/2010 – 26-02-2010)
   a. Grids
   b. Clouds

II. Pathology Application (01/03/2010 – 04/05/2010)
   a. Design (01/03/2010 – 22/03/2010)
   b. Implementation (11/03/2010 – 04/05/2010)

III. Evaluation (22/04/2010 – 06/05/2010)

IV. Submission (12/05/2010)

1.6 **Summary**

With defined aims, objectives, minimum requirements, and possible enhancements, Chapter 2 examines relevant literature to achieve the necessary milestones as presented in the initial project plan.
Chapter 2: Background Reading & Research

2.1 Introduction

This project as stated in the objectives aims to re-deploy a histopathology application onto a cloud virtual infrastructure manager, specifically Nimbus. Throughout the preliminary research, this project sufficiently aims to cover Grids, Clouds and the infrastructures, architectures, applications, middleware’s, and virtualization techniques that are relevant in re-implementing parts of the system application.

The application supports pathologists to perform digital image analysis of 2D tissue cell slices taken from patients. Aiding pathologist to examine these tissue cells and performing an accurate diagnosis, it compiles the 2D cell samples into a single 3D model. This reconstruction method gives a visual representation of the cells multiple layers in sample space, which provides a better context linked to the physical locality and construction per each tissue sample set. This intricate applications ability is hindered merely by the current deployment techniques of a traditional client server model. With high level resource demands for retrieving each tissue sample potentially gigabytes in size, the application faces a reduction in performance, reliability, and efficiency imposed on the servers inability to be scalable.

To tackle these predicaments in the current implementation I will prepare to re-deploy even re-engineer a nucleus detection application onto a more scalable orientated paradigm infrastructure of a cloud. This application is one of many counterpart applications that assist to diagnose and identify nuclei in the image slice taken from patients, providing a proof of concept. To achieve this research approach I will endeavour to uncover concurrently, similarities and differences linking clouds and grids, both of which are still relatively new concepts. Exploring Girds, a forerunner to Clouds will facilitate in providing a useful insight into distributed technology interactions, attained through a variety of virtualization, data management techniques in resources used between both environments.

The context of the project aims to explore the nature of Virtualization, Grid Computing, Cloud Computing and re-deployment of the pathology application. These four aspects will form the core foundation to analyse various objectives critical or unforeseen for eventually re-deploying an operational application onto the Nimbus middleware. The plan used for the literature review can be found in Appendix B.
2.2 Virtualization and Visualisation

Virtualization as with many technologies provide an abstract infrastructure mimicking the underlying physical system, but with advances in combining virtual and physical interaction mediums some technologies such as Grids and Clouds use virtual environments/interfaces to unify both environments taking a foothold on utilizing computer resources. With Grids being a precursor development to clouds, the virtual level of interaction provided is a virtual object level which forms the basis of Virtual Organisations (VO) commonly seen. The notion of Virtual Organisations is a community shared resource network; with each institute having its own administrative policies on resource sharing, preventing un-authorised system access. An issue of interaction and utilising these shared resources filters down to the basic communication middleware of invoking virtual objects. Whereby these shared networked machines use virtual objects to procure the use of physical resources, dealing with difficulties foreseen in heterogeneity software and hardware through specially created middleware (Shinder & Vanover, 2008).

The use of virtualization presents many potential advantages and disadvantages ranging from basic low level boundary communication in virtual data management to high level interactions whilst retaining a transparent configuration (Shinder & Vanover, 2008). Clouds have revolutionized the use of virtualization, taking the basic level object interaction to a whole new plane. With Clouds evolving from Grids, the similarity of establishing and interrelating between physical resources is similar in setup but differs with the formalism of virtual machines or a “VMReady state” (The_IDC_Circle, 2008). Virtual machines are created via an almost identical approach seen first in server virtualization, where specialist software acts as a medium enabling the administrator of the system to take a physical sever and generate multiple virtual machines. This commercial arrangement contains the ability to gain a scalable, efficient framework for deployment of resources both vertically and horizontally across the system, consolidating use of further equipment. This virtualization paradigm is central to a cloud computing infrastructure, whereby cloud Virtual Infrastructure Managers (VIM) such as Nimbus uses a similar approach to obtain and fully utilise underlying networked resources. This virtual level of interaction and flexibility with complete resources usage not only enables the user to avoid un-necessary compatibility issues dealt through the middleware setup to the physical resources, but permits the user to replicate their environment setup customizing it to their preference. The user on a cloud virtual environment has the ability to tailor produce their own particular virtual image complete with their user parameters, even allowing, avoiding issues of vendor incompatibility (Shinder & Vanover, 2008) as the virtual infrastructure manager deals with heterogeneous system interactions within the cloud service.
Using virtualization comes within reach of achieving a system wide consolidation of physical hardware, or even completely utilising somewhat idle system resources therefore providing reduction in system operations costs (The_IDC_Circle, 2008). The importance of virtualization as a whole exposes relevance to the wider domain, but more specifically provides prominent cause applicable for use contained by Grids and more so in Cloud infrastructures. Understanding the physical interaction measures is one factor the project faces from deploying a service, protocols and standards, but in addition to these, grasping the virtual operation procedures to the underlying physical framework exposes a new set of issues on grids however more importantly in clouds.

2.3 Grids

Sharing resources is not a new concept in computing and is commonly used in every day technologies, from storage on servers or using network resources whether in a public/private entity. Prominent to this type of system setup is the restrictive amount of core infrastructure immediately available when high demand situations cause bottlenecks in communication.

Historically Grids started off as the main framework, compacting distributed heterogeneous technologies to establish a shared resource paradigm. The underpinning objective is to create supercomputer resource equivalence, through sharing and merging pooled computer resources. Creating a computing infrastructure similar if not better than a supercomputer is achieved via combining all connected institutes and their resources, such that the computer network becomes a powerful computing infrastructure (Foster, The grid: a new infrastructure for 21st Century science, 2003). Constructing this distributed physical lattice framework and understanding it naturally provides the ground work to comprehend clouds, as Grids are the predecessor from which clouds have engineered from.

Grid computing is another resource sharing development but substantially larger in scale than most common schemes and mainly seen in academic communities. Their construction however differs from the norm via linking distributed network computer clusters in conjunction with each domain operating special grid network software. This move towards distributed resource sharing aims to tap into inactive or even redundant computer resources. To contend with heterogeneous entities exposed throughout the grid system it falls to the middleware to bridge the gap and allow different machines to run applications or processes across the entire network. The grid initiative is to provide limitless computing power, a similar but slightly varied approach used in the evolution model of cloud computing. The importance of understanding grids, a forerunner to clouds, offers the ability to appreciate and research its architectural contribution and paradigm shift towards clouds. The fact that many grid applications and systems in use today link to cloud setups, provides an invaluable aspect to explore and comprehend, thus enabling the project to transfer skill sets such as the training on Globus
GT4 which has evolved into a cloud virtual infrastructure manager named Nimbus (Keahey & Freeman, 2008).

This section will endeavour to present a break down of the grid structural abilities split into four subsections: Architecture, Application, Infrastructure and Middleware.

### 2.3.1 Architecture

At the most basic level grid computing is designed to make the most use of distributed computer resources, applications and facilities. A grid computing system is most often a collection of heterogeneous resources ranging from a single machine to large scale institutes and their cluster of machines forming the interconnected backbone to this shared community network. To tap into this distributed resource network requires no special proprietary software, tools or even physical hardware often named as the fabric layer in grids. An important concept for providing all organisations aiming to utilise the limitless computing power, is standardization of protocols. Protocols and standards such as the Open Grid Services Architecture (OGSA) (Foster, Kesselman, Nick, & Tuecke, 2002), established by the Open Grid Forum (OGF) is assisting horizontal scaling of resources in the fabric layer and userdevelopers to have interoperability to operate unused remote resources successfully.

The issue of interoperability is a key piece which grid computing must adhere to; ensuring sharing relationships are dynamically maintained to changes in resource languages, remote invocations from different platforms connected to the grid service architectures. The OGSA framework has supported the push towards an “hourglass model” attempting to provide a consistent level of accessibility throughout the grid domain (Foster, Kesselman, & Tuecke, 2001). Central to the hourglass model is the middleware with the core services/protocols that provide the basic infrastructure and greatest

![Layered diagram of OGSA, GT4, WSRF, and Web Services](http://gdp.globus.org/gt4-tutorial/multiplehtml/ch01s01.html)
interoperability to the high level global service applications running at the top of the model, and also
supporting the large resource concentration at the bottom of the model (e.g. Local operating systems,
physical hardware, etc…).

Assisting and exploiting the computing resources across the grid comes in form of using the Globus
middleware toolkit, the middleware performs a layer of abstraction between the physical resources
creating a virtualization method of interaction (Foster, Kesselman, & Tuecke, 2001). Using
virtualization within grids presents a medium to which forms the concepts of Virtual Organisations
(VO), who are able to access the distributed networked resources available to them (Foster,
Kesselman, & Tuecke, 2001). The concept of VO defines a boundary that encapsulates the
organisations resources exposed on the grid, whether solely providing particular resources or
collaborative institutes across its domain. The fundamental nature of using a virtual level interaction
provides the grid to maintain connectivity and utmost accessibility to the distributed resources
registered throughout the grid computing network.

Although at the forefront to grids is resource exploitation, many VO have initialised a service
orientated paradigm. Whereby an application front-end hosts the service interface embedded within it,
effectively the application is a conduit to the base level services providing an uncomplicated input
output parameter solution to the incoming requests. The Open Grid Service Architecture (OGSA)
built upon the concept technologies of distributed grid and web service communication. The
fundamental principal is to create a mechanism for the standards in establishing naming and
discovering transient services, whilst retaining a paramount nature of transparency (in terms of
location, protocols, platforms) to the end user. This form of interaction presents a uniform
compatibility issue, which is addressed by the use of the Web Service Description Language (WSDL).
The WSDL provides a reusable interface design for distributed systems across the architecture to
access the service binding in a language neutral design with XML syntax. Whether the grid/web
service is written in C/C++, if the service types can be defined in a XML format, the service can be
extended and embedded into a new service written in java. The use of WSDL PortType defines the
service interface and because of its language neutral construction it maintains service level code
connections, providing better maintenance possibilities. (Foster, Kesselman, Nick, & Tuecke, 2002).

2.3.2 Application

Active grid projects are pushing the boundaries for application demands, whether requiring extra
memory, CPU processing power or temporary/permanent data storage not achievable on single
computer installation. Applications simulating or running large scale problems surpass the linear
resource qualities limited on single machine. Conventionally it would fall to a supercomputer to host
the application and provide the powerful infrastructure to contend with the application, but considering the expense to purchase and implement the architecture it would easily go beyond many institutions budgeting requirements. Spreading the demand and using a shared community network rivalling and exceeding supercomputer facilities whilst retaining affordability, flexibility and enhancing performance is why applications are migrating to grids.

Many research organisations and academic projects are accounting for grid systems operations, one of the largest being the Large Hadron Collider (LHC) a research project based at CERN in Geneva, Switzerland (CERN, 2010). This is a prominent data intensive application featuring high emphasis to analyze and store high intensity raw data, its presented through the supporting grid infrastructure at an “embryonic” stage of development examined in (Bunn & Newman, 2003). This research designed to smash particles smaller than ones in the universe itself and to provide evidence in support of the Higgs boson particle, in a fully operational state using the ATLAS physics detector is set to produce 700 megabytes per second (MB/s) and a up-to 15 petabytes according to (CERN, 2010). Successively comprehending and analysing the data influx produced through the LHC, the Enabling Grids for E-sciencE (EGEE) infrastructure resources takes the brunt of the work necessary in decomposing and digesting the enormous dataflow produced. The distribution of the dataflow sees the contribution of over 240 institutions collaborating to the LHC, stripping down the function of analysis between each resource specific service supplied via the many collaborating institutions (Bunn & Newman, 2003). Not only does the EGEE facilitate the needs for immense projects like the LHC, but it also hosts other collaborative applications services such as simulation and visualisation of surgical problems (Bio med), planning environmental factors (i.e. flooding crisis scenarios), data mining and global pollution models. Optimal ways to harness and solve large scale conventional data intensive algorithms, plotting out problems scenarios whilst retaining scalability and the complexity issues that arise is the forefront of these applications (Bunn & Newman, 2003). These global grid infrastructures are aiding grid applications explore autonomous scenarios using gird enabled analysis compared to manual observational approach. On going research is seeing potential viability of maximising, utilizing resource usage within the scientific domains and their collaborative partners improve large scale database management issues predominantly in areas of High Energy Physics (HEP) environments. Although not at the same level as seen in LHC project the histopathology application is a similar data orientated application, placing high demands on the physical infrastructure it operates under.

Another grid application smaller in scale, but focused on real time large scale data storage and analysis similar to the LHC, is the Distributed Aircraft Maintenance Environment (DAME) project (Hey & Trefethen, 2003). Involving a distributed array of academic and industry entities it sees the use of White Rose Computational Grid (WRCG) (DAME: Project Overview, 2010) with grid and web services (based on the OGSA standards) combine geographically distributed resources using Globus
GT3. It is orientated centrally on diagnostic and maintenance of aircraft engine data. Deploying a grid middleware to be data orientated rather than computationally focused, using self archiving initiatives to help comprehend with a localised daily input of real time data recorded from the many airport station monitors of engine data, accumulation becomes an issue that DAME strive to solve. This is achieved by deploying recognition algorithms or decision systems, identifying irregularities in the diagnostic phase (DAME: Project Overview, 2010) (Hey & Trefethen, 2003). Binding and surveying this data distributed throughout the connected domains is the central framework enabling the diagnostic setup to sufficiently handle the data influx, driving the use of persistent data storage and using open data publishing and archiving data digitally with sufficient metadata information improving data accessibility (Hey & Trefethen, 2003).

As grid technologies progressively mature, there is a need for more applications to become grid compliant. Helping to fill this missing functionality has led to the Grid Application Toolkit (GAT), developed at (GridLab, 2010). The GAT uses adaptors to handle types of complexity and grid middleware applications; it also presents a unified high level programming interface to the grid infrastructure, serving the need of grid application development. Taking advantage of classifying and enabling grid applications (Allen, Goodale, Russell, Seidel, & Shalf, 2003) emphasize the use of grid on demand, infrastructure, bridging applications not grid enabled through GAT-API’s, with potential re-configurations on applications to become grid enabled, a similar approach that will be taken to re-implement the histopathology application onto a cloud. More importantly the authors (Allen, Goodale, Russell, Seidel, & Shalf, 2003) outline specific Cactus “Framework for scientific numerical simulations” (GridLab, 2010) assisting in testing grid-enabled applications. This gives the ability to harness grid technologies and the significance of using a dynamic grid framework, whilst using compatible OGSA standards. Some of the data management, but more so the skills/knowledge communicational channels connecting across the different domain in the DAME project could assist in providing a better middleware interaction model needed in the Grid Application Toolkit (GAT) helping elevate some of the complexities seen.

2.3.3 Infrastructure

It is a prevalent architecture for major academic and industrial institutes, bridging resource availability through versatile network infrastructures. Formidably each grid infrastructure serve the need in connecting institutes, either on the basis of exposing use of powerful resources or facilitating access to the vast quantity of specialist resource services deployed. One of many infrastructures is the TeraGrid situated in the United States, the European Enabling Grids for E-sciencE (EGEE), the UK National Grid Service (UK NGS) and smaller regional based White Rose Grid (WRG).
The TeraGrid, a US situated grid infrastructure, is the world’s largest grid, which encompasses a substantial range of universities across the US, creating a cyber infrastructure geared towards open scientific research projects (TeraGrid, 2010). It supports combinational analysis of multi-source large scale data volumes through the joint correlation of large scientific data archives, high performance computing, data sharing covering more than 100 different research disciplines from collaborative research institutes, all via the operational high speed network (Reed, 2003). Through the oracle databases the TeraGrid hosts the ability to provide on demand petabytes of processing capabilities (TeraGrid, 2010). Helping to maintain an open-ended research endeavor for all participating institutes across the five domains, the infrastructure uses the Globus middleware toolkit, not only does this provide an OGSA standard across the domains but enables the partners to maintain a almost flawless interaction scope to the service databases throughout the numerous services exposed (Reed, 2003). The backbone to the infrastructure sees an integrated grid middleware capable of using a “40-gigabit-per-second optical network” deployed, enabling researchers access to research projects on fields of astronomy, HEP and biology virtualization services based within and around the five major institutions (Reed, 2003).

On a much smaller scale to that of the TeraGrid, is the White Rose Grid connecting three of the Yorkshire based universities Leeds, Sheffield and York. It situates four nodes, two of which are at Leeds, splitting the functionality of the grid resources around certain tasks orientated either on computational fluid dynamics and high performance computing applications (Clark & Schmidt, 2005). Many of the applications that are seeing use on the WRG is real time data analysis with the DAME and Broaden project aimed at using the resources of the WRG in a bid to efficiently analyze aircraft engine data diagnostics. Other projects such as the ones in York are utilizing the WRG to collaborate in the DAME project but also further scientific research into the field of bioinformatics. Again similar to the TeraGrid the WRG exploit existing standard mechanism evident in the Globus middleware to bridge service usage across the university sites (Clark & Schmidt, 2005). Although significantly smaller in scale compared to the TeraGrid, the WRG delivers unrivalled processing capabilities to that of a single supercomputer could provide, however it still poses a problematic perspective in maintaining authentication of grid credentials using UK National Grid Service (UK NGS) certificates and login procedures (Clark & Schmidt, 2005). Considering this is smaller scale deployed infrastructure than the TeraGrid, effective management and complying with different administrative domains remains the principle outline on authentication, authorization and policies including single sign on, mapping local security mechanisms, delegation and community authorization factors (Foster, The grid: a new infrastructure for 21st Century science, 2003). Supporting this theme the National Grid Service (NGS) have base line services in place making essential support and providing a suitable infrastructure and vital components more prevalent (Geddes, 2006).
Uses of the EGEE grid have seen an application deployment of interest; the Taverna workflow management system from (Maheshwari, Goble, Missier, & Montagnat, 2009). It presents an ideal outlook and importance for bridging the Taverna system with an activity gLite plug-in onto the EGEE grid infrastructure. The context of the T2 Taverna plug-in mainly focus on complex medical imaging which are resource intensive applications like interpolating “a series of 2D images corresponding to one volume to form one 3D image” (Maheshwari, Goble, Missier, & Montagnat, 2009). The objective of the research reveals a need to improve efficiency through grid computing by using the gLite middleware plug-in, to provide the multi-threaded interface into the vastly distributed grid service infrastructure workflow. The benefits of the plug-in display a significant performance increase for medical imaging workflow being ported to the grid infrastructure, making full use of batch orientated and multi-interface distributed infrastructure. Although the Taverna plug-in used the grid infrastructure and gLite middleware, it shows the context and realism of medical imaging taking full advantage of the distributed, somewhat otherwise idle processing capabilities. This provides a similar approach that will be taken in re-deploying the pathology medical application onto the cloud middleware. Looking specifically at the application re-deployment to take full advantage of the elasticity, scalability and potentially addressing other fundamental issues such as data management and parallelism through the deployed resources.

2.3.4 Middleware

A grid infrastructure like many other development paradigms could operate without the need of a middleware, however incorporating a standardized format elevates issues such as non-standard protocols and allows networked computers to effectively communicate. Grid middleware: Globus is a development toolkit that mediates mostly between low level heterogeneous resources and high level virtual organisations (e.g. end users). Similar in construction to a variety of proprietary software it negotiates between the high and low level services without exposing the user to the physical service, only providing reference interaction in the form of an interface/service for the end user. Its primary goal provides a type of Service Level Agreement (SLA) between the participating users of the grid infrastructure. This establishes how the user interacts with the types of service available to them, anything from basic utility computing (i.e. loaning distributed machines on a pay-by-use model) to more software-as-a-service (SaaS) as seen in cloud computing terms (Foster, Yong, Raicu, & Lu, 2008).

Known grid middleware currently include many proprietary and open source toolkits like gLite, UNICORE (UNiform Interface to COmputing Resources) and widely used Globus GT4 middleware currently favoured and deployed in many grid infrastructure (Foster, Kesselman, & Tuecke, 2001) (Globus_Alliance, Globus toolkit, 2010). The Globus toolkit is purposefully based on providing
interaction on the same par as a web service, through protocols such as SOAP, WSDL documents and WS-Inspection standards (Foster, Kesselman, Nick, & Tuecke, 2002). These common standards establish the typical service orientated architecture which associates grid service and applications with the ability to interact amid web services, via the existing protocols. Being able to invoke such services is provided through the WSDL document, an open standard language using XML mark-up, which describes the service and the varied methods and how to access it. This is then published and registered as a service through a service registry such as UDDI (Universal Discovery Description and Integration) (Foster, The grid: a new infrastructure for 21st Century science, 2003). With the Globus toolkit constructed using service and grid level applications, it’s able to support the discovery, monitoring, resource management on the same principal as to a web service. Thus producing a commonly known notion of a service orientated architecture (SOA) for distributed resources or services. Throughout the different versions from 1.0 to the latest release of 5.0, Globus has maintained an open standard grid service approach providing limitless capabilities of many different technology hardware infrastructures (Globus_Altalliance, Globus toolkit, 2010). This is evident also in the evolutionary cloud computing model Nimbus that has developed from the foundational model of its predecessor the Globus GT4 toolkit. The latest edition to the Globus evolutionary model has seen the introduction of Globus GT5 toolkit launched, in March 2010 (Globus_Altalliance, Globus toolkit, 2010).

Apart of the review has uncovered various projects that have migrated workflow applications onto the grid and taking full advantage of the resources at their disposal. One such project involved using a grid middleware is the (Maheshwari, Goble, Missier, & Montagnat, 2009) and the Taverna management system on the EGEE grid. Essentially it bridges the gap of their application with the inclusion of a gLite plug-in, tapping into the power of distributed computing and storage. The Taverna project is an ideal paper that presents a similar association to this project. One major difference that the Taverna project has is that it is implemented using grid middleware software rather than the cloud middleware proposed by this project. However the project similar in scope to a medical application deployment method provides a valuable addition to review, and the approach taken to address a similar medical imaging project.

As a core part of the project, the re-deployment of the pathology application will use the Nimbus cloud middleware. The Nimbus middleware is a natural extension evolved from the Globus GT4 toolkit. Having a good understanding in both the theory and the practical training of the Globus GT4 toolkit relatively places the project with capabilities to address the re-deployment effectively and more directly. Any training issues uncovered in the Globus GT4 toolkit phase of the project will no doubt provide a better plan of action in re-deploying onto the cloud. As apart of the literature review process there is still the need to investigate the other frameworks available to grid architecture, thus
enabling an extensive overview to be carried out from the various implementations currently operating and actual technical abilities of these systems that assist in delivering the outcome.

2.4 **Clouds**

Cloud computing is becoming increasingly known for on demand and utility computing. Although cloud computing definitions do not present a unified consensus (Vaquero, Rodero-Merino, Caceres, & Lindner, 2009), it is still a relatively new concept effectively superseded from the grid paradigm, and other technologies such as cluster computing and generally from distributed systems. The two main services that clouds are commonly associated with are commercial entities like Amazon (Amazon_EC2, 2010) and Google, also with the open and academic communities such as projects involving Eucalyptus, Nimbus and OpenNebula middleware. Generally clouds create a virtual interface to provide users with either or all of the following services, infrastructure-as-a-service (IaaS), software-as-a-service (SaaS), platform-as-a-service (PaaS) and is set to become a computing utility model inline with other utilities as gas, water, and electricity (Buyya, Shin Yeo, Venugo, Broberg, & Brandic, 2009).

2.4.1 **Architecture**

![Figure 2 Taxonomy for cloud computing. Reference (The_Cloud_Computing_Use_Case_Discussion_Group, 2009)](image)

A cloud computing architecture is the backbone service of the cloud itself. The construction of the cloud services loosely resembles a condensed OSI model consisting of the user/client, application, platform, infrastructure and the physical hardware connected such that they can engage resources on demand in real time between subsequent service layers. A commonality shared with both grids and more so clouds is the use of web services, which are a type of loosely coupled business logic located on an internet or intranet domain linking applications together. A web services generally comprise of
open standard XML scheme description language more commonly know as a WSDL. This establishes an open standard description language neutral to specific programming paradigms enabling a versatile invocation method to the application service. A certain level of virtualization in grids enables virtual organisations to invoke remote methods and service applications, this is again adopted in cloud. Furthermore with clouds, it extends the use of a complete virtual environment known as virtual machines “VM image” essentially creating a virtual representation of the actual machine above the physical infrastructure and being able to dynamically change its configuration whilst in operation. (The_Cloud_Computing_Use_Case_Discussion_Group, 2009).

Central to any cloud architecture is the incorporation of the software-as-a-service (Saas), platform-as-a-service (Paas) and infrastructure-as-a-service (Iaas). This is a structured stack as previously introduced, enabling access across the three service tiers enforcing the individual service level agreements (SLA) a type of contract ensuring uptime, security and privacy requirements tailor to the system based user privileges (The_Cloud_Computing_Use_Case_Discussion_Group, 2009).

Software-As-A-Service (Saas)
The software-as-a-service presents the user with capabilities of using the pre-installed applications that the provider has available on the cloud infrastructure. Effectively the client does not have control over the underlying network, security, operating systems and the general cloud infrastructure, but merely uses the application service in the sense of a thin client. Whereby the client does not actually own any of the applications of the cloud itself, but uses the service through an interface such as a web browser commonly seen with (Google_Apps, 2010). Clearly this type of service is limited in the scope of providing a procedural interaction to the service/application, this simple setup is effectively associated to grid interaction procedures to remote grid service invocations.

Platform-As-A-Service (Paas)
This is more on the basis of providing a framework for the user to deploy and debug applications on the internal cloud infrastructure. Providing a middleware-style service the consumer has some direct control over the hosting component services such as the operating system, application parameters similar to an employee having access to their user account in a company (Cisco_No.3, 2009). Active businesses that are providing platform-as-a-service include the (Microsoft Azure, 2010). The simplest way of describing Azure is an environment in which specific MS Windows applications are able to manipulate and store persistent data in the cloud (Chappell, 2010). Having this level of ability in grids is almost an impossible task unless you are the service provider and having to re-engineer the service upon every user request.
**Infrastructure-As-A-Service (IaaS)**

The infrastructure provides the greatest degree of control between the three service tiers. Typical IaaS providers present you with the ability to take advantage of the raw processing power. A typical provider is the (Amazon EC2, 2010) supplying the capability of renting a virtual environment image with complete control over manipulating CPU, memory, and disk capacity along with the OS and applications, but limiting access to the cloud underlying infrastructure (Cisco No.3, 2009) (The Cloud Computing Use Case Discussion Group, 2009).

Fundamental to all three tiers is virtualization, tying together the subsequent layers whether it is on deployed through a public, private or hybrid cloud infrastructure (Sun Microsystems, 2009). A versatility of using a virtual image in a cloud over such mediums as internet and intranet, exhibit a global accessibility option rather than a secluded location confined by a grid network infrastructure. Helping to maintain an interoperable environment for consumers to migrate their VM image to other providers is the use of open source software and application programming interfaces avoiding “vendor specific lock in scenarios” (Sun Microsystems, 2009). This approach is assisting in pushing forward the utility based model of computing an implementation already in operation with Amazon EC2 and S3 storage payment models, where consumers are charged based on the number of CPU, memory, or data storage hours taken in processing a particular use of the system (Foster, Yong, Raicu, & Lu, 2008).

**2.4.2 Application**

At present there are an unprecedented number of software applications available to purchase, substantial in range varying in infrastructure and platform dependencies. One common factor seen in the current age is the issues of having to purchase or develop software either on a commercial licence or for private usage, often using the application for basic commands to high level tasks given the context. Further problems with particular applications can lead to specific demands on the type of IT infrastructure configured to operate it; this can be from memory, processor speeds, amount of physical hardware storage and most often highly dependant on the operating system (e.g. Microsoft, Apple OSx and various Unix systems). With high demand for resources and particular demands on platform requirements for present and even legacy systems it can become a vital issue to address when upgrading or migrating system setups (Sun Microsystems, Optimizing applications for cloud computing environments, 2009).

With clouds having the innate quality of flexibility and utilising configuring VM environments to practically any requirement it becomes an ideal architecture. Certain applications such as batch payroll runs, logistical data, processing scientific data such as the LHC and planning natural disasters require an immense quantity of resources commonly causing bottlenecks in traditional server
deployments. Although a server can eventually cope with the processing load it’s not ideally customized to fully use the physical resources to their potential, whereas a cloud harnesses complete resource control providing a virtual image proficient enough to elastic-ate and scale given the core infrastructure supplying the processing power (The_Cloud_Computing_Use_Case_Discussion_Group, 2009).

The key to the cloud computing applications is that the client does not inherently own the application; merely the applications are hosted by the provider and enable the user to access the service it specifically provides. More so the software is likely placed on a virtual manager infrastructure, such as Nimbus, OpenNebula (Nimbus_Projects, 2010) (OpenNebula.org, 2010). These virtual managers provide the interface for utilising the cloud resources. A non-cloud example of this type of approach can be seen through the use of the (Google apps, 2010), providing basic business services such as word processing and e-mail.

Offering these types of capabilities to the user provides an ability to access cloud applications relatively from any domain, day or night, permitting that the cloud is accessible from the public/private domain it is deployed on. The scope of this vast accessibility places the user to not only use particular applications on demand, but take advantage of scalability and elasticity capability in real time. The pay-by-use model is the solution model that provides the main functionality for its cloud users, this can be a user simply paying for the CPU processing time taken to run the application or additionally paying for the virtual storage for data processed (Amazon_S3, 2010) (Amazon_EC2, 2010). Considering that the user does not have to initially purchase physical hardware and software, it clearly places a significant reduction on client overheads and better usage of a service orientated architecture than a grid infrastructure.

It is key to understand that not all applications are suitable for deployment on a cloud, although there are significant advantages to this type of architecture it does not provide the silver bullet option to solve every application performance and resource issues. Fundamentally a cloud by definition facilitates vast proportional resource capabilities; it is applications specifically restraining core resources that are likely to be formatted and deployed through a cloud platform. Applications such as the severely resource intensive demands on systematic storage of algorithmic data, dynamically produced in high peek fluctuating environments is one of many possible scenarios clouds are able to adjust and boost resource functionality. This is one factor that bottlenecks the histopathology application, similarly explored and used in the LHC. With small every day applications such as word processing, again it would be viable to be re-deployed, however the implication and cost to initialise a virtual machine image, starting up the virtual environment and accessing and using the deployed application would most likely produce far more additional resource consumption compared to a
standalone hardware installation. This would be mainly evident in the virtualisation of the application, with resources replicating unnecessarily and performing twice the amount of resource requests, firstly within the virtual machine, secondly translated to the physical infrastructure housing the virtual environment.

### 2.4.3 Infrastructure

There are three commonly deployed infrastructures; public, private and hybrid clouds. Each of these deployments, does present particular trade offs for IT organisations considering to migrate their services. However although these three cloud infrastructures provide a new deployment approach, its operational capabilities mirror other infrastructures in terms of security, Quality-of-service (QoS), availability similar to public/private networks setups, and more so on the internet and intranet level.

Constructing a cloud infrastructure does present problems of software and hardware issues; however the natural ability of the cloud to deal with heterogeneous software and hardware is simply dealt through virtual managers acting as a medium between the applications and the physical machines. Keeping a cloud to conform to open concepts proves to be a recommended option specifically avoiding vendor locking issues (Sun_Microsystems, 2009). Preliminary uses case scenarios where cloud solutions have been deployed craft a valuable cost and error reductions model and increasing the benefits to the company management (The_Cloud_Computing_Use_Case_Discussion_Group, 2009).

As with any cloud infrastructure the main basis for the services provided must appear transparent as possible from the user and even the developer’s perspective.

### Public Clouds

This type of cloud infrastructure is more on the mainstream approach, providing accessibility to the wider domain customers. It is usually implemented by third party providers giving a utility computing orientated service (e.g. utilities as gas, electricity, water…) (Buyya, Shin Yeo, Venugo, Broberg, & Brandic, 2009), such as incorporating provisioned resources and ability to scale on demand and respond dynamically. The actual deployment of the cloud doesn’t necessarily have to be on location of the customer’s premises, but enables the customer access through authentication services via web applications or web services over the internet. Such cloud providers as the Amazon EC2 and storage S3 cloud present consumers with the ability to register and use a pay-as-you-go utility charging scheme, whereby consumers after formatting their virtual environment based on their user privilege level effectively pay for computing time utilized or in the case of the S3 cloud payment for downloading and uploading persistent data to the virtual infrastructure (Amazon_EC2, 2010) (Amazon_S3, 2010).
Looking at the public cloud from an enterprise view one of many benefits is companies with a smaller private cloud in are able to shift infrastructure risks in the event of exceeding their resources abilities onto a much larger public cloud provider, even just temporarily (Sun Microsystems, Introduction to cloud computing architecture, 2009).

**Private Clouds**

Private clouds are intentionally built for use by single clients/enterprises; this can range from IT organisations such as the Amazon, through to academic research institutes for example the cloud-test bed at the University of Leeds. The clear advantages of using and implementing a private cloud essentially provides high level control over the Quality-of-service (QoS), security, control of information and how the virtual managers and applications are deployed onto it. Also from an academic perspective it gives the opportunity to discover cloud technical boundaries, not always visible through commercial providers.

**Hybrid Clouds**

A hybrid cloud is a cross integration between public and private cloud infrastructures. As middle layer model it augments with public and private cloud resources depending on the rapid workflow fluctuations being migrated to a public cloud provider until such resource demands have subsided (Sun Microsystems, 2009) (The Cloud Computing Use Case Discussion Group, 2009). This is a key functionality that the hybrid capabilities can address at a moments notice, although there are issues of security when transferring sensitive data from organisations to public clouds for processing. However, this can be addressed through Service-Level-Agreements (SLA) between the user and the provider to ensure full compatibility and personal level of service maintained throughout.

**2.4.4 Middleware**

Monitoring traffic and user demands are done through a set of rules or more known as protocols. These protocols are a special kind of middleware that serves the purpose of allowing networked computers to communicate with each other. The revolutionary development of cloud computing is pushing the boundaries of using virtualization, autonomous abilities such as self-managing (certain measures taken in monitoring and repairing problems) and conveying this through a transparent dynamically environment. At this point there are a number of middleware available to bridge the gap for non-cloud complainant applications ranging from Nimbus, OpenNebula and Eucalyptus and many more.
Middleware frameworks such as OpenNebula again provide the bare essentials for re-deployment, but is virtual management infrastructure primarily using virtualization tools (OpenNebula.org, 2010). Projects as the reservoir model seen in (Rochwerger, et al., 2009) present a development focusing on business service management and challenges of leveraging and embedding the virtualization inherent within the model itself and OpenNebula.

In correlation to this project, the Nimbus cloud middleware will be used for the re-deployment of the pathology application. This open source framework provides the necessary capabilities to deploy and manage virtual machines on the physical resources. One main difference of deploying the application on Nimbus is that while it exposes remote interfaces (such as those based on the (Amazon_EC2, 2010) and with the Web Service Resource Framework (WSRF)) it can handle more of the security issues compared with the OpenNebula middleware, (Nimbus_Projects, 2010). The more subtle fact is Nimbus has evolved from Globus GT4, providing a more interlinked development between the training on the grid and cloud middleware that is being undertaken in the project. Other Nimbus projects have “proved popular among projects as diverse as high-energy physics, computer science, bioinformatics, and more recently economics”, (Keahey & Freeman, 2008). Having the advantage of doing the foundational training and literature review on the Globus toolkit does set the project with better prospects in re-deploying the application.

For this project I will emphasize in using Nimbus, for its availability and also for the sake of the evaluation as another similar project will be re-deploying the same application under a different virtual manager infrastructure OpenNebula. Therefore it is important for the evaluation section for both projects being able to compare the implemented application on the different virtual manager infrastructure.

### 2.5 Histopathology Application

The pathology application which is going to be re-deployed onto the University of Leeds cloud test bed is designed to manipulate 2D virtual slides. These virtual slides begin the same initial preparation techniques used by pathologist to obtain tissue samples, once at the stage of analysing the tissue cells, naturally conducted under a microscope, the glass slide is taken and has a digital scan taken and stored on a specialist server within the hospital. Preparing and using digital slides has motivated pathologist to look at new lines of diagnosis, one of which is creating a three dimensional model of the set of tissue cells. This sophisticated model and reconstruction approach is the main downfall to which the application is suffering due to the demands placed on the computing resources available.
2.6 Current Application State

The current application offers pathologists to view digital virtual slides which are a set of 2D image slices taken from patient tissue samples. The application then invokes a recognition algorithm purposefully designed to capture and identify nucleus detection within the image, through applying de-convolution methods and various stage of image adaptations the system implementation (which runs on a single PC) does types of “volumetric reconstruction” which is limited by the physical computational and memory limitation of the single machine, (Magee, Djemame, & Treanor, Reconstruction of 3D Volumes from multiple 2D Gigapixel Microscopy Images using Transparent Cloud Technology, 1 Internal Document). Certain histopathology reconstruction process involves co-registration images slices and predominant use of algorithms to align and manipulate large Gigapixel images into suitable 3D volume rendering for analysis, as shown in (Magee, Treanor, & Quirke, Non-rigid Registration of Gigapixel Histopathology Images for Volumetric Reconstruction Using by Multiple Registration Combination, 2 Internal Document). Additional factors limiting the current system include the specialist single server which hosts the accesses for the large backend database to retrieve requested images by the end user. Considering that these tissue sample images are potentially up to Terabytes in scale (with single images typically 100K x 80K pixels, and up to 20 GB uncompressed) (Treanor, 2010 (Private Communication)), it proves somewhat cumbersome in being able perform automatic image analysis.

Creating and forming a digital slide for the application, involves a lengthy routine commonly followed by pathologist at the hospital laboratories. One of the many procedures is first obtaining a tissue sample whether from a biopsy (an endoscope device fed into the patients body and small excisions are taken from the target area of interest, for example this could be tissue from a liver where cancerous cells have formed) or complete removal of patient organs (Treanor, 2010 (Private Communication)). Each cell sample is located and prepared by dissecting it into smaller manageable pieces. The next stage sees the tissue piece coated in a paraffin resin keeping the cell in a suspended state permanently. Using special precise equipment each tissue cell block is sliced into strips of thin slices smaller than single strand of human hair and mounted carefully onto glass slides. Each glass slide is then processed by dipping them into various combinations of dyes, staining the cell in accordance to the tissue it was obtained from. This procedure enables the pathologists to purposefully add colour to the already colourless tissue cell and use the staining method to highlight areas of interest such as nucleus detection, which turns a violet colour once stained (Treanor, 2010 (Private Communication)). Having prepared the glass slides the penultimate stage would fall to pathologist to perform a diagnosis through a microscope, however it is at this stage the glass slides enter the high resolution scanning phase creating a detailed snap shot of the tissue cell and stored onto server (Treanor, 2010 (Private Communication)). Converting these slides to digital virtual slides comes through the use of a modified microscope producing resolution capable of “up to 100,000 dots per
inch, they may be 100,000 pixels by 100,000 pixels” (The_Vision_Group, 2010). Having these potential enormous amount of data, and being able to magnify the images to a suitable level in relation to that of a traditional microscope isn’t as simple as rotating the lens. The server which holds the data is a single machine with massive back end storage of these typically sized image samples. This creates substantial problems of the server accessing and rendering the HTTP requests from the user wanted to analyze the tissues samples.

Having committed a copy of the virtual slide to a server repository, it is this same location and resources used by the server for the application to perform the necessary calculations and display the high resolution images.

![Current Implementation of the System](http://129.11.64.252/Research_S/3D_Histo/Liver/Cirrhosis/Liver_3/H%20NE/III275.svs)

**Figure 3 Current Pathology Interaction**

With a limited range of physical resource available to the server in the first instance it provides a restricted range of ability to support multiple users. The range of virtual slide resolution qualities also contributes to the decrease in performance of the application sustaining an interactive environment to the end user. With the application being hosted on a network this continues the variety of bottlenecks commonly seen through network latency issues and maintaining a connected link to the server and user machines.

The project will address the re-deployment of the application which will ultimately be seen through the Cloud middleware Nimbus. The application will foremost re-deploy onto the cloud test bed using the nimbus middleware, making it operational within the virtual environment. This will provide the user to transparently access the application which will be hosted as a software-as-a-service (saas) on the cloud test bed infrastructure, which is already setup at the University.
2.7 Benefits of moving to a Cloud Virtual Infrastructure Manager – Nimbus

A cloud in many retrospects will bring versatility to the histopathology application. With no apparent scalable system supporting the current application, by definition the cloud would help to elevate computation and communicational limitations evident within the existing deployed architecture (Vaquero, Rodero-Merino, Caceres, & Lindner, 2009) (Keahey & Freeman, 2008). The use of scalability assists a system wide resource utilisation dynamically configured to meet persistent high/low demands scaling effectively and reflected by use of specific resources when necessary. Dynamic scaling presents improved performance, dealing with memory and CPU intensive resources, preventing issues of system freezing and memory stack overflow situations. A key functionality supporting scalable cloud architecture is enabling elasticity highlighted though the use of virtualization. As the application is being re-engineered to operate on a virtual machine deployed onto the cloud, the underlying cloud Xen hypervisor dynamically engages with more resources as your client environment usage alters, distributing the load balance across the underlying system architecture.

![Proposed direction for the Re-deployment of the Application](image)

2.7.1 Why Choose Nimbus

Nimbus provides a set of open source tools for development (Nimbus_Projects, 2010). General uses of Nimbus have emphasis on science, but many non-scientific use cases are supported as well (Keahey & Freeman, 2008). It provides an infrastructure-as-a-Service (IaaS) cloud computing solution. Allows a client to lease remote resources, through deploying virtual machines (VMs), each of those resources can be configured to represent an environment desired by the user. Also it supports the Globus GT4 toolkit execution commands, making it an ideal development environment as it relates and maps directly with the Globus training that is apart of the project schedule.
2.8 Methodologies

In order to test, re-engineer and deploy the pathology application on the cloud TestBed, requires the solution to conform to a specific cloud middleware currently deployed. With the cloud currently operating two middleware solutions OpenNebula and Nimbus, the designing must conform to the specific middleware standards and then tested, with the latter middleware being the case here.

2.8.1 Software Development

Developing the prototype breaks down into three categories, Construction of a Service, Deployment of a Service and Operation of a Service. With each section relatively self contained and requiring a cyclic repetition of re-engineering the pathology application it ideally conforms to the evolutionary methodology. It seems that although the pathologist have well defined set of requirements that the V model (Verification and Validation) may be appropriate, but if the client is unsure of many details then Extreme Programming (XP) would be better because the client can influence the design of the product as its development proceeds. With the validation of the project prototype the V life-cycle and other broadly set iterative models highlights the verification and validation processes, but it assumes that the requirements are stable, and hence the various tests can be planned well in advance. This is not so apparent and applicable to the rapid prototyping procedure that will dictate the course of the development and implementation, more so the XP model requires a test suite to be produced for each iteration effectively testing code on each progressive level applied through various compiling and debugging of the code. A formidable factor for using XP within the project is the effect of the planning and scheduling, using iterative models uniformly provide well defined schedule of activities; such that it is also clear when the project finishes. Since it comprises an indeterminate number of iterations, the use of XP means that it will be difficult/impossible to produce a well-defined schedule, and it will be difficult to determine how much of the project still needs to be done at any time in the project. Using a Rapid Application Development maintains the quick prototyping and small timetable, but is likely to demonstrate that the resources needed for even well defined stages will be difficult to predict. A limiting factor that may hinder the quality of the final product is that quality cannot be tested into a product; it must be designed in from the beginning. Although it will be much easier to guarantee the quality of the final product using the iterative life-cycle models than XP, the prototype using the XP methodology will identify potential conflicts in building the product earlier and provide a level of correctness and consistency of the requirements and eliminating undesired vague functionality at each progressive phase (Sommerville, 2006). Appendix F highlights the different product life cycle models available.
2.8.2 Cloud TestBed
The University of Leeds currently operates a small scale Cloud TestBed, based within the School of Computing (Soc) (See Figure below which illustrates the clouds various layers). The Cloud TestBed comprises of two physical machines (Dew, Djemame, Nizamani, Armstrong, & Klancnik, 2009) capable of handling up to twelve virtual machines. It construction supports the three key services of Infrastructure (Iaas), Platform (Paas) and Software as a Service (Saas) models. In addition to this two Virtual Infrastructure Managers (VIM) Nimbus and OpenNebula middleware have been deployed either of which will be used to deploy the pathology application onto and perform the necessary testing needed.

An important consideration to this project as well as the software development methodology is understanding the formidable core functionality that the new cloud computing environment presents. With the project aiming to re-deploy and existing nucleus detection application onto this relatively new paradigm there is a substantial amount of unfamiliarity that must be understood personally. More precisely the methodology is not only about the software development, but more intricately involved in the re-engineering characteristic first of all apparent in using the cloud testbed and the re-engineered application. Simply being able to drag and drop the application onto the cloud is a clear understatement to the volume of preliminary work that must be addressed first hand. As the project aims to deploy using the Nimbus virtual infrastructure manager, an open freeware software, the re-engineering structure involves:

- Looking at creating a new virtual machine or virtual image
- Learning and understanding Nimbus invocation commands and environmental setup procedures
- Tailoring the environment to cater for the user application to be deployed
- Adapting the environment with new functional requirements

Figure 5 University of Leeds Cloud TestBed

Current work concentrating on:
- Prototyping of cloud system components

Two cloud architectures:
- OpenNebula
- Nimbus

Created using:
- Xen and Open Nebula
- Dell Commodity Servers:
  CPU: Quad Core 2.83Ghz / Memory: 4GB / NIC: 1GBit
• Understanding and using basic command sequences to familiarise with and then utilise specific operation capabilities on the cloud

This brief indication to the level of information needed to be understood has to be factored in, not only applicable within the software development methodology.

2.9 Summary
With a comprehensive coverage on Grids, Clouds, and Virtualization complete, the design and development of the project can commence helping to ensure a prompt delivery with the project milestones.
Chapter 3: Design

3.1 Introduction
The design phase comprises of two main elements, planning modules and new functionality, within the boundaries of a distributed system, using class/sequence diagrams and UML modelling, reviewing issues in the incremental development of the prototype found in the testing phase of the developmental process, with each successive iteration to address and rectify it. Planning a method of studying the integral design phase of the application, via various further research forms the basis to analyse the problem domain. The design places emphasis on establishing a transparent re-deployment, minimal interactive front end interface and creating a simple deployment relatively similar in nature to the current application command line approach taken.

3.2 Methodology
An appropriate development process has been formulated for this project. It brings together the positive aspects of the developmental processes as discussed in the literature review, ensuring to accommodate the prototyping of system components, establish a small scale orientated iterative and incremental development combining relevant agile techniques of UML modelling and class diagrams. The development process consists of three phases designing, implementing and testing, preformed iteratively as a whole, but systematically through each new implemented functionality ensuring minimal issues within the coding. Due to the process of prototyping within the project, small scale testing will be conducted on each iteration of the development when a new feature is introduced, but full scale performance/efficiency/usability testing of the application will be combined into the implementation chapter because of the cyclic nature of the development. Thus it would be more beneficial and cover any new functionality unforeseen and combined to the operational prototype.

3.3 Project Management

3.3.1 Original Schedule
This section will describe the initial project schedule, giving an overview to the intended final year direction the project was committed to produce. The following list illustrates the nature of the four major stages needed to be completed to ensure the deadlines were met. For a week to week schedule, see Appendix D. Accompanying this is a Gantt chart in Appendix D, presenting a visual version representation of the schedule.

I. Preliminary investigation (18/01/2010 – 26-02-2010)
   a. Grids
   b. Clouds
II. Pathology Application (01/03/2010 – 04/05/2010)
The early stages of the project were to primarily focus on the understanding behind the histopathology application and to gain a firm perspective of research efforts conducted within the distributed systems surrounding cloud computing. The aim to finish the background and planning material by the end of weeks 6. With the addition of the mid term project due for submission on week 6 (included in the initial weekly schedule) it was vital to ensure all planning and preliminary work to be completed, as it formed the necessary material for the document. Having completed the background and the planning the implementation and testing should have started, this can be seen in week 7 of the weekly schedule. The implementation involves creating a prototype for re-engineering the nucleus pathology application, and applying the appropriate testing criteria. Accounting for the Easter vacation between weeks 8, the development is set to continue and ensure the evaluation of the project is conducted. Being able to create the final year project report the plan is to begin the core write up mid way in the Easter vacation to allow for prompt delivery and flexibility to make alterations.

### 3.3.2 Revised Schedule

I. Preliminary investigation (25/01/2010 – 10-03-2010)
   a. Grids
   b. Clouds

II. Pathology Application (01/03/2010 – 12/05/2010)
   a. Design (01/03/2010 – 22/03/2010)
   b. Implementation (22/04/2010 – 10/05/2010)

III. Evaluation (19/05/2010 – 21/05/2010)

IV. Submission (26/05/2010)

The first revision to the project management involved the background reading. Having difficulty locating necessary research information, initially anticipated, to take a short amount of time mainly came in the form of looking into grids. With the project hoping to identify a comprehensive overview of both clouds and grids it was formulated to research both architectures, applications, infrastructures and middleware for both paradigms. This was to ensure a direct coverage of literature and gain a deep understanding the main principals to both platforms. As can been seen the pathology application analysis and design aspects started as planned and finished on schedule with the initial intended plan. However, the implementation phase of the project
suffered a major set back due to lack of information available when it came to re-deploying the application. With the project training required for the grid, on the globus GT4 toolkit using the Sun VirtualBox environment needing to be installed, the technical requirements needed a 64 bit image and CPU compatible platform to work on. Having no immediately available 64 bit machine at my disposable, apart from the school of computing which did not have VirtualBox GUI installed, the project implementation was pushed back by almost two weeks. This misjudgement on anticipating the Sun VirtualBox was capable of accepting and running on my home 32 bit system was a big error on not identifying this earlier. Fortunately having personally obtained a laptop capable to run the training material platform dealt with the issue, but it meant the scheduled development initially predicted pushed the Easter holiday implementation further behind.

Due to unforeseen circumstances the core of the at least three weeks of the implementation between week 8 and start of week 10 key files were lost. This became a serious issue that affected the entire performance and the testing phases, which had to be pushed further back to an already tight schedule. Obtaining a two week extension to the initially schedule project deadline meant the project had to be condensed further, and re-evaluated to comply and submit for the 26/05/2010. This can be seen in the final Gantt chart in Appendix D. To ensure the implementation was successfully completed further time had to be applied to recuperating previous created software functionalities quickly.

Having illustrated the revised project schedule and how the project management suffered a major setback the following design, implementation, testing and evaluation chapters follow the reassessed schedule deadline.

### 3.4 Application Design

As the pathology application in place at the moment use a variety of different functionality for manipulating the digital histopathology images, it is unclear to the extent of the technical knowledge that they poses. As the current application upon each invocation uses command line arguments supplied to it by the user for the various zoom, height, width and other parameters it seems necessary to continue with this method of interaction. Ensuring the application meets the required level of acceptability for the pathologist to comprehend the new re-engineered solution; agile methods of UML modelling, Class diagrams accompanied with Sequence diagrams will enable a transparent outlook on the functionality of the prototype and deviations that are not clear. Producing these diagrams form a user preference should help to convey how to operate the new prototype without extensive knowledge, but from a developer’s aspect it will aid the implementation phase, regarding how and why and what the new code introduced to the application should feature.
3.5 Application Implementation

The application is set to be deployed onto the University of Leeds Cloud TestBed. Its design element will explore and investigate deploying the application first as a re-engineered Globus Resource Allocation Manager (GRAM) 2 compatible non-web service, second a Java Web Service deployed onto the cloud accessed as a remote resource and finally Globus GRAM 4 web service. Once the application has been successfully implemented it will then be migrated onto the Cloud TestBed via the Nimbus Virtual Infrastructure Manager (VIM) and tested with sample data sets. It is hoped the project is able to progressively link between the individual development plans. Although each of the three suggested design models requires different technical abilities to deploy and re-engineer the application, they each follow a closely linked structure. With the standalone java web service its natural construction includes the use of build operations of a WSDL document, similarly used within the GRAM GT4 development toolkit. Developing in the GT2 toolkit, although vastly different to the two web service orientated methods; it still deploys applications in a similar scope to that of the latest GT4 toolkit. These models although different in constructing the service follow similar underlying principals that will assist each other when and if difficulties are evident.

3.6 Application Testing

Testing the application forms the need to observe the application from both the user and the technical capabilities the systems can perform. Involving the user includes the ability to assess the project prototype first hand against the current system, identifying potential improvements and or missing functionality that may be required and not available. Essentially gathering formal feedback from the medic by asking s/he to use their current application on one machine and then to use the re-deployed application on another machine. Thus being able assess the users opinions on the application, asking the user in an open type interview the advantages they see compared to the old system which would form a key foundation to the analysis. This could lead to exploring the usability, accessibility of the application and potential improvements, such as features within the current application that they might want to remove or have re-designed to provide a better interface to the application.

```
globus@globus01:~/java$ ./nuc_det_server
Usage: <url> <zoom> <minx> <miny> <width> <height> {<channel>} {<stain>}
globus@globus01:~/java$
```

```
globus@globus01:~/java$ ./nuc_det_server "http://129.11.64.252/research_5/10_Histo/liver/Cirrhosis/Lliver_3/H%26E/51275.svg" 20 10000 10000 1000 1000 1 "H&E"
> ~/output/out.xml
```

Figure 6 Command Line Invocation of the nucleus detection application
The technical perspective provides the mechanical identification to rectify any problematic functionality and vitally perform performance evaluations to the efficiency and measure the execution speed of the old versus the new system elements. It also enables testing from a software engineering approach to determine, whether the system is modular in design, structured and if it is potentially extensible.

### 3.7 Current Pathology Application Design and Setup

Currently the application closely resembles a classical client server model interaction. More so the interaction use is based on a thin client model, as the application interface is hosted through the server. This in turn accesses the back-end data required by the client at particular intervals. The relative bottlenecks that obscure the system can be seen at the server level interaction to the back-end database; this consists of a substantial high stature of material stored and being requested. Individual images have been stored through a digitised technique, producing an image with detail in comparison to that of microscope lens. These images are able to provide an unparalleled quality similar to a microscope format, which demands high volume physical storage capabilities. (See Section 2.6 Current Application State for further details)

#### 3.7.1 Technical overview – Invocation procedure of the current application

Once a user makes a request to the server for a particular image, all the rendering, algorithm invocation relevant to the application becomes a client side issue, whilst the retrieval of the images becomes a server side issue. Essentially which ever piece of hardware hosts the application inherits the resource intensive demands, taking the brunt of the computational requirements. The user is based on the principal of a thin client with minimal interaction with the application, such as manipulating the image by zooming, orientating, supplying further parameters and being able to navigate through the file directories.

Although it seems that the server should be able to cope with the basic retrieval and interaction techniques from the user, the concrete issue lies with the application being a resources demanding factor. It places a huge demand on the memory and processing power. The bottleneck to this setup lies with the core single specialist server not being able to perform on demand when required to retrieve and cope with substantial data quality for display. Difficulties can be mainly seen when the user processes the image and using zooming parameter within the application. Seeming like a simple request from the user, just displaying a particular image section on a vast proportional scale of detail, translates to the application and system requiring more processing power than it can physically handle. The solution to solving these issues is being able to handle influx of high resource demands at any given time; this is a central scheme in virtual infrastructure manager Nimbus, which will enable a scalable, elastic re-deployment of the application. (See figure 7 which illustrates the UML current state).
3.8 Proposed system re-deployment onto Nimbus Middleware

The system that will integrate and host the application will loosely form the concept shown in the UML diagram of figure 8. The system will be a virtual infrastructure manager (Nimbus) hosting the application in a virtual machine image for the client to access. The functional nature of the application once embedded into the VIM will retain the same level of interaction to a certain degree and allow the user to engage a familiar interface.

End User

The user will fundamentally have basic user privileges, only enabling the user to access the system and performing necessary invocations of the application and minor abilities to tailor the virtual environment (if applicable).

Developer

From a developers perspective the entire system will be accessible, possibly incorporating setup configuration abilities with the physical resources for example the cloud test-bed machines. The developer’s level of freedom on the cloud extends the end users privileges, whereas the user will be limited to a software-as-a-service domain on the cloud the developer will retain the ability in dealing with all three service levels (software, platform and infrastructure service) on the cloud. The main emphasis for the development will consist of manipulating both software and platform-as-a-service, instead of infrastructure where the physical machines reside. This is because the application will need only deal with the high level semantic service structure of the cloud and is vitally where configurations to the cloud services can be altered.
The System (Nimbus)

The Nimbus VIM (Virtual Infrastructure Manager) is the core of the applications deployment. It will on request invoke the clients VM image loading the operating environment the application has been embedded into. Effectively this will provide the infrastructure to the application operating and being deployed onto the cloud test-bed. Any issues of the application requiring on demand resources increase will be dealt through the nimbus infrastructure creating new virtual machines to spread the resource demands. The implementation of the application at present will only be exposed to two cloud nodes. This restriction of the cloud hosting two physical machines limits the amount of virtual resources that can be deployed on top the infrastructure, this will be a restraining factor in spawning more resources for testing efficiency and performance and bench marking the application against the current application deployment. The vital initiative for the project is to deploy the application and establish a proof of concept for the deployment onto to the cloud, once achieved it will offer the project to explore other vital areas of improvement to the applications data management and parallelization.

![Diagram of Proposed System Setup Incorporating VirtualInfrastructureManager](image)

Figure 8 Proposed system interaction model

3.9 Designing the prototype

As briefly discussed in section 3.5 Application Implementation the prototype will explore three avenues for re-engineering the application to be migrated onto the cloud. These are a standalone Java Web Service, GRAM 2 (non-web service) or GRAM 4 web service solution. Each of the solutions will be developed within the Sun VirtualBox environment, which is capable of operating a virtual grid cluster. Constructing the service in a virtual environment provides the flexibility to migrate the virtual...
image directly across to the cloud infrastructure. Although the prototype is to perform graphical displays of the retrieved images, the development environment VirtualBox is strictly a text based interface, similarly mirrored with the Cloud TestBed capabilities thus preventing actual displays of the image samples.

3.9.1 Java Web Service
A Web Service is essentially a loosely coupled system that supports service orientated architecture over internet technology. Its basic construction follows the use of WSDL documents that map the web service interface in the same mapping of remote objects through the IDL interface which describes it. The interface of the application essential would be another front end written in java, effectively this new interface created would simply map and invoke the C++ pathology application through the java Runtime Event, so you would have a java application but calling the backend C++ application without modifying the original coding infrastructure. The steps needed to create a new service are as follows (Apache_Axis, 2010):

1. Provide a java interface or class
2. Create a WSDL using Java2WSDL
3. Create bindings using WSDL2Java

The “Java2WSDL” and “WSDL2Java” emitters are Apache Axis tools that assist in creating a new web service; it takes the java interface and builds the necessary bindings for the proxy to call the client and server side stubs for the Remote Procedure Calls (RPC). Having created the interface and the new Web Service the UDDI registry and the Web Service Resource data would be deployed onto the cloud and each mapping for the service made accessible for remote invocations. This approach will simply create the core development of the prototype as a Web service.

Figure 9: Proposed Java Web Service Design
3.9.2 Java Web Service Using GRAM

The Globus Resource Allocation Manager (GRAM) is a basic remote job submission and control service (Globus_Alliance, GRAM, 2010). It supports staging of files, using grid credentials to manage, monitor, control (Input/Output) of submitted jobs whether starting, stopping or terminating individual submissions. The basic setup to a GRAM control service offers support to security, with both versions, one which includes the use of web services (GRAM 4) and the other which pre-dates the web service capabilities within grids (GRAM2) (Globus_Alliance, GRAM, 2010).

Designing a Java Grid service may indicate a backward momentum in the re-deployment and re-engineering of the application onto a grid, rather than a cloud. This is a deliberate strategy accounted for, as the Globus GT4 infrastructure toolkit is a predecessor and platform to which the Nimbus cloud virtual infrastructure manager has evolved through. This ties together the usage of OGSA standards commonly used in the Globus environment, but also available in the Nimbus infrastructure. Establishing a working re-deployed solution within the Globus environment gives a prime opportunity to migrate the system setup directly through to a virtual image hosted by Nimbus VIM. With the current application is C++ based, there are two pathways for the design and implementation to take, either the pre-dated GRAM 2 development path or the GRAM 4 web orientated method. From a design initiative and to present the two similar GRAM pathways, the project will outline the GRAM 4 design, specifically enabling to cover both paradigms, both equally identical in construction, but with GRAM 4 being web service which differentiates both.

3.9.2.1 Globus Resource Allocation Manager (GRAM)

A GRAM development follows a set pattern to development, implement and interact with deployed remote services. As the project is conducting training on the Globus Alliance: The Globus Toolkit 4 Programmer’s Tutorial (Globus_Alliance, Globus toolkit, 2010), this will form the template to which the added modalities will be combined. As discussed in “2.8.1 Software Development” there are three phases to the development, which are construction, deployment and operation of a service. This follows the almost identical course of action seen in creating a GRAM resource/service.

3.9.2.2 Designing using GRAM

There a number of chronological steps that has to be completed before the Java Web Service core can be deployed and utilised. These are as follows:

Step 1: Defining an interface in WS Description Language (WSDL)

A WSDL document is created using a language neutral Extensible Markup Language (XML). Based on a XML, it enables the WSDL to describe the web service, mapping and standardizing native language dependencies into XML format. A WSDL interface has the purpose to manage client application communications and create the service interface, defined as port-types within a WSDL, required by the WS Resource Framework (WSRF) to keep stateful resource information. This is a
vital component that will provide the communication channels to open and directly send invocation commands to the java methods within the remote service that will be hosting the pathology application. Using service interfaces this first step identifies what the service should do.

**Step 2: Implementing a Service in JAVA**
In order to fulfil what the service does and how it’s achieved comes within this step of the design of a service. The service implementation bare bones of the system are introduced and written in java to map the service and the resource together. Taking the service interfaces identified in the WS Description Language the procedural resource properties can be located reflect the literal bindings of each property value.

**Step 3: Deployment and Configuration of WS Deployment Descriptor (WSDD) and Java Naming and Directory Interface (JNDI) API**
Making the connection to the client becomes the WSDD and JNDI aspect to building a stateful web service resource. In order to identify with the client and the resource the web service interface has to be published through the service URI, deployed through a web service container (using SOAP, HTTP servers) using the WSDD. The WSDD is a configuration file of the services, mappings and various type data, constructing the client and the server side XML files, used with the Apache Axis tools (SOAP engine based used within the Globus toolkit environment). The use of JNDI maintains functionality of discovering and “ResourceHome” (Globus_Alliance, GRAM, 2010) implementations found within the registry of the container, this assist to provide an up-to-date registry of the current configurations available to the client and the server services.

**Step 4: Creating a Grid Archive (GAR) through Ant**
This aspect of the design for the service sees a “.gar” file being created, making use of all the files, documents and information required for the container to deploy a service. This is the penultimate section of the design binding all the previous steps into a functional state preparing the service to be built. The Ant tool is an Apache Java build tool, automatically providing the following stages for the “.gar” file creation:

- Missing WSDL bindings are rectified
- Client and Server side stubs classes are compiled via the WS Description Language
- Java Service Implementation (Step 3) are Compiled
- File directory structure organize individual files, and other class groupings.
Step 5: Deploy the service into a Web Container
Once the GAR file has been created successfully, all that remains is to deploy the service using the GT4 tool. Deploying through GT4 essentially copies the various WSDL, WSDD and compiled stubs and implementations into a GT4 tree file directory structure. Using Globus command line tools the service can be deployed and un-deployed.

Step 6: Using the client
Having completed the major work needed to construct the service, it is at this stage that the necessary development of the service can be deployed and invoked. The final stages involved merely require the java application (i.e. the client.java and the java pathologyservice.java) files be compiled. Once compiled the Globus container can be started, without security options enabled. Provided all the necessary steps have been completed the application should produce an output.

The information for the six different steps has been adapted using the Globus Alliance: GT4 Tutorial pages, condensed sufficiently to how the pathology application will operate. Reference: (Globus_Alliance, Globus toolkit, 2010) All source code is available in Appendix H.
The system enables the user to access the application through a basic login procedure. After the user/developer has logged into the system they can access and deploy the virtual machine depending upon the user privileges in place. Essential to the cloud test-bed it initialises the Nimbus VIM and this in turn enables/acquires the necessary data with regards to the virtual infrastructure of the client VM image. Once it has obtained the detailed specification for deployment and the application resource demands, it starts the virtual machine on the cloud test-bed through the Nimbus infrastructure.

3.10 Deployment plan of application on Cloud Virtual Image/Machine

In order to host the application on the Nimbus infrastructure, it needs to deploy and utilise a virtual machine or virtual image. With the core of the development being conducted on the Sun VirtualBox platform which hosts the Globus GT4 virtual four node cluster, it seems logical to re-use this same environment as the virtual machine. With the Nimbus architecture constructed and already built from the original globus command infrastructure openly created by the Globus Alliance, its technical
abilities are supportive, almost identical to the virtual cluster used within VirtualBox. With this distinctive advantage to the cloud virtual infrastructure manager it would be wrong to deviate and re-deploy the application purpose built using globus, onto a non globus environment.

Once the preliminary implementation has been formalised the web service and or non-web service re-engineered applications would fundamentally apply the same procedure as conducted with the native VirtualBox globus environment, when deploying onto the Cloud TestBed. In essence replicating the program applications, but installing and setting up the virtual environment before hand to suit the original globus image originally developed on. Thus reducing the possibilities of further unforeseen complexities.

3.11 Summary
With the design completed, the implementation of the software deliverables will be reviewed and corrected beside the methodology that the project has adopted. This will aim to continually enhance the prototype and throughout the development life cycle.
Chapter 4: Implementation

4.1 Introduction
The implementation phase comprise of implementing new structural features into the current pathology application. Throughout the development the distributed computing paradigm presents new issues arisen from new functionality that has been introduced. Ensuring swift rectification of each predicament faced the combinational methodology that the project has adopted aims to sufficiently deal with each iteration of the development, testing the application progressively through the evolution of the prototype. The chapter follows the initial specifications created in the design chapter. Periodical testing throughout the project will be systematically dealt with, ensuring the new aspects are applicable to the application. Testing of the prototype is covered in the final section of this chapter.

4.2 Tasks Undertaken
The implementation phase of the project has seen a particular set of tasks that have needed to be completed. The implementation began with moving the nucleus detection application onto the Sun VirtualBox. As the projects aims were of re-deployment rather than re-development and the existing application retaining all the required functionality it did not need completely re-designing. With the nucleus detection application currently under research at the hospital forming a small part of a much larger 3D reconstruction application it remained from a design perspective paramount to retain the current desired functionality that the pathologist were using for diagnosis of patient image data. Having migrated the application on the Sun VirtualBox globus virtual cluster, the first priorities were to investigate and install the missing libraries dependencies the application used. Once the application libraries were setup and verified the next tasks involved understanding and comprehending the globus GT4 tutorial. Having completed this focused shifted onto the standalone java web service and the using the globus GRAM java web service development. In between the different service deployment approaches for the application a series throw away local java application prototypes were developed. This local procedure enabled me as the developer to ascertain and trail out new functionalities that were either incorporated or removed from the final implemented solution produced for use on the cloud.

The remainder of this chapter aims to sufficiently detail the different tasks that have been indicated.

4.3 Globus Training
Comprehending the Globus GT4 Tutorial, identical to the design outlined in section “3.9.2 Java Service using GRAM”, is relatively core to understanding the principals and the techniques required to re-deploy any type of application, although be it in a grid environment. The tutorial serves two purposes utilised by this project, firstly it provided the necessary skills to be learned and adapted,
secondly its service tutorials becomes the template in combining and using the nucleus detection file as an extension of the already deployable service.

### 4.3.1 Sun VirtualBox

Apart of the training phase required by the GT4 Training material on the Globus Alliance website required a functioning Globus environment. To address this issue a fully functioning Globus environment had to be firstly installed using VirtualBox. VirtualBox is a GUI front end emulation environment, supplying virtually any operating system the ability to be deployed as a virtual machine, given the virtual image initially used. Its virtualization setup however does not provided external graphical support if given a command line based operating system, such as the four node grid cluster using a Debian UNIX 64 bit architecture installed for the re-development for the application. This hindered the ability to invoke and display any graphical orientated application, whether it is a local application to the UNIX operating system or any locally installed application, such as the pathology C++ file, invoked on the command line infrastructure.

### 4.3.2 Four Node Grid Cluster

The project required familiarisation with the Globus GT4 environment. Thus the project used the Sun VirtualBox GUI, which provided the medium to initialise and install a virtual grid four node cluster. The VirtualBox environment interface is capable of emulating the installed virtual machines, operating systems through its virtualization interface, but limited to a text based command line platform. Understanding and working solely in a Globus grid environment presented an ideal opportunity to both familiarise personally with a grid platform, almost similarly mirrored in the Nimbus VIM on the Cloud. With the project aiming to re-deploy the application onto a Nimbus VIM, which uses pre-dated Globus GT4 commands and OGSA standards, not only is a key aspect of the project, but enables clear transfer of skill sets. When using the Globus Virtual Cluster, virtual nodes are registered with the head node globus01. The virtual network operates on the subnet 10.0.2.0/16. The other three virtual nodes globu02, 03 and 04 each on the virtual network can be accessed via SSH from the host machine. Each of the four virtual nodes once installed provides three users profiles:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>Installation of additional software</td>
</tr>
<tr>
<td>globus</td>
<td>Deploying web services, configuring Globus</td>
</tr>
<tr>
<td>globusclient</td>
<td>Client web services, submitting jobs</td>
</tr>
</tbody>
</table>

Table 1: Globus User Profile
4.4 Pathology Application

The pathology application had to be first migrated and copied across from the University of Leeds computing network. This was achieved through SSH and SFTP commands via the host virtual node on VirtualBox and accessing the Universities server to copy the application. Having obtained the nucleus detection application and before attempting to apply the design elements listed in the previous chapter, it remained to invoke and execute the application locally without any alterations. A key factor to attempt to engage the application locally, presented the Globus environment needing to install missing GUI, numerical and text based binary libraries. Using the UNIX “ldd ./nuc_det_server” command it enabled to discover the missing libraries. Using the “root” super user on the host virtual node, having the ability to install missing software, enabled the application to find and install 50% of the missing functionalities. However, with the discovery of older libraries either no longer available or supported, just invoking the application became a major hurdle. After consulting and even personally e-mailing one of the original users of the application Derek Magee, it was suggested that the University systems the application stored on still possessed and used the missing binary libraries. Using the University of Leeds 64 bit architecture library document (lib64), it became apparent that the missing older libraries were still active on these systems, thus were promptly copied to the Globus environment. Rectifying this problem, a time consuming operation, once complete enabled the virtual node to promptly compile and successfully initialise the application producing the necessary output, but minus the graphical display of the retrieved patient image due to the virtual environment being text based. With the application able to run successfully and foundation functionalities located, the various other design methodologies continued.

4.5 Java Web Service

As suggested in the design chapter the project explored to notion of re-deploying the application using a purpose built web service. With the Apache axis tool emitters of java2wsdl and wsdl2 java providing an automatic client and server “.class” files used by a web service, it provided the easiest route to development. In order to use these emitters, the nucleus detection application had to be wrapped into a java file first. The main approach undertaken to create the java application was foremost to retain the original application parameter invocation, so no modification would be made to the original coding. Using a throw away prototyping method, quick applications were created eventually leading to a better solution in each successive iteration. Using this approach the newly created application (pathology.java) uses the Java RunTime event to pass local parameters required by the pathology application. Effectively the prototype program situated in the same directory of the original nucleus C++ program, used the command line UNIX interface to interact and call external applications, enabling clear route of access to the file being called and all output streams to be rerouted back through the new pathology.java application, via the local UNIX globus bash shell. Incorporating a new application, java based provided the development to quickly progress in using the
Apache tools to generate the required WS Description Language file. With the ability to invoke and use the pathology.java application, but having to supply the full parameter statically via the program, it limited the scope of program being able dynamically alter the string variables to use and supply new parameter options. An immediate option to solving this solution was to automatically enable the user to supply the full file parameters as command line arguments, identical to the nucleus detection program already exhibiting this method of interaction (See Figure 6 Command Line Invocation of the nucleus detection application). Avoiding the issue of just merely wrapping the original C++ program with a java program and sustaining the same interaction method proved to be a limiting factor when using the Globus environment. With the Globus nodes capable of deploying a grid web service more efficiently, through a RSL file the pathology.java program was modified to include this new route. The use of a Resource Specification Language (RSL) file is a language neutral xml document, specifically design to use the GT4 GRAM commands to deploy and submit jobs to the globus container.

Having completed the initial structuring of the code needed to execute the application the next steps involved using and supplying Globus resource GRAM commands via the working pathology.java application. However issues fundamental of the application not being able to accept and run the local GRAM commands, unfortunately ended the development of creating a web service through this standalone suggested route. Major concern with registering the local pathology.java application with the service registry in a Globus orientated environment was the biggest concern, not supported with the Apache emitters.

Appendix H: Source Code of the report contains the static java program originally created (pathocall.java version 1.1), and the adapted version using the local GT4 GRAM command structure (pathology.java)

4.6 Prototype Evolution

4.7 Globus Java Web Service – Using GRAM

With the development catering to use GRAM 2 a non-web based application deployment approach it seemed ideal as the application reflected a non web based setup already. Beginning the development in GRAM 2 required the project to step further back than the already conducted training using the GRAM 4 (GT4) tutorial. During the preliminary training required to use the GT2 toolkit it became evident support documentation not being available on the Globus website. Reviewing that the training earlier conducted was sufficient enough to adapt the existing GT4 tutorial files used previously, the development swiftly abandoned and moved directly to the GT4 path.

Using the design methodology proposed for a GRAM based implementation the development moved focus to using the Globus GT4 tutorial files as a template to construct and deploy the pathology
application. Having already created the application and generated a comparable WSDL document via the Java Web Service phase earlier, this provided a simple adaptation to be included into the Math.wsdl document to factor in the missing new functionality that the service would host. Creating and using the GRAM commands followed exactly the same procedure outlined in the design chapter for the Globus GRAM web service. Having altered the WSDL file and after various compilations and debugging throughout the development added to the existing service, attention turned to invoking the application from a client perspective and the remote server side.

To solve these issues using the already existing MathService_instance files from the GT4 Globus tutorial, the client.java file and the newly created pathology.java file produced in the Java web service section were simply merged, producing a new file named ClientC.java (Located in Appendix H: Source code of the report). Using this formulation retained the original MathService_instance tutorial service to the remote service, but simultaneous incorporated the new methods needed to invoke the pathology application. Having addressed the client side of the application the same procedure was applied to the server side of the tutorial service files being altered, adaptations were made to the MathService.java. A problem of the MathService.java unable to allow the user to enter the grid proxy password to generate and deploy a working proxy meant the pathology.java file functionality was split between the client and the server side. The client upon invocation would call the MathService.java service, but rather than the MathService.java bypassing the password the client incorporated this feature. This enabled the password of the proxy to be manually entered through the command line interface, but progressively enabled the MathService.java server side to call the application and then destroy the proxy upon completion. Another issue that had to be addressed was how to invoke a remote service capable of in turn invoking and running a local application. Solving this dilemma came through the use of using a Resource Specification Language (RSL) capable of using local GRAM command and supporting the use of a Globus grid orientated proxy generation and destruction. The RSL file is a separate file containing XML Markup, again language neutral in construction that would effectively supply the nucleus detection application with the required parameter to call the service.

In order to prove the concept of using an RSL file, another pathology.java file was modified using custom created runtime calls to the Globus interface to try and run the nucleus application. With the development constructed and using the “globus” user account on the VirtualBox cluster, it emerged it was not setup with the right grid user proxy credential needed to deploy use a service and submit jobs.

Overcoming this issue again came through personally having to think of new solutions. Using the “grid-cert-request” commands it was first decided to try and create a purpose user grid credential. Using this command a new grid credential created the missing certificate, but still failed to deploy the grid service and register as a valid certificate. Facing more delays to an already tight schedule, I decided to migrate the “globusclient” user account grid credentials across to the “globus” user
account, due to that fact of the “globusclient” was already registered and able to submit and use client web services. Upon migrating these credentials to the “globus” user account it proved to be a decisive action that enabled the “globus” user account deploy the remote GT4 services and produce the required output. Trialling and testing this with the adapted pathology.java application thus produced and program capable of starting a grid proxy certificate, deploying the nucleus detection program via calling a RSL document, staging the file to the remote service container and then retrieving and storing both the command line data and the Out.xml document produced from the server operating the application algorithms. Staging the file in its own functionality provides a TCP/IP protocol like structure, whereby the GRAM command used to deploy the job submission would from start to finish stage the file in, hold for a response, supply the output and clean up the transmitted data streams on both ends of the service.

The purpose for retrieving the output from the nucleus application was to contain the calculation and algorithmic data needed for the application identifying and enabling diagnoses vital for the pathologist to compare and validate problem areas. Once pathology.java application finished calling the submitted job the staging out process initiates and closes down the connection constructed, this in turn is then followed by the GT4 environment destroying the grid credentials using the “grid-proxy-destroy” command.

![Globus Resource Allocation Manager - State Diagram](image)

High Level Overview of GRAM Job Submission Procedure

With the local pathology.java application capable of operating and providing the medium to use the local command structure within the Globus environment, the functionality was separated into the client and server service side files, contained in the already created MathService_instance tutorial documents. As discussed in the design chapter to the various steps required to invoke the java web service using GRAM remotely, using and compiling it, is identical to the process applied for the final prototype of the application using and deploying a remote web service.

Appendix: G Command sequence to start and use the service illustrate the commands needed to deploy the service. Establish a working remotely deployable service the application can is now migrated to the cloud.
4.7.1 Grid Service GRAM Staging in/out a job

The diagram illustrates the stage of when the java grid service application calls the local GRAM commands and executes the application job. Each tier represents the different levels of communication needed to first setup the grid proxy, execute the Globus run command to call the custom Resource Specification Language (.rsl) file and invoke the pathology application. Once the file is invoked, the file is staged into the service until the program completes its desired functionality. Upon completion, a command is sent to the proxy to start the procedure of staging the output stream of the application called and print to the “stderr” and “stdout” files, allowing all data to be persistently stored for the user to access. Finally, once the staging out has completed, the application grid service destroys the proxy.

The following image has been taken from the Globus Alliance Website. Reference: http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Approach.html
4.8 Moving it to the Cloud - Nimbus

4.8.1 Deploying a Virtual Machine Image
Similar to the setup required for the VirtualBox environment the cloud virtual infrastructure manager Nimbus, uses and deploys virtual machines. With the University of Leeds cloud TestBed only hosting the Nimbus VIM, it remained for me as the developer to locate and install a virtual image file suitable for the application to be incorporated within. After careful consideration, the Virtual Machine (VM) image selected to host the application, was the generic Globus virtual cluster VM image used in the VirtualBox initial setup. This selection proved to be the most sensible option as it would limit any potential functionality issues dramatically already foreseen and dealt with during the prototype creation. Having already familiarised myself to the working environment used on the GT4 training and developed the application to co-inside and operate using this environment, it made perfect sense to continue this.

4.8.2 Virtual Machine Image Settings
After acquiring the (globus.deb.cfg) virtual machine image it was stored in a new directory (/home/nimbus/Bhupinder), the preliminary settings on the Nimbus VIM needed to be prepared. To reflect the environment simulated used in VirtualBox and the four node cluster, the sample-workspace.xml document (Appendix H: Source code), a configuration file used to define the virtual image location, diskspace and hardware allocation, were altered on the Cloud TestBed. Changing these files is essential for the Nimbus virtual workspace to identify and maintain a persistent state, such that any alteration conducted to the virtual machine would remain in effect upon shutdown and start-up made by the user.

4.8.3 Using the Virtual Machine
To access the virtual machine, first the globus.deb.cfg image needs to be deployed. Using the “sudo xm list” command initially identifies of currently deployed virtual machines, if any on the TestBed. Using the “sudo xm create ./globus.deb.cfg” command enable the Nimbus VIM to deploy and manage the virtual machine. Having the settings and the virtual machine deployed, the user application needed to be re-deployed into this new virtual machine. With new VM image deployed and using the generic environment as with the four node cluster, the three user login credentials root, globus and globusclient, all with the same functionality as before available in the new globus.deb.cfg image used. Using “ssh globus@10.0.2.1” enables you to log into the virtual machine (see Table 1: Globus User Profile). Once logged into the virtual machine the next phase to re-deploying the application was to replicate the initial setup procedure applied to the four node grid cluster. This meant the new machine image required all the missing libraries to be re-installed and the missing grid proxy credentials to be rectified. Effectively the same objective undertaken in the VirtualBox, that provides support for the application had to be applied.
Upon completing the re-installation procedure in the globus virtual image, the application source code files were merely packaged up and migrated across. Again due to the virtual machine image used to develop and run the application being text based orientated, the cloud virtual infrastructure manager reflected this.

The exact command sequence and to accessing and executing the application on the cloud can be located in the appendix of the report under the section Appendix H: Source Code.

4.9 Testing
A continual testing of individual software components when developed, involved evaluating and reviewing the software prototype identifying problem situations. With the project methodology using an evolutionary development strategy to systematically progress between each iteration, a standard set of small scale testing procedures were in place to deal with erroneous defects with the source code, filtering out unintended functionality. With each characteristic continually incorporated and verified this ensured the software prototype deliverable retained and prioritised further development attributes.

To ascertain the prototype produced throughout the development phase was correct, a series of test were formalised and conducted. Involvement and creation of small scale throw away prototyping were the heart of the project quickly identifying and defining pathways for the development to maintain. This method vastly improved and distinguished clear directions to take, with the minimum requirements and possible extensions reflected upon within each small prototype using the preceding design phase. Although the design phase in the initial prototypes did not provide an accurate correlation to what was actually produced in the UML models, it still enabled identification and analysis of the user interface and output documents produced needing to be factored into the prototype.
Other software functionality vital for the application to physically compile and initialise including downloading and installing missing C libraries used for both the development of the remote web service and the original application.

Testing the reliability of the prototype and persistency to recall the algorithmic detection calculations produced by the nucleus detection program, the web service was deployed and invoked using different nucdetserver.rsl file parameters, ranging form small to large image slides and varying the image patch size, the results of which are included in Appendix C.

The efficiency and preserving maintainability of the prototype code again remained in effect throughout the duration of the prototype evolution, with a cyclic software development life cycle any unsatisfactory lines of development were removed after careful consideration.

4.10 Summary
This chapter has discussed in detail the implementation of the distributed software prototype. Each part of the chapter has explained the various development phases involved to create and produce the remote web service that finally emerged, mapped to Chapter 3, the design, in a one to one mapping especially with the grid web service development. This chapter has also outlined the how the systems has been tested. Chapter 5 will examine the prototype produced in the implementation phase of this project and objectively evaluate it based on set criteria.
Chapter 5: Evaluation

5.1 Introduction
This chapter discusses the evaluation criteria used to evaluate the final software prototype against minimum requirements and any potential improvements or future work considered. The efficiency and the usability of the software performance are also examined along with the methodology the project adopted. The final section of this chapter discusses whether the aims have been achieved.

5.2 Evaluation Criteria
The evaluation criteria of the project have been created to consider the technical and user perspective, cloud application and evaluating the project as a whole. The use of functional and non-functional requirements establishes the basis to which the project will evaluate certain key criteria. The functional requirements are composed from the minimum requirements the project initially set out to achieve, as illustrated in Chapter 1 Introduction. The non-functional requirements are defined to identify the objectives of the prototype performance, efficiency and other external objectives whether the complexities foreseen in the surrounding original application have been reduced.

5.2.1 Technical Evaluation
To ascertain whether the final project prototype had a specific influence on the performance of the application three criteria were subjected to the testing of the original and re-deployed application. These criteria involve looking at the elapsed time taken for the application or “Real Time”, “User Time” which calculates the total CPU time taken in seconds for the process to use in user mode and finally the System time which calculated the CPU total time taken for a process in kernel mode. To ensure the both software applications were subject to a fair testing structure, the University of Leeds, School of Computing, computer laboratory machine selected was the cslin070 machine, using an Intel Quad CPU 2.66GHz, 4GB memory on a 64 bit Kernel. This hosted the application re-deployed locally and provided the medium to contrast the applications performance mirrored with the re-deployed prototype seen on the cloud. The cloud physical machines supporting the Nimbus VIM are called Testgrid 3 and Testgrid 4, each have an Intel Xeon Quad CPU 2.83GHz, 4GB and supports 64 bit kernel. Each subsequent test conducted for the original application and the Cloud TestBed application used this machine, although it is worth noting due to the machine being a networked computer it was still available for other users on the computing network to run processes in the background. The types of tests that have been conducted on both the original and the re-engineered application predominantly look at performance. The typical tests range from:

- Selecting at least five different images sizes and supplying both applications with the identical parameters and calculating the respective elapsed, user and system time taken for the application to complete.
- Invoking the applications and the selected image files at least ten times to gain a sufficient representation of the average and the standard deviation for comparison purposes.
- Selecting the image file 81275.svs a small 1.41GB image and subjecting it to the different image patch sizes form 500 x 500 up to 1000x 1000, to identify any changes in performance gained or lost due to extra algorithmic calculations.

The following are the results from applying the testing criteria to both applications models.

**Definitions of the measurements**

**Real Data:** Elapsed real time taken in seconds.

**User Time:** Total number of CPU-Seconds that the process spent in User mode.

**System Time:** Total number of CPU-Seconds that the process spent in Kernel mode.

<table>
<thead>
<tr>
<th>Overall User Time Mean</th>
<th>1.41GB</th>
<th>1.54GB</th>
<th>1.76GB</th>
<th>1.98GB</th>
<th>2.15GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1.080</td>
<td>0.914</td>
<td>0.514</td>
<td>6.015</td>
<td>2.846</td>
</tr>
<tr>
<td>Cloud</td>
<td>1.376</td>
<td>1.402</td>
<td>1.388</td>
<td>1.424</td>
<td>1.423</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overall Real Time Mean</th>
<th>1.41GB</th>
<th>1.54GB</th>
<th>1.76GB</th>
<th>1.98GB</th>
<th>2.15GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1.904</td>
<td>1.736</td>
<td>1.278</td>
<td>6.874</td>
<td>3.738</td>
</tr>
<tr>
<td>Cloud</td>
<td>17.946</td>
<td>24.253</td>
<td>28.539</td>
<td>13.112</td>
<td>12.998</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overall System Time Mean</th>
<th>1.41GB</th>
<th>1.54GB</th>
<th>1.76GB</th>
<th>1.98GB</th>
<th>2.15GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>0.058</td>
<td>0.058</td>
<td>0.056</td>
<td>0.071</td>
<td>0.064</td>
</tr>
<tr>
<td>Cloud</td>
<td>0.073</td>
<td>0.065</td>
<td>0.093</td>
<td>0.062</td>
<td>0.067</td>
</tr>
</tbody>
</table>

Table: 2 Overall Mean Time Taken for the varying image files to be invoked.

![Overall User Time - Original Vs. Cloud](image)

**Figure 11: Overall Mean User Time**
From the results of invoking both applications it is clear that from the elapsed time taken (Figure 12 Real Time) the cloud re-engineered prototype suffers a prolonged period of time, longer than the original application. Although there seems to be a dramatic performance increase by the new application, there are a number of intermediate steps that have extended the original invocation of the nucleus detection program. Having encapsulated the pathology.java application and combined into globus GT4 GRAM command the ClientC.java once invoked requires the user to enter the grid proxy password. Only until this manual action is preformed, it is then the new pathology application is able to call the remotely deployed web service, this is a limitation to the deployed cloud application which adds to the time taken for the service to run. For a fairer comparison I would have got rid of all of the security issues, so that I wouldn’t need to enter the password. But at the end of the day this is how it works.
However with the new application prototype elapsed time taken significantly longer than the original program, the user CPU time taken by the process almost on each image file dramatically out performs the original program (Figure 11). There is a discrepancy of the original application taking longer at the 1.98GB file used. Due to the machine being used to conduct the tests being freely available on the university network, it could have been other processes simultaneously using my machine processing capabilities, thus affecting its performance as clearly seen in the results. The factors of the User process CPU time taken indicate the applications ability to utilise and spread demanding system resource across the cloud. Due to the nature and setup of the cloud hosting the Nimbus VIM, any resources or application invoked on this platform dynamically spreads to the load of the process requiring specific resources. With the original program performing on par, even better than the new application, but with the new application consistently performing and scaling resource demands dynamically relatively less in comparison, it is a clear indication to the foremost objective that the project set out to prove. The objective being able to re-deploy the original application onto a more dynamically scalable orientated architecture.

Considering the mean system time taken for both applications, again the indications are that the original nucleus detection application performed better. However this indication although apparent is too small of a difference with 0.020 seconds being the significance.

### 5.2.2 Increasing the Computational load

To gain a better understanding to both application capabilities evaluating the programs based on scaling the image patches was conducted. With the original application suffering from bottlenecks in the server side of the architecture unable to perform effectively, it proved to be an ideal test to apply.

<table>
<thead>
<tr>
<th>Size (GB)</th>
<th>500 x 500</th>
<th>800 x 800</th>
<th>1000 x 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>Original Application</td>
<td>1.904</td>
<td>15.244</td>
</tr>
<tr>
<td></td>
<td>Cloud Application</td>
<td>17.946</td>
<td>28.637</td>
</tr>
<tr>
<td>User</td>
<td>Original Application</td>
<td>1.080</td>
<td>13.885</td>
</tr>
<tr>
<td></td>
<td>Cloud Application</td>
<td>1.376</td>
<td>1.394</td>
</tr>
<tr>
<td>System</td>
<td>Original Application</td>
<td>0.058</td>
<td>0.110</td>
</tr>
<tr>
<td></td>
<td>Cloud Application</td>
<td>0.073</td>
<td>0.063</td>
</tr>
</tbody>
</table>

Table 3 Raw data results of the changing image patch sizes
Figure 14 Elapsed time taken

Figure 15 User time taken varying image patch

Figure 16 System Time Taken
The result of the experiments show the cloud application performance again has a slower service compared to that of the original application, this is clearly seen in figure 14 the elapsed time taken for the application to run. However as this set of experiment were to ascertain whether increasing the computational load on the both application would effect the performance, there seems to be a indication that the cloud application although slower in comparison to the original application on the elapsed time has a significant performance difference in the CPU level aspect of the system. Upon closer inspection the cloud application in figure 15 out performs the original application substantially. Again as seen in figure 16 the application re-engineered perform better with much higher computational loads. Although this is a significance that support the theory behind successfully deploying application onto the cloud, its performance advantage could lay directly with the underlying physical hardware the Nimbus VIM interacts amongst. With the Nimbus VIM hosted directly upon two Dell Commodity Servers each with quad core hardware, the distribution of the process work load could have been delegated sufficiently to these CPU’s, most likely being the explanation to the significant performance that has been outputted when the application was run specifically on the cloud. Although the suggestion of the cloud architecture hardware contributing to the performance advantage seen in the results, it does prove that globus virtual machine deployed using the Nimbus VIM does partially use parallelization to spread the load of resource demands. This does support the dynamic nature of cloud environments virtual machine able to contribute to resource fluctuations on demand.

5.2.3 Issues of Overhead in VIM
A key consideration is the overhead produced when using the virtual infrastructure manager. With Nimbus being deployed to mange the virtual machine, in itself it will create some overheads because it is a layered architecture, effectively becoming extra software. This would be mostly be reflected in the visualization of the application environment, with the virtual operating systems calls being exchanged in the environment, this would then be translated through the abstraction layers of Nimbus to procure the use of the physical resources needed for the virtual machine.
5.2.4 **User Evaluation Plan**
The user evaluation plan was primarily considered to determine whether the user interface solution produced was adequate and still functionally viable for the pathologist to justify its use. The aim was to eradicate undesired functionality and conform to a more user friendly environment setup. The proposed idea was to, upon producing the prototype sufficient and capable of proving the required functionality originally intended, but deployed through a web service, if it was as easy to operate and as transparent to the end user. This would have involved personally taking the solution back to the pathologist at the university hospital and allowing for a sequence of similar parameter calls on image slides using both the original and the re-deployed application. It was hoped that this procedure would have enable the prototype become more tailored to the pathologist unique perspective on using the system. However due to time constrains and the prototype eventually implemented and tested according to a tight schedule this section of the evaluation remain to be. It would have provided indications of the usability of the program and how easy the system was effective compared to the original. If I had time I would have asked the user to check this is the list of criteria:

- Easy of using the new application
- Speed and execution time of the application performing against the original application
- Further functionalities that are not seen in the current system that may be viable to incorporate into the new application.
- Accessibility of the application
- Potential improvements, such as features within the current application that they might want to remove or have re-designed to provide a better interface to the application.

5.2.5 **Development Evaluation**
This section concerns the software development of the system and the application produced. Looking at the various middleware toolkits involved, architecture of the systems used and the overall development outline. The two major factors that have attributed to the implementation prototype being able to function were the training phase of the project and the re-deployment architecture of Nimbus. Considering the development produced a proof of concept, it is hoped that an in depth discussion of this project development will provide some new information/documentary evidence of using cloud computing, still a relatively new paradigm, and should be the focus for new research being conducted.

5.2.6 **Globus GT4 Middleware as a General tool**
Early in the project it was considered the implementation was to use and train using the Globus GT4 middleware. The selection of using the Globus middleware was mainly due to the fact the university has previous experience with the development platform and it covered the relative basic command sequences seen in new cloud middleware infrastructures, for example the Nimbus VIM.
The training element of the project was mainly used in conjunction with the Globus GT4 tutorial, available online in the Globus Alliance webpage. The training material itself was relatively in depth covering Globus client and server side implementations and accompanied with detailed in depth information and diagrams of how a specific services were deployed with example code output commands.

Using the tutorial the basic example mathservice enabled me to understand and even test the applications remote web service abilities through deploying it onto the Globus virtual image. The material covered basic math addition and subtraction method calls to the application being invoked, producing a simple output interaction, to the higher factory service models well beyond the scope of the project development intensions. The source code provided by this tutorial became the template to build and re-engineer the nucleus detection application into a web service, without this information I feel the project would not have progressed as sufficiently.

Having completed the training the development focused its attention turned onto the Globus Resource Allocation Manager or GRAM toolkit which is a webs service tool formed and incorporated apart of the Globus virtual node. A problem with the example mathservice tutorial was the fact of service were created and built into the methods of the server and client coding making it difficult to include the nucleus detection application as a service. However looking further into the GRAM tool apart of reading and understanding GRAM uncovered its ability for applications to be automatically invoked, using the Resource Specification Language (RSL). This enabled the project to discover and incorporate a better deployment method, not previously considered. With the GRAM documentation on level par to the Globus training material this functionality was easily understood and became apart of the implementation bringing the prototype one step further.

Having progressed with the development the project overcame issues of having to make custom modifications to the existing mathservice WSDL documents using sections of previously implemented standalone web service PortType, client application source code, service side source code and generate a .rsl file for the service to call using the GRAM commands (See Implementation chapter for full details and Appendix H for the source code). As a general tool the GT4 middleware had issues, but with a sufficiently detailed tutorial and documentation when located it became as can seen in the development as an invaluable aspect.

5.2.7 Nimbus
A considerable down fall for the development was the Nimbus training which was restricted due the poor training material that was available for the deploying and using it as a virtual infrastructure
manager. Although the Nimbus VIM is constructed and developed from its pre-dated version of Globus GT4 its lack of documentation became huge problem. This was only one of many problems that the Nimbus VIM proved to create. With the university cloud hosting and deployed Nimbus VIM it was still error prone and was only able to deploy a single virtual machine. This hampered the testing phase only being able to deploy and use a single image. Having had problems with configuration documents and other functionality not becoming immediately available in the Nimbus VIM, it remained in a fragile state throughout the development. This caused huge concern when the re-deployment came to using the cloud and fulfilling the minimum requirement. Due to the open nature of the software and it still being a relatively a new virtual infrastructure manager there is still no clear documentation available to fully utilises its capabilities. Having experienced these issues first hand the suggestion would be to avoid using Nimbus, and research more tested and reliable infrastructure managers. The only favourable aspect to using the Nimbus infrastructure was its backward compatibility with the globus GT4 infrastructure using OGSA standards, if it was not for this the functionality the development would have suffered an almost unrecoverable perspective. Another more prominent factor that slowed the development was the text only interface, when developing the client and server side coding continually having to type to navigate, edit and migrate files across the system became tedious. It would have been useful for the Nimbus VIM to incorporate a more GUI orientated interface, similarly again for the Globus GT4 virtual machine which again was command line based.

5.2.8 Other Projects
As well as this project re-deploying the nucleus detection application onto the cloud Nimbus VIM, there was another project similarly re-deploying the same application but involved in using OpenNebula. With both projects involved in re-deploy the same application, it proved to be an ideal opportunity to uncover the efficient between both systems. It was intended the OpenNebula project researched and developed by Kirk Godber, would have enabled both projects to reflect and characterise the development and project experiences. Unfortunately this did not transpire as the development faced in OpenNebula similarly encountered familiar issues when it came to constructing the service orientated development in the globus virtual node cluster. It was intended that both virtual infrastructure managers would have proved invaluable performance opportunities to explore efficiency, comparing the original and the newly developed solution between both Nimbus and OpenNebula. In addition to this it would have provided both projects to reflect on individual experiences, rather than from a user initiative.

5.3 Minimum Requirements
To evaluate the project as a whole the minimum requirements need to be reviewed and linked directly to progress made. The question of whether or not the deliverable of the project meets the minimum
requirements needs to be answered with evidence to support the conclusions that are made. The minimum requirement where defined as the following:

I. Design suitable pathology application for implementation on a cloud.
II. Develop the product on Nimbus middleware.

The first minimum requirement has been achieved with the original pathology application being re-designed and encapsulated into a java web service. Understanding the original pathology application needed to retain its internal algorithmic calculations, the source code for the application was not modified, but in order to invoke the application through a new programming environment a pathology.java application was created. This effectively encapsulates the original application, keeping its functional abilities intact. Having created the new pathology.java application a java grid web service was produced and made operational firstly in the Sun VirtualBox architecture, using the globus virtual nodes. The second minimum requirement was met by the globus virtual image hosted on the Nimbus VIM and with its capabilities to support GT4 commands, was able to deploy a remote web service and interact with the service to produce the required output files.

5.4 Improvements and Future Work

There are two complexities with the histopathology application, firstly parallelization of the application processing capabilities and secondly the data management of the patient digital image slides. Being able to deal with these two aspects would first of all need a bigger cooperation cloud infrastructure with more pooled resources at its disposal, greatly improving the capabilities to utilise and configure virtual machines and widen the scope of experiments, not currently available in the University of Leeds Cloud TestBed.

Parallelization

The project as a whole has partially addressed this situation with the ability to deploy a globus virtual machine capable of dynamically allocating resource demands to the underlying infrastructure. Although due to the limitation of Nimbus only being able to run a single virtual machine, it restricts simultaneous creation and deployment of further virtual machines. However having the ability to create more virtual machines per each additional user that uses the system to invoke the application would spread the load across the entire cloud, greatly enabling the parallelization being addressed.

A more demanding aspect to the parallelization would concern the source code of the application, and how to effectively parallelize and distribute of creating the model image slide. The creation of a model sees the application retrieve and process the image slides in linear sequence. All this is supplied to a single process or single core of a multi core CPU, which deals and conducts the algorithmic calculations and applying the co-registration of result projected onto the before and after display of the slide images. Altering the source code and re-designing and adding MPI (Message Passing
Interface: MPI permits applications and program to synchronise with other processes through sending and receiving message through separate address spaces) coding would enable the application to deal with and delegate processing commands to multiple processes simultaneously. Another way could be trying to get the code parallelised in such a way there is minimal interactions between the tasks, moving towards a highly parallelizable or embarrassingly parallel application without necessarily having MPI, but just needing these tasks to run on specific processors at specific times.

Data Management
Another factor in the application is the use and retrieval of images slides from remote locations. Using images potentially gigabytes in size at lot of network latency issues emerge for the image slides to be transferred across and supplied to the application. A solution to this issue would be to store the image slide more closely to the virtual machine images, more so ideally on the cloud infrastructure itself. It is worth noting although the image slides would reduce the latency issues; the cloud would need to integrate policy constraints (e.g. confidential compliance with patient data records).

5.5 Conclusion
In conclusion the project went according to plan, discounting the fact of the unforeseen circumstance that affected the project deliverables slightly. Having at the start of the project decided to break down the project into individual eight work package deliverables (Deep Grid Literature Review, Grid Middleware Training: Globus GT4, Deep Cloud Literature Review, Cloud Middleware Training: Nimbus, Pathology Application, Re-deployment of Application onto Cloud, Evaluation, Write Up) this enabled the project to identify and allocate time necessary for each phase requiring most attention.

Although not all deliverables were straight forward, the training element conducted using the globus middleware environment proved to be the most problematic of all the phases. This was a continuation with the cloud middleware Nimbus, as it was originally evolved using the globus GT4 command structure. With this being a serious issues in the development personally experienced, it is worth noting that using freely available open software does not always produce reliable environments on face value. Especially with any further projects looking to explore and use globus GT4, the time spent developing in this field would be better spent locating and using more refined and tested middleware. The Nimbus VIM and the cloud did present problems, but having over come theses issues and used the cloud I still believe the cloud computing paradigm although not fully proved and deployed in a commercial sense, will with the right technology benefit and fulfil a demand already seen in both business and research models needing and using elastic, scalable, and pay-by-use infrastructures.
References


62
Treanor, D. (2010 (Private Communication)). Pathology Meeting at St James University Hospital Leeds. (Final_Year_Undergraduates_(Myself_Included), Interviewer)

Appendix A: Personal Reflections

The scheduling of the project has not been too difficult, I have effectively managed my time and organised/prioritised this around my other commitments and responsibilities. Although looking at the initial project plan (Gantt chart) it is clear that some scheduled tasks were due for completion a lot earlier, there have been a number of technical issues that delayed certain phases. Whilst this may seem like an absolute concern for the project, I intentionally overlapped and condensed the project schedule such that I was able to achieve imperative objectives much earlier. This allowed me to factor in and account for unforeseen issues that were likely to occur to a certain extent.

Some technical issues were predominantly apparent in the training phase of the project. Discovering that the Globus GT4 toolkit required a 64 bit machine to run the virtual image, this delayed my training by almost two weeks. Although solutions were in motion to rectify this issue for me, noting that the time scale of the Globus GT4 training should have only taken single week to complete, I took my own initiative and obtained a laptop to solve the issues myself. Other minor setbacks were evident in mounting the Globus GT4 four node cluster images onto the Sun VirtualBox virtualization; however these problems were easily addressed and did not hinder the training phase any further.

Beginning the training section on the university cloud-test bed presented problems with the Nimbus infrastructure not functioning properly, due to a re-installation of Nimbus support controls. Although this was a problem there was sufficient material already deployed onto the cloud-test bed to enable me to at least familiarise myself with the system I was going to re-deploy the application onto. Having been supplied with some basic deployment scripts to engage and grasp the working of the cloud enabled me to map the code constructs that the cloud needed, vital when it came to re-deploying the application.

Meanwhile having had problems with the training aspect of the project I found myself applying the committed time, which happen to be somewhat idle now being utilised in other sections such as the continuation of deep cloud/grid literature review phase of the project. This facilitated the project to progress further and expose other later sections of interest more rapidly.

With issues of the cloud test-bed continuing it was suggested and agreed in a meeting that the pathology application be re-deployed onto the Globus images through VirtualBox. This new route of deploying the application as a grid service was a step back for the project. Although this course of action proved to be a wise initiative as the Nimbus VIM in my case had effective evolved from Globus GT4 toolkit. This meant that the brunt of the application development was going to take place in the Globus infrastructure, and having completed this it would be simply migrated across to the Nimbus infrastructure which conveniently still supported the Globus protocols and standard with minor exceptions.

Globus toolkit

Having agreed with my supervisor to deploy the application onto the Globus virtual cluster first, looking at the GRAM2 documentation became an issue with the Globus alliance webpage not having the documentation available any more. The GRAM 2 development plan indicated a non-web service deployment method that supported a C/C++ applications and programs, but with issues of merely trying to access the documentation became all too time consuming. To rectify this I personally sent an e-mail to the organisation to try and get some further information vital to the applications re-deployment onto the cloud. This enforced further delays to the re-deployment of the application becoming a reality; however after a week I received a reply from the Globus alliance organisation to confirm that they had resolved the issues. The process of deploying the application onto a GRAM2 toolkit would enable me to establish a working VM image to the extent which it would be transferred onto the VM image created on the Nimbus cloud infrastructure. Upon reading the GRAM 2 training material it became evident that the documentation was predominantly referring to older grid globus commands and clearly did not indicate specific development paths to take. Having wasted more time on this route it was abandoned.
This meant the development needed to follow and install using the recent GRAM GT4 tutorials as conducted in the initial training phase of the project. Developing using the newest version presented an ideal opportunity to re-deploy the application as a web service, but due to the complexity of merging the application required to create a web service tailored for a globus environment, this created a steep learning curve to use.

The lesson that should be made clear is that personally I had never done cloud computing developments, nor had I developed using the globus GT4 toolkit. It had been relatively been studied form a theoretical perspective apart of my degree. Having completed the project a clear indication that must be understood for any other students taking a similar course of action within their project, is never take open free software as guaranteed to work first and every time. This was unfortunately evident in the training and the re-deployment phase of the project. Another lesson that I have learned over due course of the project is that if you are unfamiliar with any aspect of the project make sure that this is resolved immediately. Having suffered technical issues that should have been dealt within days, took weeks to fix. This again strained the already tight schedule further.

A regret that I have that although when I tried to effectively manage my time and deliver the report on time was that the technical issues were not dealt with promptly. Although the project supervisor was able to solve certain simple issues, major technical blockages could not have been solved with an expert in the field. Luckily in my case the cloud testbed was managed and developed by a PHD student. Being able to refer technical problems back to the support person for the cloud assisted when problems became beyond my knowledge and understanding. Although problems were solved and the PHD student working on his studies there were a number of communication breakdowns again affecting the scheduled time. A personal reflection and lesson I have learned is you can never expect the un-expected until it happens, always make sure you have a backup procedure and factor time delaying and or unforeseen events into your project.

My advice to other students’ branch from the lessons I have learnt above:

I. Depending on the project you are undertaking, it is vital that the proportion of time allocated for each section is clearly thought about making sure that the project adheres to the marking scheme as a reference for the outline of your project plan.

II. The most invaluable information on a project will ultimately come from your project supervisor, making notes in the meeting will help you recall vital information even using a Dictaphone to record the meetings (with the permission of the supervisor). This is a tool that I personally used to great effect when unsure about certain information that may have been missed off my personal paper notes.

III. Try to create weekly objective to-do lists, or a personal planner/diary helping you to clearly define the set requirements for the forthcoming weeks. Also this will crucially assist you in producing a precise final project plan and highlight unforeseen issues that you may have forgotten about.

IV. A vital component to the project must account for unforeseen circumstances, always prepare for the worst the best you can. Special attention to aspects of development or any physical work created used or prepared vital to a project must be kept safe, either through backup procedures or means of security.

Largely this project has been a great experience and a key highlight to my university career, being able to apply some of theoretical aspects learnt in the university course practically to this project is one of many positive factors I will look back on.
Appendix B: Literature Review (areas of interest)

Literature Review Plan

The following document was used to outline the literature research, direction to sufficiently understand the knowledge to re-deploy the current pathology application in the project. Each heading of the sections depicts an area of interest followed by each subsequent bullet point indicating the aspects looked at.

Literature Review Area of Interest

Virtualization
- Advantages
- Disadvantages
- Purpose within:
  - Grid
    - Virtual Organisations
    - Remote object invocation
  - Cloud
    - Virtual Infrastructure Managers
    - Virtual Machines

Grids
- Architecture
  - Anatomy
  - Philology
  - Open Grid Services Architecture (OGSA)
  - Hourglass Model
- Application
  - Data intensive
  - Computing intensive
  - Service orientated
  - Physics - High Energy Physics (HEP)
  - Biology - Bioinformatics
  - Astronomy
  - Maintenance - Distributed Aircraft Maintenance Environment (DAME) and Broden
- Middleware
  - Commodity Grid middleware
  - Globus GT4
  - gLite (Briefly)
  - UNICORE
- Infrastructure
  - White Rose Grid (WRG)
  - TeraGrid
  - UK National Grid Service
  - Enabling Grids for E-sciencE (EGEE)

Clouds
- Architecture
  - Software-as-a-service (saas)
  - Infrastructure-as-a-service (iaas)
  - Platform-as-a-service (paas)
- Application
  - Software-as-a-service (saas) – Google Applications
  - Platform-as-a-service (paas) - Microsoft Azure platform
- Middleware
  - Academic solutions
    - Nimbus
    - OpenNebula
    - Eucalyptus
  - Commercial
- Microsoft Azure
  - Infrastructure
    - Public Cloud
    - Private Cloud
    - Hybrid Cloud

Pathology Application
- Local implementation of application source code (nuc_det.cpp)
- Remote application source code (nuc_det_server.cpp)
- Service Level binaries
- Server (Further Work)
  - Bottlenecks issues
  - Latency issues
  - Data Management
  - Parallelization

Project Life Cycles
- Rapid Application Development
- Evolutionary Development
- Incremental Delivery
- Verification and Validation Model
- Spiral Model
Appendix C: Testing Results

The following results illustrate the original nucleus detection application versus the newly re-engineered application deployed on the cloud TestBed. A varying range of images sizes have been used to evaluate to commonality between both systems. The data below and associated tasks were used to gather statistical data on how affective the system processes the image files supplied to the program applications. The values in the table are in seconds.

Real Data: Elapsed real time taken in seconds.
User Time: Total number of CPU-Seconds that the process spent in User mode.
System Time: Total number of CPU-Seconds that the process spent in Kernel mode.

Note: All the testing has been conducted through the school of computing machine, of csln070.

Original Application Raw Data Results
The following table results illustrate the original nucleus detection application performance results.

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Image Dimensions</th>
<th>Size (GB)</th>
<th>Magnification</th>
</tr>
</thead>
<tbody>
<tr>
<td>81275.svs</td>
<td>23976 x 21031</td>
<td>1.41GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Original Real Data</th>
<th>Original User Time</th>
<th>Original System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.752</td>
<td>1.078</td>
<td>0.056</td>
</tr>
<tr>
<td>2</td>
<td>1.767</td>
<td>1.083</td>
<td>0.053</td>
</tr>
<tr>
<td>3</td>
<td>1.952</td>
<td>1.091</td>
<td>0.051</td>
</tr>
<tr>
<td>4</td>
<td>1.973</td>
<td>1.082</td>
<td>0.054</td>
</tr>
<tr>
<td>5</td>
<td>2.026</td>
<td>1.082</td>
<td>0.061</td>
</tr>
<tr>
<td>6</td>
<td>1.774</td>
<td>1.075</td>
<td>0.062</td>
</tr>
<tr>
<td>7</td>
<td>1.795</td>
<td>1.081</td>
<td>0.055</td>
</tr>
<tr>
<td>8</td>
<td>1.825</td>
<td>1.077</td>
<td>0.061</td>
</tr>
<tr>
<td>9</td>
<td>2.026</td>
<td>1.080</td>
<td>0.060</td>
</tr>
<tr>
<td>10</td>
<td>2.146</td>
<td>1.075</td>
<td>0.065</td>
</tr>
<tr>
<td>Total</td>
<td>19.036</td>
<td>10.804</td>
<td>0.578</td>
</tr>
<tr>
<td>Mean</td>
<td>1.904</td>
<td>1.080</td>
<td>0.058</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.138</td>
<td>0.005</td>
<td>0.005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Image Dimensions</th>
<th>Size (GB)</th>
<th>Magnification</th>
</tr>
</thead>
<tbody>
<tr>
<td>81262.svs</td>
<td>25974 x 21219</td>
<td>1.54GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Original Real Data</th>
<th>Original User Time</th>
<th>Original System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.467</td>
<td>0.913</td>
<td>0.059</td>
</tr>
<tr>
<td>2</td>
<td>1.700</td>
<td>0.916</td>
<td>0.055</td>
</tr>
<tr>
<td>3</td>
<td>1.515</td>
<td>0.916</td>
<td>0.059</td>
</tr>
<tr>
<td>4</td>
<td>1.699</td>
<td>0.910</td>
<td>0.062</td>
</tr>
<tr>
<td>5</td>
<td>1.571</td>
<td>0.917</td>
<td>0.057</td>
</tr>
<tr>
<td>6</td>
<td>1.836</td>
<td>0.916</td>
<td>0.061</td>
</tr>
<tr>
<td>7</td>
<td>1.553</td>
<td>0.908</td>
<td>0.063</td>
</tr>
<tr>
<td>8</td>
<td>1.773</td>
<td>0.912</td>
<td>0.057</td>
</tr>
<tr>
<td>9</td>
<td>1.623</td>
<td>0.917</td>
<td>0.056</td>
</tr>
<tr>
<td>10</td>
<td>1.624</td>
<td>0.917</td>
<td>0.050</td>
</tr>
<tr>
<td>Total</td>
<td>17.361</td>
<td>9.142</td>
<td>0.579</td>
</tr>
<tr>
<td>Mean</td>
<td>1.736</td>
<td>0.914</td>
<td>0.058</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.276</td>
<td>0.003</td>
<td>0.004</td>
</tr>
<tr>
<td>Image Name</td>
<td>Image Dimensions</td>
<td>Size (GB)</td>
<td>Magnification</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>81259.svs</td>
<td>27972 x 22490</td>
<td>1.76GB</td>
<td>20x</td>
</tr>
<tr>
<td>81283.svs</td>
<td>27972 x 25378</td>
<td>1.98GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Original Real Data</th>
<th>Original User Time</th>
<th>Original System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.044</td>
<td>0.516</td>
<td>0.056</td>
</tr>
<tr>
<td>2</td>
<td>1.097</td>
<td>0.512</td>
<td>0.060</td>
</tr>
<tr>
<td>3</td>
<td>1.140</td>
<td>0.516</td>
<td>0.049</td>
</tr>
<tr>
<td>4</td>
<td>1.275</td>
<td>0.509</td>
<td>0.063</td>
</tr>
<tr>
<td>5</td>
<td>1.121</td>
<td>0.516</td>
<td>0.060</td>
</tr>
<tr>
<td>6</td>
<td>1.282</td>
<td>0.510</td>
<td>0.054</td>
</tr>
<tr>
<td>7</td>
<td>1.217</td>
<td>0.512</td>
<td>0.055</td>
</tr>
<tr>
<td>8</td>
<td>1.295</td>
<td>0.513</td>
<td>0.054</td>
</tr>
<tr>
<td>9</td>
<td>1.210</td>
<td>0.515</td>
<td>0.054</td>
</tr>
<tr>
<td>10</td>
<td>1.097</td>
<td>0.516</td>
<td>0.059</td>
</tr>
<tr>
<td>Total</td>
<td>12.778</td>
<td>5.135</td>
<td>0.564</td>
</tr>
<tr>
<td>Mean</td>
<td>1.278</td>
<td>0.514</td>
<td>0.056</td>
</tr>
</tbody>
</table>

Standard Deviation: 0.280

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Original Real Data</th>
<th>Original User Time</th>
<th>Original System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.465</td>
<td>6.005</td>
<td>0.068</td>
</tr>
<tr>
<td>2</td>
<td>6.832</td>
<td>6.084</td>
<td>0.073</td>
</tr>
<tr>
<td>3</td>
<td>7.093</td>
<td>5.989</td>
<td>0.073</td>
</tr>
<tr>
<td>4</td>
<td>6.800</td>
<td>5.994</td>
<td>0.074</td>
</tr>
<tr>
<td>5</td>
<td>6.780</td>
<td>5.984</td>
<td>0.068</td>
</tr>
<tr>
<td>6</td>
<td>6.780</td>
<td>6.021</td>
<td>0.069</td>
</tr>
<tr>
<td>7</td>
<td>6.735</td>
<td>6.005</td>
<td>0.067</td>
</tr>
<tr>
<td>8</td>
<td>6.775</td>
<td>6.006</td>
<td>0.072</td>
</tr>
<tr>
<td>9</td>
<td>6.756</td>
<td>6.045</td>
<td>0.074</td>
</tr>
<tr>
<td>10</td>
<td>6.719</td>
<td>6.016</td>
<td>0.068</td>
</tr>
<tr>
<td>Total</td>
<td>68.735</td>
<td>60.149</td>
<td>0.706</td>
</tr>
<tr>
<td>Mean</td>
<td>6.874</td>
<td>6.015</td>
<td>0.071</td>
</tr>
</tbody>
</table>

Standard Deviation: 0.233

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Image Dimensions</th>
<th>Size (GB)</th>
<th>Magnification</th>
</tr>
</thead>
<tbody>
<tr>
<td>81257.svs</td>
<td>29970 x 25622</td>
<td>2.15GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Original Real Data</th>
<th>Original User Time</th>
<th>Original System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.741</td>
<td>2.849</td>
<td>0.061</td>
</tr>
<tr>
<td>2</td>
<td>3.702</td>
<td>2.847</td>
<td>0.059</td>
</tr>
<tr>
<td>3</td>
<td>3.670</td>
<td>2.850</td>
<td>0.059</td>
</tr>
<tr>
<td>4</td>
<td>3.887</td>
<td>2.845</td>
<td>0.061</td>
</tr>
<tr>
<td>5</td>
<td>3.594</td>
<td>2.836</td>
<td>0.071</td>
</tr>
<tr>
<td>6</td>
<td>3.632</td>
<td>2.852</td>
<td>0.066</td>
</tr>
<tr>
<td>7</td>
<td>3.397</td>
<td>2.844</td>
<td>0.065</td>
</tr>
<tr>
<td>8</td>
<td>3.538</td>
<td>2.847</td>
<td>0.068</td>
</tr>
<tr>
<td>9</td>
<td>3.620</td>
<td>2.856</td>
<td>0.061</td>
</tr>
<tr>
<td>10</td>
<td>3.598</td>
<td>2.836</td>
<td>0.065</td>
</tr>
<tr>
<td>Total</td>
<td>37.379</td>
<td>28.462</td>
<td>0.636</td>
</tr>
<tr>
<td>Mean</td>
<td>3.738</td>
<td>2.846</td>
<td>0.064</td>
</tr>
</tbody>
</table>

Standard Deviation: 0.374

Total Time Taken: 12.778 GB, 5.135 GB, 0.564 GB
Mean Time Taken: 1.278 GB, 0.514 GB, 0.056 GB
Standard Deviation of Time Taken: 0.280 GB, 0.003 GB, 0.004 GB
## Re-engineered Application Raw Data Results

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Image Dimensions</th>
<th>Size (GB)</th>
<th>Magnification</th>
</tr>
</thead>
<tbody>
<tr>
<td>81275.svs</td>
<td>23976 x 21031</td>
<td>1.41GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

### Time Taken

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Cloud Real Data</th>
<th>Cloud User Time</th>
<th>Cloud System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.997</td>
<td>1.470</td>
<td>0.060</td>
</tr>
<tr>
<td>2</td>
<td>8.012</td>
<td>1.400</td>
<td>0.060</td>
</tr>
<tr>
<td>3</td>
<td>7.846</td>
<td>1.420</td>
<td>0.060</td>
</tr>
<tr>
<td>4</td>
<td>42.930</td>
<td>1.390</td>
<td>0.070</td>
</tr>
<tr>
<td>5</td>
<td>9.041</td>
<td>1.400</td>
<td>0.050</td>
</tr>
<tr>
<td>6</td>
<td>8.542</td>
<td>1.370</td>
<td>0.090</td>
</tr>
<tr>
<td>7</td>
<td>46.037</td>
<td>1.410</td>
<td>0.040</td>
</tr>
<tr>
<td>8</td>
<td>8.789</td>
<td>1.390</td>
<td>0.080</td>
</tr>
<tr>
<td>9</td>
<td>29.940</td>
<td>1.200</td>
<td>0.090</td>
</tr>
<tr>
<td>10</td>
<td>8.330</td>
<td>1.310</td>
<td>0.130</td>
</tr>
<tr>
<td>Total</td>
<td>179.464</td>
<td>13.760</td>
<td>0.730</td>
</tr>
<tr>
<td>Mean</td>
<td>17.946</td>
<td>1.376</td>
<td>0.073</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>15.510</td>
<td>0.074</td>
<td>0.026</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Image Dimensions</th>
<th>Size (GB)</th>
<th>Magnification</th>
</tr>
</thead>
<tbody>
<tr>
<td>81259.svs</td>
<td>27972 x 22490</td>
<td>1.76GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

### Time Taken

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Cloud Real Data</th>
<th>Cloud User Time</th>
<th>Cloud System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.472</td>
<td>1.320</td>
<td>0.120</td>
</tr>
<tr>
<td>2</td>
<td>7.113</td>
<td>1.540</td>
<td>0.070</td>
</tr>
<tr>
<td>3</td>
<td>6.998</td>
<td>1.390</td>
<td>0.080</td>
</tr>
<tr>
<td>4</td>
<td>53.511</td>
<td>1.410</td>
<td>0.060</td>
</tr>
<tr>
<td>5</td>
<td>33.947</td>
<td>1.420</td>
<td>0.050</td>
</tr>
<tr>
<td>6</td>
<td>7.917</td>
<td>1.370</td>
<td>0.090</td>
</tr>
<tr>
<td>7</td>
<td>41.995</td>
<td>1.340</td>
<td>0.160</td>
</tr>
<tr>
<td>8</td>
<td>8.124</td>
<td>1.380</td>
<td>0.080</td>
</tr>
<tr>
<td>9</td>
<td>74.109</td>
<td>1.340</td>
<td>0.130</td>
</tr>
<tr>
<td>10</td>
<td>44.201</td>
<td>1.370</td>
<td>0.090</td>
</tr>
<tr>
<td>Total</td>
<td>285.387</td>
<td>13.880</td>
<td>0.930</td>
</tr>
<tr>
<td>Mean</td>
<td>28.539</td>
<td>1.388</td>
<td>0.093</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>24.415</td>
<td>0.062</td>
<td>0.034</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Name</th>
<th>Image Dimensions</th>
<th>Size (GB)</th>
<th>Magnification</th>
</tr>
</thead>
<tbody>
<tr>
<td>81283.svs</td>
<td>27972 x 25378</td>
<td>1.98GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

### Time Taken

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Cloud Real Data</th>
<th>Cloud User Time</th>
<th>Cloud System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18.179</td>
<td>1.460</td>
<td>0.030</td>
</tr>
<tr>
<td>2</td>
<td>11.715</td>
<td>1.470</td>
<td>0.060</td>
</tr>
<tr>
<td>3</td>
<td>12.563</td>
<td>1.400</td>
<td>0.040</td>
</tr>
<tr>
<td>4</td>
<td>12.358</td>
<td>1.430</td>
<td>0.040</td>
</tr>
<tr>
<td>5</td>
<td>13.265</td>
<td>1.340</td>
<td>0.110</td>
</tr>
<tr>
<td>6</td>
<td>11.546</td>
<td>1.450</td>
<td>0.110</td>
</tr>
<tr>
<td>7</td>
<td>12.951</td>
<td>1.420</td>
<td>0.060</td>
</tr>
<tr>
<td>8</td>
<td>13.951</td>
<td>1.410</td>
<td>0.060</td>
</tr>
<tr>
<td>9</td>
<td>12.856</td>
<td>1.460</td>
<td>0.040</td>
</tr>
<tr>
<td>10</td>
<td>11.733</td>
<td>1.400</td>
<td>0.070</td>
</tr>
<tr>
<td>Total</td>
<td>131.117</td>
<td>14.240</td>
<td>0.620</td>
</tr>
<tr>
<td>Mean</td>
<td>13.112</td>
<td>1.424</td>
<td>0.062</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>1.934</td>
<td>0.039</td>
<td>0.028</td>
</tr>
<tr>
<td>Image Name</td>
<td>Image Dimensions</td>
<td>Size (GB)</td>
<td>Magnification</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>81257.svs</td>
<td>29970 x 25622</td>
<td>2.15GB</td>
<td>20x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. of Execution</th>
<th>Cloud Real Data</th>
<th>Cloud User Time</th>
<th>Cloud System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.707</td>
<td>1.410</td>
<td>0.100</td>
</tr>
<tr>
<td>2</td>
<td>10.299</td>
<td>1.400</td>
<td>0.070</td>
</tr>
<tr>
<td>3</td>
<td>8.434</td>
<td>1.450</td>
<td>0.050</td>
</tr>
<tr>
<td>4</td>
<td>9.032</td>
<td>1.410</td>
<td>0.090</td>
</tr>
<tr>
<td>5</td>
<td>10.779</td>
<td>1.470</td>
<td>0.020</td>
</tr>
<tr>
<td>6</td>
<td>9.538</td>
<td>1.410</td>
<td>0.070</td>
</tr>
<tr>
<td>7</td>
<td>45.509</td>
<td>1.470</td>
<td>0.030</td>
</tr>
<tr>
<td>8</td>
<td>9.932</td>
<td>1.380</td>
<td>0.080</td>
</tr>
<tr>
<td>9</td>
<td>10.506</td>
<td>1.410</td>
<td>0.070</td>
</tr>
<tr>
<td>10</td>
<td>5.244</td>
<td>1.420</td>
<td>0.090</td>
</tr>
<tr>
<td>Total</td>
<td>129.980</td>
<td>14.230</td>
<td>0.670</td>
</tr>
<tr>
<td>Mean</td>
<td>12.998</td>
<td>1.423</td>
<td>0.067</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>11.541</td>
<td>0.030</td>
<td>0.026</td>
</tr>
</tbody>
</table>
Original Application versus Re-Deployed Application Graphs

1. **Original Vs. Re-deployed Application Graph**
   - Image: 81275.svs
   - Graphs show comparisons between Original Real Data and Cloud Real Data.
   - X-axis represents the number of executions.
   - Y-axis represents time taken in seconds.

2. **Original Vs. Re-deployed Application Graph**
   - Image: 81275.svs
   - Graphs show comparisons between Original User Time and Cloud User Time.
   - X-axis represents the number of executions.
   - Y-axis represents time taken in seconds.

3. **Original Vs. Re-deployed Application Graph**
   - Image: 81275.svs
   - Graphs show comparisons between Original System Time and Cloud System Time.
   - X-axis represents the number of executions.
   - Y-axis represents time taken in seconds.
Original Vs. Re-deployed Application
Image: 81259.svs

- Time Taken In Seconds
- Number of Executions

Original Real Data vs Cloud Real Data

Original User Time vs Cloud User Time

Original System Time vs Cloud System Time
Original Vs. Re-deployed Application
Image:81283.svs

Original Real Data  Cloud Real Data

Original User Time  Cloud User Time

Original System Time  Cloud System Time
## Appendix D: Schedule

**Week to week initial project schedule**

<table>
<thead>
<tr>
<th>Week</th>
<th>Activity</th>
<th>Deliverables / Deadlines</th>
</tr>
</thead>
</table>
| 1    | **Begin Background reading (General)**  
**Minimum Aims and requirements**  
**Discuss project scenario (Supervisor Meeting)**  
**Research Seminar (Tuesday)** | **Min aims and requirements 29/01/2010**  
**Read and start research papers’ referencing and reviewing.**  
**Finish background reading 01.02.2010** |
| 2    | **Begin Literature Research**  
- **Literature Search Meeting 03.02.2010**  
- **Literature Search Training 04.02.2010**  
**Finish Background Reading (General)**  
**Start Deeper Specific literature review of: GRIDS**  
**Outline Project Plan** | **Reference List and Review of each reference**  
**Project Plan – For next supervisor meeting 08.02.2010**  
**Keyword list for training session on 4.02.2010** |
| 3    | **Attend Research Seminar 09.02.2010**  
**Produce a Literature List for research on: GRIDS**  
**Tue – Thurs: Start Training on GRID middleware (Globus GT4 SDK)**  
**Create Notes on Methodology Plan**  
**Selecting paper allocation (Research Seminar)**  
**Star Mid-project report**  
**Draft Background reading chapter for report.**  
**Finish literature review on: GRIDS** | **List of selected Literature to review and reference: GRIDS**  
**Finish GRID literature review**  
**Finish GRD Training**  
**Notes on methodology**  
**Research seminar – Selected paper for presentation**  
**Must read Research paper of choice allocated and make Notes**  
**Draft background chapter report (By End of week)** |
| 4    | **Wed – Fri: Start Deeper Literature Review: CLOUD**  
**Produce a Literature List for research on: CLOUD**  
**Tue – Thurs: Start Training on CLOUD middleware (Either OpenNebula or Nimbus)**  
**Revised Project Plan**  
**Problem Description ????**  
**Questionnaire Form submission**  
**Finish literature review on: CLOUD** | **List of selected Literature to review and reference: CLOUD**  
**Revised Project Plan**  
**Submission of Questionnaire Form 19.02.2010**  
**Finish Training CLOUD**  
**Finish CLOUD literature review**  
**Draft Chapter Background reading** |
| 5    | **Fri: Review and reflect on both middleware trained on**  
**Draft chapter on the background reading**  
**Research seminar (Post Seminar Notes)** | **Background reading section of the report (Proper)**  
**Post seminar Notes** |
| 6    | **Mon: Begin analysis of Pathology Application**  
**Tue: Design re-deployment of application**  
**Wed: Talk to the clients (If applicable)**  
**Mid-Project report (Electronic Copy) 05.03.2010**  
**Research seminar – Post seminar Notes** | **Mid –Project Report: 1 Hard copy with header sheet & 1 Copy electronically submitted (sis)**  
**Design plan and analysis**  
**Seminar Presentation** |
| 7    | **Submission 12.03.2010 of:**  
- **Table of contents**  
- **Draft chapter** | **Submission of the TOC, Draft chapter and evaluation.**  
**Presentation** |
<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Evaluation plan</td>
<td></td>
</tr>
<tr>
<td>15.03.2010</td>
<td>Begin Implementation of project application</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completed design of the solution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Research seminar – Post seminar Notes</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Collection of Mid-project report (TBA)</td>
<td></td>
</tr>
<tr>
<td>15.03.2010</td>
<td>1st week: Review and implement software components</td>
<td></td>
</tr>
<tr>
<td>Easter</td>
<td>2nd week: Error checking and Testing</td>
<td></td>
</tr>
<tr>
<td>Vacation</td>
<td>3rd Week: Begin FINAL PROJECT REPORT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4th Week: Scalability and Evaluation of product</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Student workshop presentation (TBA)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Software Modules each week</td>
<td></td>
</tr>
<tr>
<td>19.04.2010</td>
<td>any weekly reports</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Initial Final project report draft</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Presentation</td>
<td></td>
</tr>
<tr>
<td>19.04.2010</td>
<td>Demonstration</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Evaluation Plan and completed product solution</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Complete meeting by 30.04.2010</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Alterations to product</td>
<td></td>
</tr>
<tr>
<td>19.04.2010</td>
<td>Another draft report</td>
<td></td>
</tr>
<tr>
<td>26.04.2010</td>
<td>Evaluation Outcome (Supervisor Meeting)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Continue on to work on reports different sections</td>
<td></td>
</tr>
<tr>
<td>26.04.2010</td>
<td>Notes on workshop presentation – feedback and recommendations</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Alterations to product</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Another Draft Report</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Product produced</td>
<td></td>
</tr>
<tr>
<td>03.05.2010</td>
<td>Create final evaluation of the overall report in comparison to the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>initial project plan and final outcome</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Software product completion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Finalised Final Report ready for submission</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Submission of project report 12.05.2010</td>
<td></td>
</tr>
<tr>
<td>10.05.2010</td>
<td>Electronic submission of the final report 14.05.2010</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Submission of the final project report</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Submit the electronic copy</td>
<td></td>
</tr>
</tbody>
</table>
Appendix E: Evidence of Globus Training

The following screenshots are evidence of doing the Globus GT4 training, using the Sun VirtualBox environment via the install Globus virtual cluster images. The screenshots illustrate the training being used and applied to gain an understanding to the theoretical nature of deploying a grid service.

Figure 1: Invoking the Singleton Resource on Chapter 4 of the Globus GT4 Tutorial material of the Globus Alliance website.

Figure 2: Factory Resource Creation
Figure 3: Chapter 5.9 deploying an end reference to the client

Figure 4: Chapter 6.2 Running the client
### Appendix F: Project Life Cycles

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
</table>
| Waterfall          | Iterative approach to the development of the product. Iterative development likely to meet user requirements through each phase by being highly consistent the customer requirements.                       | - Iterative (logical time)  
- Effective for short low risk projects  
- Quality in testing terms is satisfied with clear documentation | - Each stage has no relevant feedback loop  
- Assumes no time/budget overrun  
- Product available at end |
| V Model            | Verification: Identifying potential conflicts of building product. Does implementation meet the design.  
Validation: correctness and consistency of initial requirements with system. Also eliminates undesired functionality. | - Quality aspect of the system in security/safety are highlighted  
- Similar to the waterfall model enables an easy validation through intermediate stages.  
- Clear that the product will be available when project finishes  
- If the client knows desired requirements this is more appropriate, the V model highlights V&V which assumes stable system requirements. | - Iterative approach  
- Timing/budget overruns  
- Product available at end |
| Spiral Model       | Continuous cycle in the development areas                                                                                                                                                                  | - Changes in the development process  
- Avoids unforeseen situations  
- Clear review of documentation                                                                                                                            | - Due to the high risk product may fail to materialise |
| Evolutionary Development »»»»»» | The intent of this model is to get user and customer feedback and iterate through several releases before the final release.                                                                                           | - Rapid Application Development (Throw away prototyping)  
- User engagement                                                                                                                                          | - Maintenance/Contract problems  
- Timing/end delivery of the product  
- Prototypes non functional aspect cannot be directly executed (security, reliability, safety, etc) |
| Incremental delivery | Provides an early feedback perspective, through workable prototypes.                                                                                                                                              | - Early versions act as the prototypes  
- Real time client feedback, through intractable development of system                                                                                     | - Time/budget overrun, extra requirements introduced  
- Procedure/planning management not clear  
- Development slow on client demands                                                                                                                          |
| Extreme Programming (XP) | Fast paced development architecture, with close client reviews. Quick prototyping to un-clear system requirements enables the influence of the product design from a client’s perspective.                               | - Close customer relation, so client can influence the product  
- Rapid Application Development  
- Functionality alterations to vague development aspects. On spot prototyping  
- Testing plans built before hand for the requirements                                                                                                      | - Tests carried out must succeed to each build stage to be successful.  
- Manage/monitor project progression  
- Documentation/maintenance improving code continuously  
- No clear schedule  
- Customer available for clarification                                                                                                                      |

Software Development Table (Sommerville, 2006)
The software methodology appropriately chosen for this project is the Evolutionary model, focusing in on the Rapid Application Development and the combined incremental delivery approach being a sustainable method to apply. The relevant literature reviewed can be located in (Sommerville, 2006). Although there is not a tabular break down of the information in the book section it has been constructed in view of a clear separation for individual relevant details to be apparent.

<table>
<thead>
<tr>
<th>Name</th>
<th>Small Projects</th>
<th>Larger Projects</th>
<th>Fast/Short term Products</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall</td>
<td>X</td>
<td></td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Verification &amp; Validation</td>
<td>X</td>
<td></td>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Spiral Model</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolutionary Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incremental delivery</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extreme Programming (XP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The software methodology appropriately chosen for this project is the Evolutionary model, focusing in on the Rapid Application Development and the combined incremental delivery approach being a sustainable method to apply. The relevant literature reviewed can be located in (Sommerville, 2006). Although there is not a tabular break down of the information in the book section it has been constructed in view of a clear separation for individual relevant details to be apparent.
Appendix G: Using the Cloud Virtual Infrastructure Manager – Nimbus

Commands and steps to access the cloud infrastructure

Logical Step Sequence to using the Pathology Application

1. Log into the University of Leeds server, through the School of computing Servers(cslin-gps.csunix.comp.leeds.ac.uk)
2. Log into Testgrid 3
3. Log into the Cloud TestBed
4. Change directory to the workspace where the Virtual image is located
5. Check if there any virtual images running (clients)
6. Deploy a virtual image of the globus image
7. SSH into the globus image
8. Change directory to the location of the grid service resource
9. Build the service
10. Deploy the service
11. Source the binaries and globus variables location to the directory
12. Compile the client.java file
13. Start the globus container with no security options
14. Invoke the service by java the client and PathologyService
   a. Wait for the client application to ask for user input
      i. Enter the grid proxy password
   b. Client then calls the PathologyService
   c. Invokes the globusrun-ws GRAM command to the service
      i. Changes directory to the location of the custom rsl file
   d. Runs the service
   e. Outputs the command line output and the out.xml file invoked at the St James university hospital server into two new files
   f. Executes the grid proxy destroy command to close the certificate
   g. Ends the program

Command list to execute the program

Logging onto the cloud
1. ssh sc07bss@cslin-gps.csunix.comp.leeds.ac.uk
   a. When prompted for the password enter: ******
2. Change the command line interface to normal bash.
   a. Type “bash” into the command line
3. ssh sc07bss@testgrid3.leeds.ac.uk or ssh 129.11.144.67
   a. When prompted for the password enter: rlr2xulrjz
4. Change to the super user of the nimbus infrastructure manager
   a. Enter: “su nimbus”
   b. Password: halo
5. Change directory to the nimbus user account
   a. Enter: cd ~
6. Change directory to Bhupinder
   a. Enter: cd Bhupinder/

Starting the virtual image
1. Check if the cloud has any virtual images running
   a. Enter: sudo xm list
2. Start the virtual image called globus.deb.cfg
   a. Enter: sudo xm create ./globus.deb.cfg
3. Again check that the Globus image has been deployed
   a. Enter: sudo xm list
4. Now change user and access the globus virtual image that you have setup
   a. ssh root@10.0.2.1
5. When prompted for a password
   a. Enter: password

Exiting the globus image to keep persistent data
6. When exiting the virtual image and shutdown the image on the cloud
   a. Enter: “shutdown –P –h now”
      i. Effectively shutdown the virtual image in power state halt mode and do it now
7. Check that the virtual image has shutdown and is now not deployed on the cloud TestBed
   a. Enter: sudo xm list
8. If no machine named globus.deb is registered then the virtual image is no longer operating

Invoking the service for the pathology application

As the virtual image is accessed but the user it automatically places the user in the root directory of the virtual image. To access the “globus” user account then “cd” into that directory.

1. Change directory to the PathologyService folder
   a. Enter: cd /home/globus/PathologyService
2. To start and run the service enter the following command in sequential order of completion

There are five essential steps needed to access writing and deploying a WSRF Web Service.
Reference (Globus_Alliance, Globus toolkit, 2010)

These are as follows:
1. Define the service’s interface. This is done with WSDL
2. Implement the service. This is done with Java.
3. Define the deployment parameters. This is done with WSDD and JNDI
4. Compile everything and generate a GAR file. This is done with Ant
5. Deploy service. This is also done with a GT4 tool

Creating a GAR FILE
Creating a GAR file is a pretty complex task which involves the following:

- Processing the WSDL file to add missing pieces (such as bindings)
- Creating the stub classes from the WSDL
- Compiling the stubs classes
- Compiling the service implementation
- Organize all the files into a very specific directory structure
**Command sequence to start and use the service**

1. Build the service uses either a or b
   a. `.globus-build-service.sh -d <service base directory> -s <service's WSDL file>
      i. `.globus-build-service.sh \
         -d org/globus/examples/services/core/first/ \
         -s schema/examples/PathologyService_instance/Pathology.wsdl
   b. `.globus-build-service.sh first

2. If everything works then a .gar file will be created in the location of the PathologyService folder.

3. Deploy the gar file
   a. `globus-deploy-gar $EXAMPLES_DIR/org_globus_examples_services_core_first.gar`

4. To undeploy the .gar file
   a. `globus-undeploy-gar org_globus_examples_services_core_first`

5. Source all the globus classpath libraries
   a. `source $GLOBUS_LOCATION/etc/globus-devel-env.sh`

6. Compile the client

7. javac \
   -classpath ./build/stubs/classes/:$CLASSPATH \
   org/globus/examples/clients/PathologyService_instance/ClientC.java

8. Start the globus container
   a. `globus-start-container --nosec &`

9. Invoke the client application to invoke the resource deployed to start the pathology application
   a. `java \
      -classpath ./build/stubs/classes/:$CLASSPATH \
      org.globus.examples.clients.PathologyService_instance.Client \
      http://10.0.2.1:8080/wsrf/services/examples/core/first/PathologyService`

10. If everything works well then the service should produce two output files in the location of a java folder named “stderr” and “stdout”. These two files will contain the command line output of the program and the out.xml file retrieved from the server.
Appendix H: Development Source Code

Pathocall.java

The following code listed above displays the string variables in the code are statically bind-ed to the local parameter that the nucleus detection program uses.

```java
import java.io.*;

// Bhupinder Singh Soorma
// Uses the runtime method to call and execute a local program
// that is in the same directory as the application invoking it
//
// Version 1.1

public class pathocall {
    public static void main (String args[]) throws Exception{
        Process tp = null;
        BufferedReader ins = null;
        BufferedReader output = null;
        System.out.println("Setup variables");
        //Try to call the other program
        try {
            String url = 
"http://129.11.64.252/Research_5/3D_Histo/Liver/Cirrhosis/Liver_3/H%26E/81275.svs ";

            //String Variables
            int zoom = 20;   int minx = 10000;   int miny = 10000;
            int width = 500; int height = 500;   int channel = 1;
            String stain = "H";   String chev = "> "; String outfile = " out.xml";
            System.out.println("./nuc_det_server " + url + zoom + " " + minx + " " + miny + " " + width + " " + height + " " + channel+ " " + stain + " " + chev + "out.xml");

            // Process output
            tp = Runtime.getRuntime().exec("./nuc_det_server " + url + zoom + " " + minx + " " + miny + " " + width + " " + height + " " + channel+ " " + stain + " " + chev + "out.xml");

            ins = new BufferedReader(new InputStreamReader(tp.getInputStream()));
            String x;
            int stop = 0;
            while ((x = ins.readLine()) != null) {
                System.out.println(x);
            }
            System.out.println("nDONE!!");
        } catch (IOException e) {
            System.err.println("ERROR");
            e.printStackTrace();
        }
    }
}
```
import java.io.*;

// Bhupinder Singh Soorma
// Uses the runtime method to call and execute a local program
// that is in the same directory as the application invoking it
// Version 2.0

public class pathology {
    public static void main (String args[]) throws Exception {

        // Setup the system variables for the java runtime event
        
        String proxyinit = "grid-proxy-init";
        String globusrun = "globusrun-ws -submit -S -f nucdetserver.rsl";
        String proxydestroy = "grid-proxy-destroy";
        System.out.println("System Vairable Setup
" Serial.println("System Vairable Setup"));

        // Use try block to run:
        // Proxy
        // Deploy the application through GRAM
        // Destroy-proxy
        try{
            call(proxyinit);
            call(globusrun);
            call(proxydestroy);
            System.out.println("nJob Complete!");
        }
        catch (IOException e){
            System.out.println("NOT SO GOOD");
            e.printStackTrace();
        }{}

        // Using the Java Runtime Event call the GRAM command in the bash environment
        public static void call (String command) throws Exception {
            Process tp = null;
            BufferedReader ins = null;
            try {
                tp = Runtime.getRuntime().exec(command);
                ins = new BufferedReader(new InputStreamReader(tp.getInputStream()));
                String x;
                while ((x = ins.readLine()) != null) {
                    System.out.println(x);
                }
                System.out.println("nRunning: "+command);
            }
            catch(IOException e) {
                System.err.println("Main ERROR!!");
                e.printStackTrace();
            }{}
        }
    }
}
package org.globus.examples.clients.MathService_instance;

import org.apache.axis.message.addressing.Address;
import org.apache.axis.message.addressing.EndpointReferenceType;
import org.globus.examples.stubs.MathService_instance.MathPortType;
import org.globus.examples.stubs.MathService_instance.GetValueRP;
import org.globus.examples.stubs.MathService_instance.service.MathServiceAddressingLocator;
import java.io.*;

//Adapted By Bhupinder Soorma

public class ClientC {
    public static void main(String[] args) throws Exception {
        MathServiceAddressingLocator locator = new MathServiceAddressingLocator();
        try {
            String serviceURI = args[0];
            // Create endpoint reference to service
            EndpointReferenceType endpoint = new EndpointReferenceType();
            endpoint.setAddress(new Address(serviceURI));
            MathPortType math = locator.getMathPortTypePort(endpoint);
            // CODE ADDED
            // ADAPTED HERE!!
            System.out.println("Client output");
            String p = "/usr/local/globus-4.0.8/bin/grid-proxy-init";
            System.out.println(p);
            Process tp = null;
            BufferedReader ins = null;
            try {
                tp = Runtime.getRuntime().exec(p);
                ins = new BufferedReader(new InputStreamReader(tp.getInputStream()));
                String x;
                while ((x = ins.readLine()) != null) {
                    System.out.println(x);
                }
            } catch (IOException e) {
                System.err.println("Grid Proxy did not work");
                e.printStackTrace();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
The Resource Specification Language code illustrate the command line arguments normally supplied to the nucleus detection program converted to a xml format, used and supplied through the `globusrun-ws` GRAM command.

```xml
<job>
  <executable>/home/globus/java /nuc_det_server</executable>
  <argument>http://129.11.64.252/Research_5/3D_Histo/Liver/Cirrhosis/Liver_3/H%26E/81275.svs</argument>
    <argument>20</argument>
    <argument>10000</argument>
    <argument>10000</argument>
    <argument>500</argument>
    <argument>500</argument>
    <argument>1</argument>
    <argument>H&amp;E</argument>
  <argument>out.xml</argument>
  <stdout>/home/globus/java /nuc_det_server /stdout</stdout>
  <stderr>/home/globus/java /nuc_det_server /stderr</stderr>
</job>
```
Sample-workspace.xml source code

The `<def:location>` displays the alterations made for the virtual workspace to identify and keep a persistent state.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<VirtualWorkspace
    xmlns="http://www.globus.org/2008/06/workspace/metadata"
    xmlns:def="http://www.globus.org/2008/06/workspace/metadata/definition"
    xmlns:log="http://www.globus.org/2008/06/workspace/metadata/logistics"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <!-- name is an arbitrary URI -->
    <name>http://localhost/globusTest</name>

    <log:logistics>
        <log:networking>
            <log:nic>
                <log:name>eth0</log:name>
                <log:ipConfig>
                    <log:acquisitionMethod>AllocateAndConfigure</log:acquisitionMethod>
                </log:ipConfig>
                <!--
                    The association string allows you to associate the NIC with specific networking contexts (a common example is a string which resolves to which network to bridge the virtual NIC to; a simple scheme would be 'public' vs. 'private'. Another example is VPN). A site will advertise the available associations in the workspace factory resource properties (to query with the sample client, use the factoryrp option).
                    -->
                <log:association>public</log:association>
            </log:nic>
        </log:networking>
    </log:logistics>

    <def:definition>
        <def:requirements>
            <jsdl:CPUArchitecture>
                <jsdl:CPUArchitectureName>x86_64</jsdl:CPUArchitectureName>
            </jsdl:CPUArchitecture>
            <def:type>Xen</def:type>
            <def:version>3.3.1</def:version>
        </def:requirements>

        <def:diskCollection>
            <def:rootVBD>
                <!--
                    Relative path names like in this example will be resolved relative to the deployment node's local image repository
                    -->
            <def:location>file://deb.img</def:location>
            <def:location>scp://testgrid3.leeds.ac.uk/home/nimbus/bhupinder/globus.deb.img</def:location>
            <def:mountAs>sda1</def:mountAs>
            <def:permissions>ReadWrite</def:permissions>
        </def:rootVBD>
    </def:diskCollection>
</def:definition>
</VirtualWorkspace>
```
MathService.java (PathologyService.java) Source Code

The following was used as a template to host the new methods the services.

```java
package org.globus.examples.services.core.first.impl;

import java.rmi.RemoteException;
import org.globus.wsrf.Resource;
import org.globus.wsrf.ResourceProperties;
import org.globus.wsrf.ResourceProperty;
import org.globus.wsrf.ResourcePropertySet;
import org.globus.wsrf.impl.ReflectionResourceProperty;
import org.globus.wsrf.impl.SimpleResourcePropertySet;
import org.globus.examples.stubs.MathService_instance.AddResponse;
import org.globus.examples.stubs.MathService_instance.SubtractResponse;
import org.globus.examples.stubs.MathService_instance.GetValueRP;

//My Variables added
import java.io.*;
import java.util.*;

public class MathService implements Resource, ResourceProperties {
    /* Resource Property set */
    private ResourcePropertySet propSet;
    /* Resource properties */
    private int value;
    private String lastOp;
    /* Constructor. Initializing RPs */
    public MathService() throws RemoteException {
        /* Create RP set */
        this.propSet = new SimpleResourcePropertySet(
            MathQNames.RESOURCE_PROPERTIES);
        /* Initialize the RP's */
        try {
            ResourceProperty valueRP = new ReflectionResourceProperty(
                MathQNames.RP_VALUE, "Value", this);
            this.propSet.add(valueRP);
            setValue(0);
            ResourceProperty lastOpRP = new ReflectionResourceProperty(
                MathQNames.RP_LASTOP, "LastOp", this);
            this.propSet.add(lastOpRP);
            setLastOp("NONE");
        }
    }
    /* Get/Setters for the RPs */
    public int getValue() {
        return value;
    }
    public void setValue(int value) {
        this.value = value;
    }
    public String getLastOp() {
        return lastOp;
    }
    public void setLastOp(String lastOp) {
        this.lastOp = lastOp;
    }
}
```
/* Remotely-accessible operations */

public AddResponse add(int a, String command) throws RemoteException {
    value += a;
    lastOp = "ADDITION";
    //CODE ADDED
    System.out.println("\n Calling MathService GRAM commands...");
    String proxyinit = "grid-proxy-init";
    String globusrun = "globusrun -ws -submit -S -f nucdetserver.rsl";
    String proxydestroy = "grid-proxy-destroy";
    System.out.println("System Vairable Setup\n");
    //Use try block to run:
    // Proxy
    // Deploy the application through GRAM
    // Destroy-proxy
    try{
        call(proxyinit);
        call(globusrun);
        call(proxydestroy);
        //tp = Runtime.getRuntime().exec("mv stderr cmdout");
        System.out.println("\nTry Block Complete");
    } catch (IOException e){
        System.out.println("Problem with MathService.add --> Try Block");
e.printStackTrace();
    }
    return new AddResponse();
}

//NEW CODE CALL METHOD
public static void call (String command) throws Exception {
    Process tp = null;
    BufferedReader ins = null;
    try {
        tp = Runtime.getRuntime().exec(command);
        ins = new BufferedReader(new InputStreamReader(tp.getInputStream()));
        String x;
        while ((x = ins.readLine()) != null) {
            System.out.println(x);
        }
        System.out.println("\nRunning: "+command);
    }
    catch (IOException e) {
        System.err.println("Problem with MathService.call method --> Try Block");
e.printStackTrace();
    }
}

public SubtractResponse subtract(int a) throws RemoteException {
    value -= a;
    lastOp = "SUBTRACTION";
    return new SubtractResponse();
}

public int getValueRP(GetValueRP params) throws RemoteException {
    return value;
}

/* Required by interface ResourceProperties */
public ResourcePropertySet getResourcePropertySet() {
    return this.propSet;
}